Empowering Deep Learning on Graphs: Tackling Over-smoothing and Over-squashing

Choi, Jeongwhan

Department of Artificial Intelligence

Graduate School

Yonsei University

Empowering Deep Learning on Graphs: Tackling Over-smoothing and Over-squashing

Advisors Cho, Sung-Bae and Park, Noseong

A Dissertation Submitted
to the Department of Artificial Intelligence
and the Committee on Graduate School
of Yonsei University in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Choi, Jeongwhan

July 2025

Empowering Deep Learning on Graphs:
Tackling Over-smoothing and Over-squashing

This Certifies that the Dissertation of Choi, Jeongwhan is Approved

Committee Chair	**Cho, Sung-Bae**
Committee Member	**Park, Noseong**
Committee Member	**Lee, Dongha**
Committee Member	**Hwang, Seong Jae**
Committee Member	**Shin, Won-Yong**

Department of Artificial Intelligence

Graduate School

Yonsei University

July 2025

Acknowledgement

First of all, I would like to express my deepest gratitude to my advisors, Noseong Park and Professor Sung-Bae Cho. Noseong Park not only guided the direction of my research but also taught me the attitude of a researcher. He provided careful guidance and encouragement for the completion of my thesis, and whenever I felt overwhelmed with research, he offered new perspectives and gave me courage. Noseong Park's academic passion and far-sighted insights have been a great foundation for my growth as a researcher. I also extend my deep gratitude to my other advisor, Sung-Bae Cho. Even after Noseong Park moved to KAIST, he continued to provide valuable comments for my research and offered support so that I could focus on my research. He reviewed my papers and provided precious feedback that greatly contributed to the quality of my thesis.

I sincerely thank the committee members: Dongha Lee, Seong Jae Hwang, and Won-Yong Shin. Through their thoughtful questions and constructive advice, I enhanced the quality of my thesis. Thank you for providing valuable insights about my research despite your busy schedules.

I also thank my research collaborators at Yonsei University who worked together on each chapter of this thesis. Chapter 2 was developed through discussions and research with Seoyoung Hong, under the guidance of my advisors. Chapter 3 was accomplished through collaboration with Kookjin Lee and Nathaniel Trask's comments and feedback, as well as co-first-author work with Hyowon Wi, which allowed me to conduct research in truly diverse fields through collaboration. Chapters 4 and 5 were also written through collaboration with Hyowon Wi and Sumin Park under the guidance of my advisors. Through these experiences, I came to realize the value of collaboration.

I would like to thank my lab colleagues who shared the research journey with me. The time spent with the BigDyL members — Jinsung Jeon, Taeyong Kong, Jayoung Kim, Sheoyeon Jin, Jeehyun Hwang, Jihyeon Hyeong, Jaehoon Lee, Minju Jo, Seunghyun Cho, Seungji Kook, Woojin Cho, Seoyoung Hong, Chaejeong Lee, Seojin Kim, Jaehyun Park, Yehjin Shin, Hyowon Wi, Seonkyu Lim, Sumin Park, and Youn-Yeol Yu — represents the most special memories of my Ph.D. journey. Having colleagues who shared everything from research discussions to daily concerns and encouraged each other helped me endure difficult times. I will always remember the late nights in the lab, working on experiments and discussing the results.

I want to thank Jinsung Jeon, with whom I spent the most time and who encouraged each other as we began our Ph.D. studies together. I am grateful and delighted to have learned what collaborative research truly means through conversations and discussions with Seoyoung Hong and Hyowon Wi, with whom I had the most research discussions. I would also like to express my gratitude to my junior colleagues Chaejeong Lee and Yehjin Shin, who helped each other to complete papers through collaboration and listened to my suggestions.

Through these experiences, I gained the ability to conduct research in a collaborative manner. I would like to extend my thanks to Youn-Yeol Yu and Seonkyu Lim. I had the opportunity to collaborate with Youn-Yeol Yu, which allowed me to join AI4Science. This collaboration provided me with the chance to research graph machine learning for scientific purposes. Additionally, through collaboration with Seonkyu Lim, I was able to conduct research related to actual industry demands. Although I regret not being able to mention experiences with everyone, I received great inspiration from the passionate attitudes of all my lab colleagues.

Most of all, I want to thank and express love to my family. To my parents, who watched over my long academic journey and gave me the support that allowed me to focus on my research. To my younger sister, who has always been there for me and has cheered me on through every milestone and challenge of this journey.

I apologize for not being able to list all the names of those who helped me complete this thesis, but I sincerely thank everyone who provided even the smallest help along the way.

Finally, I thank God, from whom all blessings flow and makes everything possible, for His unconditional love and guidance throughout this journey.

August 2025
Choi, Jeongwhan

Contents

List of Figures

List of Tables

Abstract

Empowering Deep Learning on Graphs: Tackling Over-smoothing and Over-squashing

Choi, Jeongwhan
Department of Artificial Intelligence
Graduate School
Yonsei University

Graphs are ubiquitous data structures, prevalent from e-commerce platforms to complex molecular structures, abstracting the rich interactions between individual entities. The increasing reliance on graph-structured data in diverse real-world applications necessitates the development of effective representations for graph nodes. Recent advancements in deep learning on graphs, particularly through Graph Neural Networks (GNNs), have led to significant breakthroughs in various fields by enabling the learning of these representations. However, deep learning on graphs still faces fundamental challenges. Firstly, GNNs operate by propagating information through the connected neighborhood structure of the graph. While increasing the number of GNN layers allows for the capture of higher-order neighborhood information, this iterative aggregation process can lead to the over-smoothing problem, where the representations of different nodes become increasingly similar and lose their distinctiveness. This phenomenon can be understood by viewing the diffusion process inherent in GNNs. Secondly, real-world graphs often contain critical long-range dependencies and interactions. Nodes in these graphs face limitations in effectively receiving information from distant nodes due to the over-squashing problem. Over-squashing arises as a bottleneck, where a node's fixed-dimensional feature vector must compress an exponentially growing amount of information originating from its expanding receptive field in deeper layers.

This thesis proposes novel approaches to tackle these two critical issues: over-smoothing and over-squashing. To mitigate over-smoothing, we propose new architectures for both GNNs and Transformers. For GNNs, we draw inspiration from reaction-diffusion equations to design a novel layer incorporated into our proposed method, GREAD, which prevents feature convergence by balancing smoothing and sharpening. We also explore over-smoothing in the context of Transformers, interpreting their self-attention mechanism as a graph filter, and propose Graph Filter-based Self-Attention as a novel mechanism to alleviate this problem in Transformer models. To address over-squashing in GNNs, we present two distinct methods. Recognizing limitations in existing graph rewiring techniques, our first method, PANDA, introduces an expanded width-aware message passing framework designed to overcome topological bottlenecks and over-squashing. Our second method, Fractal-Inspired Message

Passing with Fractal Nodes, uses a fractal concept to improve message passing and enhance the expressive power of GNNs, thereby facilitating more effective propagation of global information and mitigating the decay of signal propagation associated with over-squashing.

As the scale and complexity of global data continue to grow, the relationships abstracted by graph structures are becoming increasingly complex. The research presented in this thesis aims to develop more effective graph representations and enhance the scalability of deep learning models by effectively addressing over-smoothing and over-squashing. This will enable these models to process complex graph relationships more efficiently and make a positive impact across a wide range of domains.

Key Words : graph machine learning, graph neural network, over-smoothing, over-squashing

Chapter 1

Introduction

Graphs are ubiquitous data structures that capture complex relationships in diverse domains — from social networks [2] and molecular structures [3] to recommendation systems [4, 5] and transportation networks [6, 7]. As our world becomes increasingly interconnected, the ability to effectively learn from graph-structured data has emerged as a fundamental challenge in machine learning. Graph Neural Networks (GNNs) have proven successful in our approach to this challenge, demonstrating remarkable results on numerous applications. However, despite their widespread adoption, GNNs face fundamental limitations that constrain their depth, expressiveness, and ability to capture long-range dependencies.

1.1 Motivation and Challenges

The success of deep learning in computer vision and natural language processing has been largely driven by its ability to stack many layers. However, directly applying this principle to graphs reveals unique challenges:

1.1.1 The Over-smoothing Problem

Unlike images or text, where deeper networks generally improve performance, GNNs suffer from a different phenomenon. As we add more layers, node representations converge to indistinguishable values, and the features of the nodes we want to learn become similar [8, 9]. This *over-smoothing* problem can be understood through the lens of graph signal processing — each layer acts as a low-pass filter, progressively removing high-frequency information until all nodes become similar.

The GNN layer performs neighbor integration, which smooths out neighboring nodes to make them similar. This process makes it difficult for the network to distinguish between different nodes as the number of layers increases. This fundamental limitation has led most

practical GNNs to be limited to shallow structures of 2 or 3 layers, limiting their expressive power.

1.1.2 The Over-squashing Problem

A second critical challenge emerges when GNNs attempt to propagate information across distant nodes. The *over-squashing* phenomenon occurs when exponentially growing receptive fields are compressed into fixed-size node representations [10], creating information bottlenecks. This is particularly challenging for tasks requiring long-range dependencies, such as molecular property prediction or social network analysis.

To address this issue, previous studies have proposed methods to add optimal edges using graph rewiring methods [11, 12, 13, 14]. However, changing the structure of the original graph through graph rewiring methods can fundamentally ignore the graph structure. We need to find a way to address over-squashing from the perspective of the model architecture, rather than relying on the rewiring method.

1.1.3 Two Distinct Challenges

These problems represent different challenges with different underlying causes:

- Over-smoothing is a phenomenon that occurs when local smoothing is excessive. Excessive neighbor averaging causes nearby nodes to become indistinguishable, even when the graph is well-connected.
- The phenomenon of over-squashing emerges from inadequate global propagation, wherein structural bottlenecks impede the effective propagation of distant information, regardless of local smoothing behaviors.

These challenges, therefore, require fundamentally different solutions: over-smoothing calls for mechanisms that preserve features during local aggregation, while over-squashing requires methods that enable efficient long-range information flow.

1.2 Thesis Contributions

This thesis presents a comprehensive treatment of these challenges through both theoretical analysis and practical solutions. Our contributions span multiple architectures and domains:

1.2.1 Addressing Over-smoothing

We develop two complementary approaches to mitigate over-smoothing:

1. **GREAD: Graph Neural Reaction-Diffusion Networks** (Chapter 2): Drawing inspiration from physics and biology, we redesign the graph convolution process through

reaction-diffusion equations. Unlike graph convolutions, which can be interpreted as a diffusion process leading to over-smoothing, reaction-diffusion systems can maintain stable, non-uniform patterns. We introduce 7 reaction types, including a novel Blurring-Sharpening (BS) mechanism, and demonstrate that GREAD can effectively balance local smoothing with feature sharpening, enabling much deeper architectures.

2. **GFSA: Graph Filter-based Self-Attention** (Chapter 3): Recognizing that over-smoothing affects not only GNNs but also Transformers, we reinterpret self-attention through the lens of graph signal processing. By designing self-attention as a learnable graph filter that adaptively combines low and high-frequency components, GFSA addresses over-smoothing while maintaining computational efficiency. Our approach achieves consistent improvements across six diverse domains, from graph classification to image classification and language modeling.

1.2.2 Addressing Over-squashing

We propose two novel frameworks that tackle over-squashing without compromising graph structure:

1. **PANDA: Expanded Width-Aware Message Passing** (Chapter 4): Rather than modifying graph topology through rewiring — which can destroy domain-specific information—PANDA selectively expands the hidden dimensions of bottleneck nodes. By identifying nodes with high centrality and enabling variable-width message passing, PANDA maintains constant signal propagation even under severe topological bottlenecks.

2. **Message Passing with Fractal Nodes: Multi-scale Graph Representations** (Chapter 5): Inspired by the fractal nature of real-world networks, we introduce auxiliary "fractal nodes" that capture subgraph-level information at multiple scales. These nodes act as information highways, enabling efficient long-range propagation while preserving local graph structure. Our approach achieves performance comparable to graph Transformers while maintaining the linear complexity of message passing.

1.3 Impact and Significance

The methods developed in this thesis have both theoretical and practical implications:

- **Theoretical Insights**: Our thesis offers new perspectives on information flow in graphs by connecting graph neural networks to diverse theoretical foundations, including reaction-diffusion systems, signal processing, and fractal geometry. Rather than empirical improvements, we methodically redesign existing graph neural networks and self-attention mechanisms through principled physical and mathematical inductive

biases. These connections offer a deeper understanding of fundamental limitations and provide a guide for future architectural innovations.

- **Practical Applications**: By enabling deeper and more expressive graph networks, our methods unlock new possibilities in drug discovery, social network analysis, and scientific computing. The improved ability to capture long-range dependencies is crucial for understanding complex systems, such as protein interactions or climate networks.
- **Broader Impact**: As graph-structured data becomes increasingly prevalent — ranging from knowledge graphs to biological networks — the ability to effectively learn from such data becomes critical. This thesis makes a significant contribution to the field by providing essential tools for the next generation of graph machine learning and graph-based AI systems.

1.4 Thesis Organization

This thesis is structured to provide both theoretical foundations and practical solutions. We tackle the research problems of over-smoothing and over-squashing in each chapter, as shown in Table 1-1. For both challenges, we cover the content and conclude as follows:

- **Chapters 2 and 3** address the over-smoothing challenge through GREAD (reaction-diffusion networks) and GFSA (graph-filter self-attention), respectively.
- **Chapters 4 and 5** tackle over-squashing via PANDA (width-aware message passing) and Fractal Nodes (multi-scale representations).
- **Chapter 6** concludes with a summary of insights, limitations, and directions for future research.

Table. 1-1: Organization of the thesis

Challange	**Research Problem**	**Chapter**
Over-smoothing	How can we mitigate over-smoothing in GNNs and Transformers?	Chapters 2 and 3
Over-squashing	How can we mitigate over-squashing in GNNs without relying on rewiring methods?	Chapters 4 and 5

Chapter 2

Over-smoothing I

GNNs are one of the most popular research topics in deep learning. GNN methods have typically been designed using the graph signal processing theory. In particular, diffusion equations have been widely used for designing the core processing layer of GNNs, and therefore, they are inevitably vulnerable to the notorious over-smoothing problem. Recently, a couple of papers paid attention to reaction equations in conjunction with diffusion equations. However, they all consider limited forms of reaction equations. To this end, we present a reaction-diffusion equation-based GNN method that considers all popular reaction equations in addition to one special reaction equation we designed. To our knowledge, our paper is one of the most comprehensive studies on reaction-diffusion equation-based GNNs. In our experiments with 9 datasets and 28 baselines, our method, called GREAD, outperforms them in most cases. Further synthetic data experiments show that it mitigates the over-smoothing problem and works well for various homophily rates.

The materials in this chapter have been published in Choi et al. [15].

2.1 Motivation

Graphs are a useful data format that occurs frequently in real-world applications, e.g., computer vision and graphics [16], molecular chemistry inference [17], recommendation systems [18, 19], drug discovery [20, 21], traffic forecasting [7], and so forth. With the rise of graph-based data, graph neural networks (GNNs) are attracting much attention these days. However, there have been fierce debates on the neural network architecture of GNNs [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36].

Recently, many proposed methods have been designed based on the diffusion concept. Many recent GNN methods that rely on low-pass filters fall into this category. Although they have shown non-trivial successes in many tasks, it is still unclear whether it is an optimal direction for designing GNNs.

In Table 2-1, we compare recent methods. Most of them rely on diffusion processes, while

Table. 2-1: A comparison table of existing methods. '△' means that it corresponds to a specific type of reaction only.

Model	Diffusion	Reaction	Continuous-time
GCN	O	X	X
FA-GCN	O	△	X
GPR-GNN	O	△	X
ACM-GCN	O	△	X
CGNN	O	X	O
GRAND	O	X	O
BLEND	O	X	O
GREAD	O	O	O

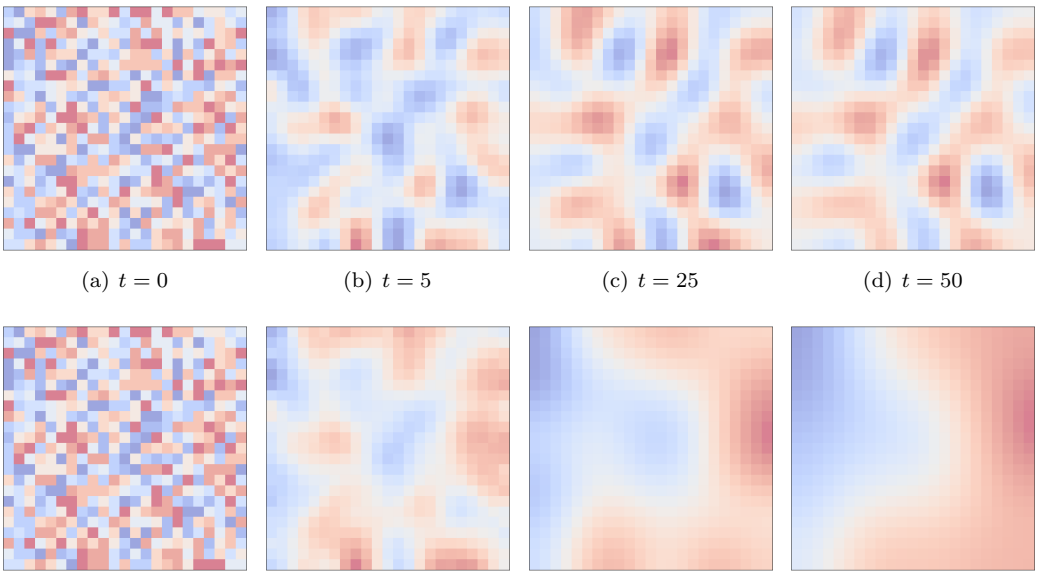
some of them (i.e., FA-GCN [37], GPR-GNN [27], and ACM-GCN [32]) partially utilize reaction processes. Those three methods, however, utilize limited forms of the reaction processes. In this regard, there do not exist any methods that fully consider diverse forms of reaction processes — we consider 7 reaction processes.

To this end, we propose the concept of graph neural reaction-diffusion network (GREAD), which is one of the most generalized architectures since we consider both the diffusion and reaction processes. Reaction-diffusion equations are physical models that can be used when i) substances are diffused over space and time, and ii) they can sometimes react to each other. Whereas diffusion processes smooth node features on a graph out, reaction-diffusion processes lead to many local clusters that are also known as Turing patterns [38] (see Figure 2-1). Since it is natural that nodes on a graph also constitute local clusters (in terms of class labels), we conjecture that reaction-diffusion equations are suitable for GNNs.

Our proposed model, GREAD, consists of three parts: an encoder, a reaction-diffusion layer, and an output layer (cf. Equations (2.6) to (2.8)). The reaction-diffusion layer has seven different types in its core part as shown in Equation (2.10): i) Fisher (F), ii) Allen-Cahn (AC), iii) Zeldovich (Z), iv) Blurring-sharpening (BS), v) Source (S), vi) Filter Bank (FB), and vii) Filter Bank* (FB*). The first three reaction-diffusion equations are widely used in many natural science domains, e.g., biology, combustion, and so on, and the last three are used by some recent GNN methods [39, 40, 32]. In particular, the blurring-sharpening (BS) equation is designed by us and marks the best accuracy in many cases.

For our experiments, we consider 6 heterophilic and 3 homophilic datasets — heterophilic (resp. homophilic) means that neighboring nodes tend to have different (resp. similar) classes. We also compare our method with a comprehensive set of 28 baselines, which covers early to recent GNNs. Our contributions can be summarized as follows:

- We design a reaction-diffusion layer that incorporates seven types of reaction equations, including one type, called BS, proposed by us.



(a) $t = 0$ (b) $t = 5$ (c) $t = 25$ (d) $t = 50$

(e) $t = 0$ (f) $t = 5$ (g) $t = 25$ (h) $t = 50$

Figure. 2-1: An illustrative comparison between the diffusion equation in Equation (2.11) (bottom) and our proposed blurring-sharpening (reaction-diffusion) equation in Equation (2.14) (top) on a grid graph with one-dimensional node features. The diffusion equation causes the problem of over-smoothing, while the reaction-diffusion seeks a balance between smoothing and sharpening.

- We carefully integrate the seven reaction equation types into our GNN method and customize its overall architecture for better accuracy. For instance, we use a soft adjacency matrix generating method, which shows a synergistic effect with the reaction-diffusion layer.
- We consider a comprehensive set of 9 datasets and 28 baselines. Our method marks the best accuracy in many cases.

2.2 Preliminaries & Related Work

We first describe the meaning of the reaction-diffusion equation and various important GNN designs, followed by neural ordinary differential equations.

2.2.1 Reaction-Diffusion Equations

Reaction–diffusion equations are typically used to model the spatial and temporal change of the concentration of one or more chemical substances, i.e., substances are transformed into

each other via local chemical reactions and spread out over a surface in space via diffusion. They are also frequently observed in other fields, such as biology, geology, physics (neutron diffusion theory), and ecology. In the field of graph machine learning, diffusion (resp. reaction) processes are typically carried out by applying low-pass (resp. high-pass) filters to graphs, which also corresponds to image blurring (resp. sharpening).

2.2.2 Graph Neural Networks

Notation Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$. The nodes are associated with a feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$, where $|\mathcal{V}|$ denotes the number of nodes and F denotes the number of input features. $\mathbf{A}^{raw} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix, where $\mathbf{A}^{raw}_{[i,j]}$ means the (i,j)-th element. The nodes are labelled by the index $i \in \mathcal{V}$, and the one-hop neighborhood of each node is denoted as $\mathcal{N}_i$. The symmetric normalized Laplacian matrix, a commonly used feature aggregation matrix in GNNs, is defined as $\mathbf{L} := \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}^{raw}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{A}$, where the diagonal degree matrix of $\mathbf{A}^{raw}$ is $\mathbf{D}$, and $\mathbf{A} := \mathbf{D}^{-1/2}\mathbf{A}^{raw}\mathbf{D}^{-1/2}$ is the symmetric normalized adjacency matrix — note that $\mathbf{A} \in [0,1]^{|\mathcal{V}| \times |\mathcal{V}|}$.

Graph Representation Learning GNNs [22, 23, 41, 25, 42] have many variants and applications. We focus on a brief introduction to the representation learning for nodes in supervised or semi-supervised classification tasks. Most existing approaches follow the message-passing framework constructed by stacking layers that propagate and aggregate node features.

The neighbor aggregation used in many existing GNNs implicitly exploits homogeneity and often fails to generalize to non-homogeneous graphs. Many existing GNNs operate as low-pass graph filters [43] that smooth features over the graph topology, which produces similar representations and, as a result, similar predictions for neighboring nodes [44, 8, 45]. Various GNNs were proposed to improve performance in low-homophily settings [46, 47, 28, 27, 48, 30, 32, 36, 49, 31] and alleviate the over-smoothing problem. [50, 26, 51, 33].

Diffusion on Graphs and Continuous GNNs The diffusion on graphs has recently been actively used in various applications [52, 53], including data clustering [54, 55], image processing [56, 57, 58], and so on. It is to apply the following diffusion process to the feature matrix $\mathbf{X}$ of a graph:

$$\frac{d\mathbf{X}(t)}{dt} := div(\mathbf{A}(\mathbf{X}(t)))\nabla\mathbf{X}(t)) = -\mathbf{L}\mathbf{X}(t), \tag{2.1}$$

where div and ∇ are the divergence and the gradient operators, respectively. The initial features are evolved under the diffusion process to have the final representation. The diffusion

equation and its unit-step Euler discretization can be defined as follows:

$$\mathbf{X}(t+1) = \mathbf{X}(t) - \mathbf{L}\mathbf{X}(t) = (\mathbf{I} - \mathbf{L})\mathbf{X}(t). \tag{2.2}$$

This is similar to GCN [22] where the following augmented diffusion process with a weight matrix $\mathbf{W}$ and a nonlinear activation σ is used:

$$\sigma((\mathbf{I} - \mathbf{L})\mathbf{X}(t)\mathbf{W}). \tag{2.3}$$

From this perspective, several papers have proposed continuous-depth GNNs [59, 5, 60, 40, 61] inspired by the graph diffusion equation. One recent work is GRAND [34], which parameterizes the diffusion equation on graphs with a neural network. BLEND [35] used a non-Euclidean diffusion equation (known as Beltrami flow) to solve a joint positional feature space problem. These approaches contribute to non-trivial improvements in graph machine learning. In this work, we extend the diffusion to the reaction-diffusion equation.

2.2.3 Neural Ordinary Differential Equations (NODEs)

Neural ordinary differential equations (NODEs) [62] solve the initial value problem (IVP), which involves a Riemann integral problem, to calculate $\mathbf{h}(t_{i+1})$ from $\mathbf{h}(t_i)$:

$$\mathbf{h}(t_{i+1}) = \mathbf{h}(t_i) + \int_{t_i}^{t_{i+1}} f(\mathbf{h}(t_i), t; \theta_f) dt, \tag{2.4}$$

where the neural network parameterized by θ_f approximates the time-derivative of $\mathbf{h}$, i.e., $\dot{\mathbf{h}} \stackrel{\text{def}}{=} \frac{d\mathbf{h}(t)}{dt}$. We rely on various ODE solvers to solve the integral problem, from the explicit Euler method to the 4th order Runge–Kutta (RK4) method and the Dormand–Prince (DOPRI) method [63]. For instance, the Euler method is as follows:

$$\mathbf{h}(t+h) = \mathbf{h}(t) + \tau \cdot f(\mathbf{h}(t)), \tag{2.5}$$

where τ, which is usually smaller than 1, is a pre-configured step size. Equation (2.5) is identical to a residual connection when $h = 1$ and therefore, NODEs are a continuous generalization of residual networks.

2.3 Proposed Method

After describing an overview of our method, we describe its detailed designs, followed by its training algorithm. The theoretical and empirical complexity analyses are in Section 2.4 and Section 2.5.4, respectively.

2.3.1 Overview of GREAD

Given a graph $\mathcal{G}$ with its node feature matrix $\mathbf{X}$, its symmetric normalized Laplacian matrix $\mathbf{L}$, and its symmetric normalized adjacency $\mathbf{A}$, the overall architecture of GREAD can be written as follows — instead of $\mathbf{A}$, we can also use a generated soft adjacency matrix $\tilde{\mathbf{A}}$, which will be described in the next subsection:

$$\mathbf{H}(0) = \mathbf{e}(\mathbf{X}) \qquad \text{(Encoding layer)}, \tag{2.6}$$

$$\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T \mathbf{f}(\mathbf{H}(t))dt \qquad \text{(Reac.-diff. layer)}, \tag{2.7}$$

$$\hat{\mathbf{y}} = \mathbf{o}(\mathbf{H}(T)) \qquad \text{(Output layer)}, \tag{2.8}$$

where $\mathbf{f}(\mathbf{H}(t)) := \frac{d\mathbf{H}(t)}{dt} = -\alpha\mathbf{L}\mathbf{H}(t) + \beta\mathbf{r}(\mathbf{H}(t))$ is in the reaction-diffusion form. $\mathbf{r}(\mathbf{H})$ is a reaction term, and α and β are trainable parameters to (de-)emphasize each term. $\mathbf{e}$ is an encoder embeds the node feature matrix $\mathbf{X}$ into an initial hidden state $\mathbf{H}(0)$. We then evolve the initial hidden state to $\mathbf{H}(T)$ via the reaction-diffusion equation of $\mathbf{f}$. The function $\mathbf{o}$ is an output layer for a downstream task, e.g., node classification. In particular, β can be either a scalar or a vector parameter, where the scalar setting means that we apply the same reaction process to all nodes. In the vector setting, we apply different reaction processes with different coefficients to nodes.

The encoder $\mathbf{e}$ has a couple of fully-connected layers with rectified linear unit (ReLU) activations. The output layer $\mathbf{o}$ is typically a fully-connected layer, followed by a softmax activation for classification in our experiments.

In particular, we consider almost all existing reaction terms for $\mathbf{r}$, which is different from existing works that do not thoroughly consider them. In this perspective, our work is the most comprehensive study on reaction-diffusion GNNs to our knowledge. In the following subsection, we also show that some choices of the reaction term correspond to other famous models — in other words, some other famous models are special cases of GREAD.

2.3.2 Soft Adjacency Matrix Generation

Given a graph $\mathcal{G}$, one can use its original symmetric normalized adjacency matrix $\mathbf{A} \in [0,1]^{|\mathcal{V}|\times|\mathcal{V}|}$ for $\mathbf{f}(\mathbf{H}(t))$. However, we also provide the method to generate a soft adjacency matrix, denoted $\tilde{\mathbf{A}} \in [0,1]^{|\mathcal{V}|\times|\mathcal{V}|}$ — we use $\tilde{\mathbf{L}}$ to denote the Laplacian counterpart of $\tilde{\mathbf{A}}$. Our soft adjacency matrix plays a crucial role in learning diffusivity. Our reaction-diffusion layer uses the soft adjacency matrix to learn the diffusivity.

To generate such soft adjacency matrices, we use the scaled dot product method [64]:

$$\tilde{\mathbf{A}}_{[i,j]} := \text{softmax}\Big(\frac{(\mathbf{W}_K\mathbf{H}_i)^T\mathbf{W}_Q\mathbf{H}_j}{d_K}\Big), \tag{2.9}$$

where $\tilde{\mathbf{A}}_{[i,j]}$ means the (i,j)-th element of $\tilde{\mathbf{A}}$, $\mathbf{W}_K$ and $\mathbf{W}_Q$ are trainable parameters, and d_K is the scale factor. $\mathbf{H}_i, \mathbf{H}_j$ are trainable embedding vectors of nodes i, j.

2.3.3 Reaction-diffusion Layer

Equation (2.7) is our method's main processing layer, called the reaction-diffusion layer. Given the definition of $\mathbf{f}$, '$-\mathbf{LH}(t)$' is a diffusion term that corresponds to the heat equation describing the spread of heat over $\mathcal{G}$ and has been used widely by various GNNs [59, 5, 34]. It is known that the diffusion term causes the problem of over-smoothing, which means that the last hidden states of nodes become too similar when only the diffusion processing is applied too much. To this end, many models prefer shallow architectures that do not cause the over-smoothing problem [25, 22] or use heuristic methods to prevent it [51, 65, 26, 66, 67, 68, 69].

In our case, we prevent the over-smoothing problem by adding the reaction term $\mathbf{r}$ and solving Equation (2.7) with ODE solvers [63]. In other words, our reaction-diffusion layer is continuous, which is yet another distinguishing point of our method since many other models are based on discrete layers [22, 37, 27, 28, 41]. We consider the following options for $\mathbf{r}$:

$$\mathbf{r}(\mathbf{H}(t)) := \begin{cases} \mathbf{H}(t) \odot (1 - \mathbf{H}(t)), \text{ if Fisher (F)} \\ \mathbf{H}(t) \odot (1 - \mathbf{H}(t)^{\circ 2}), \text{ if Allen-Cahn (AC)} \\ \mathbf{H}(t) \odot (\mathbf{H}(t) - \mathbf{H}(t)^{\circ 2}), \text{ if Zeldovich (Z)} \\ (\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t), \text{ if Blurring-Sharpening (BS)} \\ \mathbf{H}(0), \text{ if Source Term (ST)} \\ \mathbf{LH}(t), \text{ if Filter Bank (FB)} \\ \mathbf{LH}(t) + \mathbf{H}(t), \text{ if Filter Bank* (FB*)} \end{cases} \tag{2.10}$$

where '$\odot$' means the Hadamard product, and '$\circ 2$' means the Hadamard power.

The first three reaction terms — F, AC, and Z — are widely used in various domains. For instance, F is used to describe the spreading of biological populations [70], and AC is used for describing the phase separation process in multi-component alloy systems, which includes order-disorder transitions [71]. Z is a generalized equation that describes the phenomena that occur in combustion theory [72]. The last BS is specially designed by us for GNNs, which we will describe shortly. ST is a case where the initial hidden state is added as a reaction term [39]. ST is not theoretically a reaction process, but we consider it part of our model since their goals are the same, i.e., alleviating the notorious over-smoothing problem. FB means high-pass filters that correspond to reaction processes. By adding a high-pass filter, our reaction-diffusion layer acts like a filter bank holding the low and high-pass filters. FB* is a reaction term that also considers the identity channel $\mathbf{H}(t)$.

Blurring-Sharpening (BS) Given the reaction-diffusion layer in Equation (2.7), the proposed blurring-sharpening (BS) process, whose time-derivative of $\mathbf{H}(t)$ will be defined in Equation (2.14), is to perform the blurring (diffusion) and the sharpening (reaction) operations alternately in the layer. We show that our proposed blurring-sharpening process

reduces to a certain form of the reaction-diffusion process. Many GNNs can be generalized to the following blurring (or diffusion) process, i.e., the low-pass graph convolutional filtering for blurring. We also use the same blurring operation at first:

$$\begin{aligned}\mathbf{B}(t+h) &= \mathbf{H}(t) - \tilde{\mathbf{L}}\mathbf{H}(t),\\ &\Rightarrow \mathbf{H}(t) + (\tilde{\mathbf{A}} - \mathbf{I})\mathbf{H}(t),\\ &\Rightarrow \tilde{\mathbf{A}}\mathbf{H}(t).\end{aligned} \tag{2.11}$$

We then propose to apply the following high-pass graph convolutional filtering or sharpening process to $\mathbf{B}(t+h)$. In other words, there is a sharpening process following the above blurring process in a layer as follows:

$$\begin{aligned}\mathbf{H}(t+h) &= \mathbf{B}(t+h) + \tilde{\mathbf{L}}(\mathbf{B}(t+h)),\\ &\Rightarrow \tilde{\mathbf{A}}\mathbf{H}(t) + \mathbf{L}(\tilde{\mathbf{A}}\mathbf{H}(t)),\\ &\Rightarrow \tilde{\mathbf{A}}\mathbf{H}(t) + (\mathbf{I} - \tilde{\mathbf{A}})\tilde{\mathbf{A}}\mathbf{H}(t),\\ &\Rightarrow 2\tilde{\mathbf{A}}\mathbf{H}(t) - \tilde{\mathbf{A}}^2\mathbf{H}(t),\\ &\Rightarrow (2\mathbf{I} - \tilde{\mathbf{A}})\tilde{\mathbf{A}}\mathbf{H}(t),\\ &\Rightarrow (\mathbf{I} + \mathbf{L})(\mathbf{I} - \mathbf{L})\mathbf{H}(t),\\ &\Rightarrow \mathbf{H}(t) - \mathbf{L}^2\mathbf{H}(t),\\ &\Rightarrow \mathbf{H}(t) - (\mathbf{I} - \tilde{\mathbf{A}})^2\mathbf{H}(t),\\ &\Rightarrow \mathbf{H}(t) - (\mathbf{I} - \tilde{\mathbf{A}})\mathbf{H}(t) + (\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t),\\ &\Rightarrow \mathbf{H}(t) - \tilde{\mathbf{L}}\mathbf{H}(t) + (\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t).\end{aligned} \tag{2.12}$$

Therefore, we can derive the following difference equation:

$$\mathbf{H}(t+h) - \mathbf{H}(t) = -\tilde{\mathbf{L}}\mathbf{H}(t) + (\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t). \tag{2.13}$$

After taking the limit of $h \to 0$ and adding the coefficients α, β,

$$\mathbf{f}(\mathbf{H}(t)) := \frac{d\mathbf{H}(t)}{dt} = -\alpha\tilde{\mathbf{L}}\mathbf{H}(t) + \beta(\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t), \tag{2.14}$$

which is a reaction-diffusion equation where $\mathbf{r}(\mathbf{H}(t)) := (\tilde{\mathbf{A}} - \tilde{\mathbf{A}}^2)\mathbf{H}(t)$. Therefore, our proposed method, BS, uses the reaction-diffusion layer in Equation (2.7) with the specific time-derivative definition of Equation (2.14).

2.3.4 Training Algorithm

We use Algorithm 2.1 to train our proposed model. The full training process minimizes the cross-entropy loss:

$$\mathcal{L} := \sum_{i}^{n} \mathbf{y}_i^T \log \hat{\mathbf{y}}_i, \tag{2.15}$$

Algorithm 2.1 How to train our proposed GREAD

Input: Training data D_{train}, Validating data D_{val}, Maximum iteration number max_iter
Initialize model parameters θ;
$k \leftarrow 0$;
while $k < max\_iter$ **do**
  Construct a mini-batch B from D_{train}
  Train θ with Equation (2.15) and B;
  Validate and update the best parameters θ^* with D_{val}
  $k \leftarrow k + 1$;
end
return θ^*;

where $\mathbf{y}_i$ is the one-hot ground truth vector of i-th training sample, and $\hat{\mathbf{y}}_i$ is its inference outcome by our model.

2.3.5 Comparison with GNNs

When ST is used, GREAD is analogous to GCNII from the perspective that both methods inject the initial hidden state. GREAD, FA-GCN, and GPR-GNN differ in how to utilize low and high-pass filters. FA-GCN learns edge-level aggregation weights as in GAT but allows negative weights. GPR-GNN uses learnable weights that can be both positive and negative for feature propagation. These enable FA-GCN and GPR-GNN to adapt to heterophilic graphs and handle both high- and low-frequency parts of graph signals. However, GREAD-BS sharpens low-pass filtered signals following our developed reaction-diffusion system. GREAD-BS also adaptively adjusts each term.

We also compare with some continuous-time GNN models. CGNN can be derived from the reaction-diffusion layer in Equation (2.7) with $\mathbf{L}$ by setting $\mathbf{f}$ with $\mathbf{r}(\mathbf{H}(t)) := \mathbf{H}(0)$ and using a weight parameter $\mathbf{W}$:

$$\mathbf{f}(\mathbf{H}(t))_{\text{CGNN}} := -\mathbf{L}\mathbf{H}(t) + \mathbf{H}(t)\mathbf{W} + \mathbf{H}(0). \tag{2.16}$$

The linear GRAND model corresponds to using only our diffusion process:

$$\mathbf{f}(\mathbf{H}(t))_{\text{GRAND}} := -\tilde{\mathbf{L}}\mathbf{H}(t) = -(\mathbf{I} - \tilde{\mathbf{A}})\mathbf{H}(t). \tag{2.17}$$

We note that two continuous models can not capture high-frequency parts. In particular, GRAND does not use any reaction term.

2.4 Computational Complexity

The space complexity of GREAD is dominated by evaluating the soft adjacency matrix in Equation (2.9), which is $\mathcal{O}(|\mathcal{E}|\dim(\mathbf{H}))$, where $|\mathcal{E}|$ is the number of edges and $\dim(\mathbf{H})$ is the size of the hidden dimension.

Table. 2-2: Benchmark dataset properties and statistics

Dataset	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
Classes	5	5	5	5	5	5	6	7	3
Features	1,703	1,703	1,703	932	2,089	235	1,433	3,703	500
Nodes	183	251	183	7,600	5,201	2,277	2,708	3,327	19,717
Edges	279	466	277	26,752	198,353	31,371	5,278	4,552	44,324
Hom. ratio	0.11	0.21	0.30	0.22	0.22	0.23	0.81	0.74	0.80

We also analyze the time complexity of the reaction-diffusion layer in Equation (2.7). Our proposed model has different complexities depending on the reaction term r in Equation (2.10).

If we set the adjacency matrix and β to OA and SC respectively, the time complexity of the one-step GREAD-BS computation becomes $\mathcal{O}(n_\tau(|\mathcal{E}|+|\mathcal{E}_2|)\dim(\mathbf{H})+|\mathcal{E}|d_{\max})$, where n_τ, $|\mathcal{V}|$, and $d_{\max}$ are the number of steps in $[0,T]$, the number of nodes, and the maximum degree of all nodes, respectively. Given that $\mathbf{A}$ is sparse, we can calculate $\mathbf{A}^2$ in $\mathcal{O}(|\mathcal{E}|d_{\max})$ because $d_{\max}$ is equal to the maximum number of non-zeroes in any row of $\mathbf{A}$. The sparse matrix multiplication of $\mathbf{A}^2\mathbf{H}(t)$ takes $\mathcal{O}(|\mathcal{E}_2|d_{\max})$, where $|\mathcal{E}_2| = \frac{1}{2}\sum_{v\in\mathcal{V}}|\mathcal{N}_2|(v)$. The computational complexity of the one-step GREAD-F computation is $\mathcal{O}(n_\tau(|\mathcal{E}|+\dim(\mathbf{H})^k))\dim(\mathbf{H}))$, where $k=1$. In the case of GREAD-AC and GREAD-Z, their k values are 2 and 3, respectively. The computational complexities of the one-step GREAD-ST, GREAD-FB, and GREAD-FB* are $\mathcal{O}(n_\tau|\mathcal{E}|\dim(\mathbf{H})+|\mathcal{E}|d_{\max})$.

2.5 Experiments

First, we compare our method with other baselines for node classification tasks. We then discuss the ability to mitigate over-smoothing on a synthetic graph and show the experiment with different heterophily levels on other synthetic graphs. Our code is available at `https://github.com/jeongwhanchoi/GREAD`.

2.5.1 Node Classification on Real-world Datasets

Real-world Datasets We now evaluate the performance of GREAD and existing GNNs on a variety of real-world datasets. We consider 6 heterophilic datasets with low homophily ratios used in [46]: i, ii) Chameleon, Squirrel [73], iii) Film [2], iv, v, vi) Texas, Wisconsin, and Cornell from WebKB. We also test on 3 homophilic graphs with high homophily ratios: i) Cora [74], ii) CiteSeer [75], iii) PubMed [76]. Table 2-2 summarizes the number/size of nodes, edges, classes, features, and the homophily ratio. We use the dataset splits provided in [46]. We report the mean and standard deviation accuracy after running each experiment with 10 fixed train/val/test splits.

Baselines We use a comprehensive set of baselines classified into the following four groups:

1. In the first group of baselines, we consider classical GNN methods: ChebNet [24], GCN [22], GAT [23], GraphSAGE [41], and SGC [25].

2. The next group includes the GNN methods designed for heterophilic settings: MixHop [47], Geom-GCN [46], H2GCN [28], FA-GCN [37], GPR-GNN [27], WRGAT [77], GGCN [29], LINKX [30], GloGNN [31] and ACM-GCN [32].

3. The third group has GNN methods tackling the over-smoothing problem: PairNorm [51], JKNet [50], GCNII [26], and GCON [33].

4. The last group contains continuous-time GNN methods: GDE [78], CGNN [39], GRAND [34], BLEND [35], ACMP [79], Sheaf [36], and GRAFF [49].

Experimental Settings The following software and hardware environments were used for all experiments: Ubuntu 18.04 LTS, Python 3.9.12, PyTorch 1.11.0, PyTorch Geometric 2.0.4, torchdiffeq 0.2.3, Numpy 1.22.4, Scipy 1.8.1, Matplotlib 2.2.3, CUDA 11.3, and NVIDIA Driver 465.19, and i9 CPU, and NVIDIA RTX 3090. We performed 10 repetitions on the train/valid/test splits taken from [46] and strictly followed their evaluation protocol. For all data sets, we used the largest connectivity component (LCC) except for Citeseer. We use the dropout only in the encoder network and the output layer. We refer to the dropout in the encoder as 'input dropout' and the dropout in the output layer as 'dropout'.

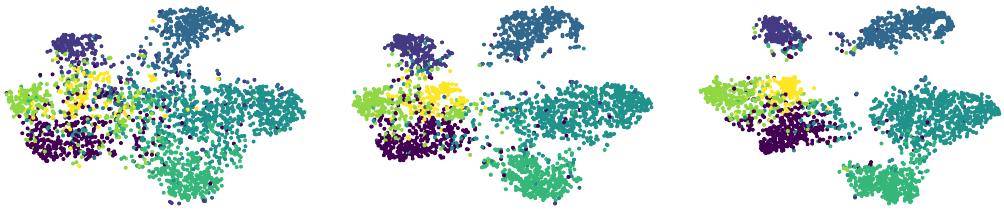
For our method, we test with the following hyperparameter configurations: we train for 200 epochs using the Adam optimizer. We fine-tune our model within the hyperparameter search space in Table 2-6 of Section 2.7.1. Our hyperparameter search used the method of W&B Sweeps [80] with a standard random search with 500 counts. We introduce the best hyperparameter configuration in Tables 2-9 to 2-15 of Section 2.7.2. If a baseline's accuracy is known and its experimental environments are identical to ours, we use the officially announced accuracy. If not, we execute a baseline using its official codes and the hyperparameter search procedures based on their suggested hyperparameter ranges.

Experimental Results GREAD-BS is ranked at the top with an average ranking of 1.56. GREAD-BS shows a significantly higher ranking than GloGNN and others. In Figure 2-2, we visualize the hidden node features at each ODE time step of Equation (2.7), and the reaction-diffusion processes of GREAD lead to local clusters after several steps.

Table 2-3 presents the detailed classification performance. As reported, our method marks the best accuracy in all cases except for Squirrel and Citeseer. GloGNN and Sheaf show comparable accuracy values from time to time. However, there are no existing methods that are as stable as GREAD-BS. For example, GCNII shows reasonably high accuracy in homophilic datasets, but not heterophilic ones. Sheaf shows the best or the second-best place

Table. 2-3: Results on real-world datasets: mean ± std. dev. accuracy for 10 different data splits. We show the best three methods in red (first), blue (second), and purple (third).

Dataset	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
MLP	80.81±±4.75	85.29±±3.31	81.89±±6.40	36.53±±0.70	28.77±±1.56	46.21±±2.99	87.16±±0.37	74.02±±1.90	75.69±±2.00
GCN	55.14±±5.16	51.76±±3.06	60.54±±5.30	27.32±±1.10	53.43±±2.01	64.82±±2.24	86.98±±1.27	76.50±±1.36	88.42±±0.50
ChebNet	78.37±±6.04	79.02±±3.18	75.68±±6.94	34.13±±1.09	36.43±±1.17	58.64±±1.64	85.45±±1.58	75.07±±1.25	89.00±±0.46
GAT	52.16±±6.63	49.41±±4.09	61.89±±5.05	27.44±±0.89	40.72±±1.55	60.26±±2.50	86.33±±0.48	76.55±±1.23	87.30±±1.10
GraphSAGE	82.43±±6.14	81.18±±5.56	75.95±±5.01	34.23±±0.99	41.61±±0.74	58.73±±1.68	86.90±±1.04	76.04±±1.30	88.45±±0.50
SGC	58.10±±4.20	55.29±±4.28	60.00±±3.59	27.20±±1.52	33.00±±1.97	42.45±±3.82	86.12±±1.44	76.01±±1.31	86.90±±1.32
MixHop	77.84±±7.73	75.88±±4.90	73.51±±6.34	32.22±±2.34	43.80±±1.48	60.50±±2.53	87.61±±0.85	76.26±±1.33	85.31±±0.61
Geom-GCN	66.76±±2.72	64.51±±3.66	60.54±±3.67	31.59±±1.15	38.15±±0.92	60.00±±2.81	85.35±±1.57	78.02±±1.15	89.95±±0.47
FA-GCN	82.43±±6.89	82.94±±7.95	79.19±±9.79	34.87±±1.25	42.59±±0.79	55.22±±3.19	87.21±±1.43	76.87±±1.56	87.45±±0.61
GPR-GNN	78.38±±4.36	82.94±±4.21	80.27±±8.11	34.63±±1.22	31.61±±1.24	46.58±±1.84	87.95±±1.18	77.13±±1.67	87.54±±0.38
H2GCN	84.86±±7.23	87.65±±4.98	82.70±±5.28	35.70±±1.00	36.48±±1.86	60.11±±2.15	87.87±±1.20	77.11±±1.57	89.49±±0.38
WRGAT	83.62±±5.50	86.98±±3.78	81.62±±3.90	36.53±±0.77	48.85±±0.78	65.24±±0.87	88.20±±2.26	76.81±±1.89	89.29±±0.38
GGCN	84.86±±4.55	86.86±±3.29	85.68±±6.63	37.54±±1.56	55.17±±1.58	71.14±±1.84	87.95±±1.05	77.14±±1.45	89.15±±0.37
LINKX	74.60±±8.37	75.49±±5.72	77.84±±5.81	36.10±±1.55	61.81±±1.80	68.42±±1.38	84.64±±1.13	73.19±±0.99	87.86±±0.77
GloGNN	84.32±±4.15	87.06±±3.53	83.51±±4.26	37.35±±1.30	57.54±±1.39	69.78±±2.42	88.31±±1.13	77.41±±1.65	89.62±±0.35
ACM-GCN	87.84±±4.40	88.43±±3.22	85.14±±6.07	36.28±±1.09	54.40±±1.88	66.93±±1.85	87.91±±0.95	77.32±±1.70	90.00±±0.52
PairNorm	60.27±±4.34	48.43±±6.14	58.92±±3.15	27.40±±1.24	50.44±±2.04	62.74±±2.82	85.79±±1.01	73.59±±1.47	87.53±±0.44
JKNet	62.70±±8.34	53.14±±5.22	59.72±±4.60	29.25±±1.37	39.78±±1.72	52.63±±3.90	86.48±±1.04	75.99±±1.28	87.23±±0.55
GCNII	77.57±±3.83	80.39±±3.40	77.86±±3.79	37.44±±1.30	38.47±±1.58	63.86±±3.04	88.37±±1.25	77.33±±1.48	90.15±±0.43
GCON-GCN	85.40±±4.20	87.80±±3.30	84.30±±4.80	34.65±±0.61	33.30±±1.57	48.08±±2.16	87.40±±1.82	76.46±±1.70	87.71±±0.35
GCON-GAT	82.20±±4.70	85.70±±3.60	83.20±±7.00	35.85±±0.84	34.45±±1.08	48.31±±1.53	86.96±±1.73	76.20±±2.12	87.73±±0.41
CGNN	71.35±±4.05	74.31±±7.26	66.22±±7.69	35.95±±0.86	29.24±±1.09	46.89±±1.66	87.10±±1.35	76.91±±1.81	87.70±±0.49
GDE	74.05±±6.96	79.80±±5.62	82.43±±7.07	35.36±±1.31	35.94±±1.91	47.76±±2.08	87.22±±1.41	76.21±±2.11	87.80±±0.38
GRAND	75.68±±7.25	79.41±±3.64	82.16±±7.09	35.62±±1.01	40.05±±1.50	54.67±±2.54	87.36±±0.96	76.46±±1.77	89.02±±0.51
BLEND	83.24±±4.65	84.12±±3.56	85.95±±6.82	35.63±±1.01	43.06±±1.39	60.11±±2.09	88.09±±1.22	76.63±±1.60	89.24±±0.42
ACMP	86.20±±0.30	86.10±±0.40	85.40±±0.70	34.44±±4.44	52.65±±2.23	52.63±±2.28	86.38±±3.79	76.52±±1.84	87.54±±0.57
Sheaf	85.05±±5.51	89.41±±4.74	84.86±±4.71	37.81±±1.15	56.34±±1.32	68.04±±1.58	86.90±±1.13	76.70±±1.57	89.49±±0.40
GRAFF	88.38±±4.53	87.45±±2.94	83.24±±6.49	36.09±±0.81	54.52±±1.37	71.08±±1.75	87.61±±0.97	76.92±±1.70	88.95±±0.52
GREAD-BS	88.92±±3.72	89.41±±3.30	86.49±±7.15	37.90±±1.17	59.22±±1.44	71.38±±1.53	88.57±±0.66	77.60±±1.81	90.23±±0.55
GREAD-F	89.73±±4.49	86.47±±4.84	86.49±±5.13	36.72±±0.66	46.16±±1.44	65.20±±1.40	88.39±±0.91	77.40±±1.54	90.09±±0.31
GREAD-AC	85.95±±2.65	86.08±±3.56	87.03±±4.95	37.21±±1.10	45.10±±2.11	65.09±±1.08	88.29±±0.67	77.38±±1.53	90.10±±0.36
GREAD-Z	87.30±±5.68	86.29±±4.32	85.68±±5.41	37.01±±1.11	46.25±±1.72	62.70±±2.30	88.31±±1.10	77.39±±1.90	90.11±±0.27
GREAD-ST	81.08±±5.67	86.67±±3.01	86.22±±5.98	37.66±±0.90	45.83±±1.40	63.03±±1.32	88.47±±1.19	77.25±±1.47	90.13±±0.36
GREAD-FB	86.76±±5.05	87.65±±3.17	86.22±±5.85	37.40±±0.55	50.83±±2.27	66.05±±1.21	88.01±±1.34	77.28±±1.73	90.07±±0.45
GREAD-FB*	87.03±±3.97	88.04±±1.63	85.95±±5.64	37.70±±0.51	50.57±±1.52	65.83±±1.10	88.01±±0.80	77.42±±1.93	90.08±±0.46



(a) $t = 0$ (b) $t = 1$ (c) $t = 3.5$

Figure. 2-2: The snapshots of the evolution process of the node feature at various ODE time points in GREAD for Cora. Different colors correspond to different ground truth classes.

in merely two cases. While GREAD-BS is the best method overall, GREAD-F is the best method for Texas and is the second-best for Cornell. GREAD-AC marks the best accuracy on Cornell.

Ablation Studies We conduct ablation studies on the soft adjacency matrix generation. GREAD can use both the original symmetric normalized adjacency matrix, denoted as OA, and the soft adjacency matrix, denoted as SA. We compare both options. As reported in Table 2-4, SA increases the model accuracy in Cornell and Film.

Next, we also perform the ablation study on β. β can be either a scalar parameter (denoted as SC) or a learnable vector parameter (denoted as VC). We compare them in Table 2-5. VC shows effectiveness for most cases. The VC setting creates a reaction-diffusion process rich enough to classify nodes, as shown in Figure 2-2.

Sensitivity w.r.t. the Terminal Integration Time T By varying T in Equation (2.7), we investigate how the model accuracy changes. The detailed results are in Figure 2-3. In Chameleon, GREAD-BS achieves the highest mean test accuracy at $T = 1.9$.

Sensitivity w.r.t. the ODE Step Size τ Figure 2-4 shows the mean test accuracy by varying the step size τ of RK4. In Chameleon, GREAD-BS shows stable test accuracy at all step sizes, whereas in Cora, GREAD-BS tends to show higher accuracy with larger step sizes.

2.5.2 Over-smoothing and Dirichlet Energy

The Dirichlet Energy We can analyze the degree of over-smoothing [81, 8] from the perspective of the Dirichlet energy [33, 82]. The Dirichlet energy $E(\mathbf{H}, \mathbf{A}^{raw})$ on the node hidden feature $\mathbf{H}$ of an undirected graph $\mathcal{G}$ is defined as follows:

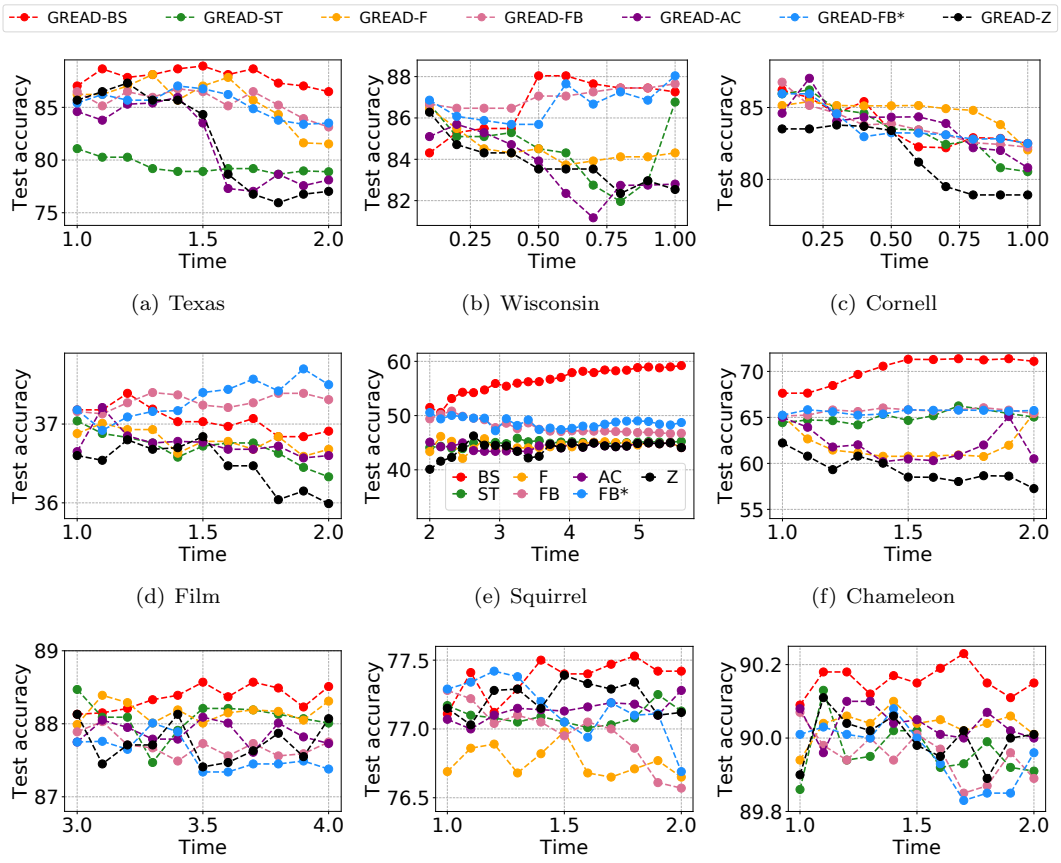
$$E(\mathbf{H}, \mathbf{A}^{raw}) = \frac{1}{N} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbf{A}^{raw}_{[i,j]} ||\mathbf{H}_i - \mathbf{H}_j||^2, \tag{2.18}$$

Table. 2-4: Ablation study on soft adjacency matrix

Dataset	**A**	GREAD-BS	GREAD-F	GREAD-AC	GREAD-Z	GREAD-ST	GREAD-FB	GREAD-FB*
Texas	OA	**88.92±±3.72**	86.49±±4.69	85.41±±5.16	**87.30±±5.68**	**81.08±±5.67**	**86.76±±5.05**	**87.03±±3.97**
	SA	85.41±±2.76	**89.73±±4.49**	**85.95±±2.65**	86.49±±3.20	80.00±±6.23	84.41±±4.22	85.14±±5.57
Wisconsin	OA	87.45±±3.53	**86.47±±4.16**	**87.26±±3.87**	86.28±±3.62	85.88±±3.26	85.13±±4.13	85.42±±4.51
	SA	**89.41±±3.30**	**86.47±±4.84**	85.69±±5.04	**86.29±±4.32**	**86.67±±3.01**	**87.65±±3.17**	**88.04±±1.63**
Squirrel	OA	47.03±±1.31	37.85±±1.11	38.07±±1.71	38.43±±1.37	41.56±±1.74	49.88±±1.44	49.21±±1.95
	SA	**59.22±±1.44**	**46.16±±1.44**	**45.10±±2.11**	**46.25±±1.72**	**45.83±±1.40**	**50.83±±2.27**	**50.57±±1.52**
Chameleon	OA	67.79±±1.91	59.80±±1.54	58.93±±1.92	54.45±±2.29	56.05±±1.28	62.41±±1.99	62.63±±1.64
	SA	**71.38±±1.31**	**65.20±±1.65**	**65.09±±1.08**	**62.70±±2.30**	**62.30±±1.99**	**66.05±±1.21**	**65.83±±1.10**
Cora	OA	87.34±±1.34	86.72±±1.17	86.88±±1.09	86.90±±1.02	87.77±±1.35	88.01±±1.34	87.67±±1.14
	SA	**88.57±±0.66**	**88.39±±0.91**	**88.29±±0.67**	**88.31±±1.10**	**88.47±±1.19**	**88.03±±0.78**	**88.01±±0.80**
Citeseer	OA	77.33±±1.74	76.29±±1.74	76.73±±1.52	76.69±±1.97	76.57±±1.29	**77.28±±1.73**	77.38±±1.79
	SA	**77.60±±1.81**	**77.40±±1.54**	**77.38±±1.53**	**77.39±±1.73**	**77.25±±1.47**	77.09±±1.73	**77.42±±1.93**
Pubmed	OA	89.98±±0.38	87.99±±0.41	88.31±±0.44	88.83±±0.37	87.87±±0.33	89.96±±0.33	89.58±±0.39
	SA	**90.23±±0.55**	**90.09±±0.31**	**90.10±±0.36**	**90.11±±0.27**	**90.13±±0.36**	**90.07±±0.45**	**90.08±±0.46**

Table. 2-5: Ablation study on β

Dataset	β	GREAD-BS	GREAD-F	GREAD-AC	GREAD-Z	GREAD-ST	GREAD-FB	GREAD-FB*
Cornell	SC	85.14±±5.57	85.41±±6.75	85.41±±6.96	84.60±±6.17	85.95±±6.60	85.65±±6.21	84.16±±6.02
	VC	**86.49±±7.15**	**86.49±±5.13**	**87.03±±4.95**	**85.68±±5.41**	**86.22±±5.98**	**86.22±±5.85**	**85.95±±5.64**
Film	SC	37.09±±1.15	36.53±±1.04	**37.21±±1.10**	**37.01±±1.11**	**37.66±±0.90**	35.07±±0.92	34.24±±1.21
	VC	**37.90±±1.17**	**37.20±±1.26**	36.76±±0.99	36.70±±0.69	37.33±±1.35	**37.40±±0.55**	**37.70±±0.51**
Squirrel	SC	42.74±±1.34	44.88±±1.62	39.61±±1.69	40.33±±2.06	43.41±±1.61	40.59±±1.14	40.15±±1.66
	VC	**59.22±±1.44**	**46.16±±1.44**	**45.10±±2.11**	**46.25±±1.72**	**45.83±±1.40**	**50.83±±2.27**	**50.57±±1.52**
Chameleon	SC	62.02±±1.86	61.80±±1.80	56.56±±2.28	59.17±±1.26	60.70±±1.40	57.57±±1.83	57.70±±2.11
	VC	**71.38±±1.31**	**65.20±±1.65**	**65.09±±1.08**	**62.70±±2.30**	**62.30±±1.99**	**66.05±±1.21**	**65.83±±1.10**
Cora	SC	87.45±±1.08	88.07±±0.96	88.01±±0.85	88.13±±0.40	88.35±±1.32	87.75±±1.24	86.68±±0.88
	VC	**88.57±±0.66**	**88.39±±0.91**	**88.29±±0.67**	**88.31±±1.10**	**88.47±±1.19**	**88.03±±0.78**	**88.01±±0.80**
Citeseer	SC	76.73±±1.73	76.70±±1.75	75.83±±1.36	76.83±±1.16	**77.25±±1.47**	**77.28±±1.73**	**77.42±±1.93**
	VC	**77.60±±1.81**	**77.40±±1.54**	**77.38±±1.53**	**77.39±±1.73**	77.13±±2.20	77.22±±2.13	77.23±±1.89
Pubmed	SC	89.96±±0.42	87.51±±0.44	88.76±±0.45	90.04±±0.26	**90.13±±0.36**	89.90±±0.47	89.99±±0.24
	VC	**90.23±±0.55**	**90.09±±0.31**	**90.10±±0.36**	**90.11±±0.27**	90.10±±0.41	**90.07±±0.45**	**90.08±±0.46**



(a) Texas (b) Wisconsin (c) Cornell

(d) Film (e) Squirrel (f) Chameleon

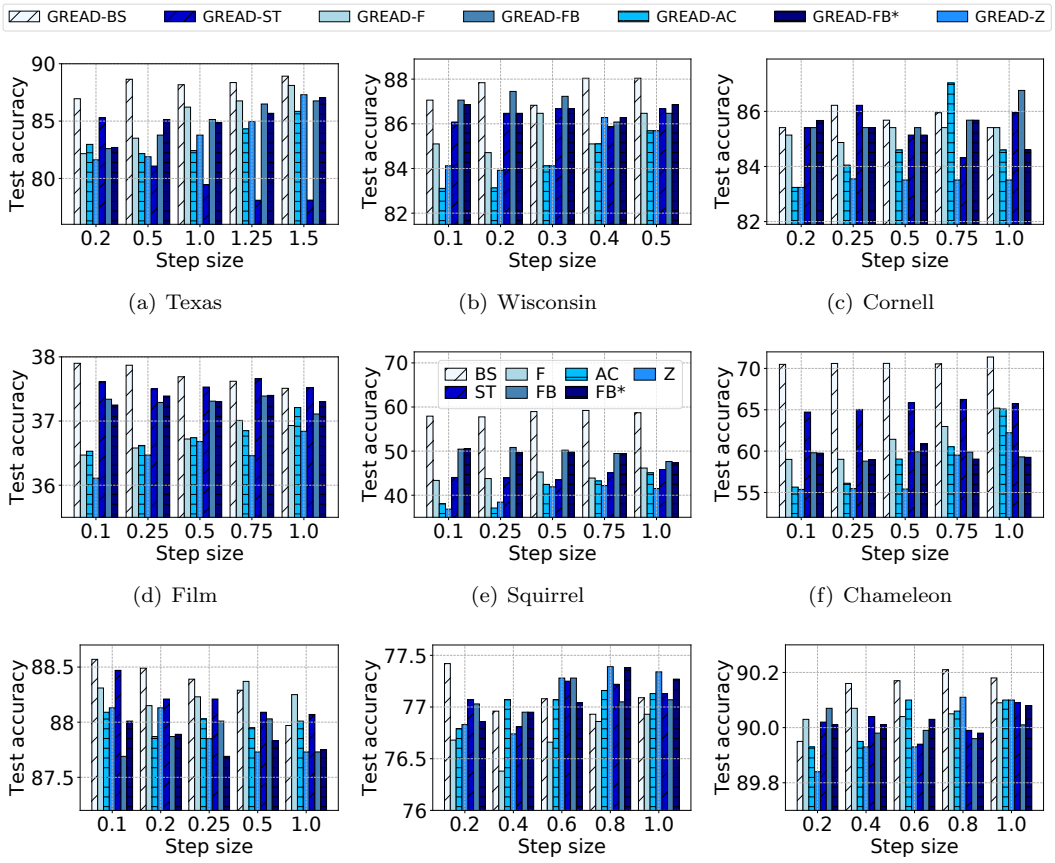
(g) Cora (h) Citeseer (i) Pubmed

Figure. 2-3: Sensitivity to T

where $\mathbf{H}_i, \mathbf{H}_j$ mean i-th and j-th rows, respectively.

The over-smoothing phenomenon occurs when the depth increases, causing node features to converge to constants. Thus, $E(\mathbf{H}, \mathbf{A})$ decays to zero asymptotically in time. We will show, via the evolution of the Dirichlet energy, that our proposed method mitigates the over-smoothing problem.

Experimental Settings We use the synthetic dataset, called cSBMs [83], to demonstrate the mitigation of over-smoothing. This synthetic data is an undirected graph representing 100 nodes in a two-dimensional space with two classes randomly connected with a probability of $p = 0.9$. In the case of GREAD, we run without any hyperparameter search but list the full hyperparameter list we used in Table 2-7 of Section 2.7.1. We report the layer-wise Dirichlet energy given a GNN of 40 layers.



(a) Texas (b) Wisconsin (c) Cornell

(d) Film (e) Squirrel (f) Chameleon

(g) Cora (h) Citeseer (i) Pubmed

Figure. 2-4: Sensitivity to step size

Experimental Results Figure 2-5 demonstrates that traditional GNNs, such as GCN and GAT, suffer from over-smoothing because the Dirichlet energy decays exponentially to zero in the first five layers. Converging to zero indicates that the node features become constant, while GREAD has no such behavior. The Dirichlet energy of GREAD can be bounded in time thanks to the reaction term. GRAND only has a diffusion term with learned diffusivity, so that it can delay over-smoothing. In the case of H2GCN, it is impossible to report on deeper layers due to memory limitations.

Ablation Studies on β We perform the ablation study on β from the perspective of the Dirichlet energy. β can be either a scalar parameter (SC) or a learnable vector parameter (VC). In Figure 2-6, we show the evolution of the Dirichlet energy on the synthetic random graph created from cSBM [83], and compare SC and VC for our proposed method. In the

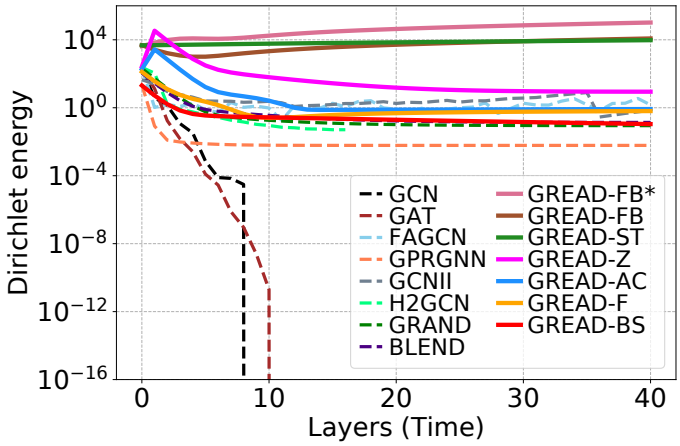


Figure. 2-5: Evolution of the Dirichlet energy on the synthetic random graph. The Y-axis is the logarithmic Dirichlet energy in each layer's output, given a GNN of 40 layers.

case of GREAD-F, GREAD-AC, GREAD-ST, and GREAD-FB, VC conserves more energy than SC, so the reaction term multiplied by β successfully mitigates the over-smoothing problem.

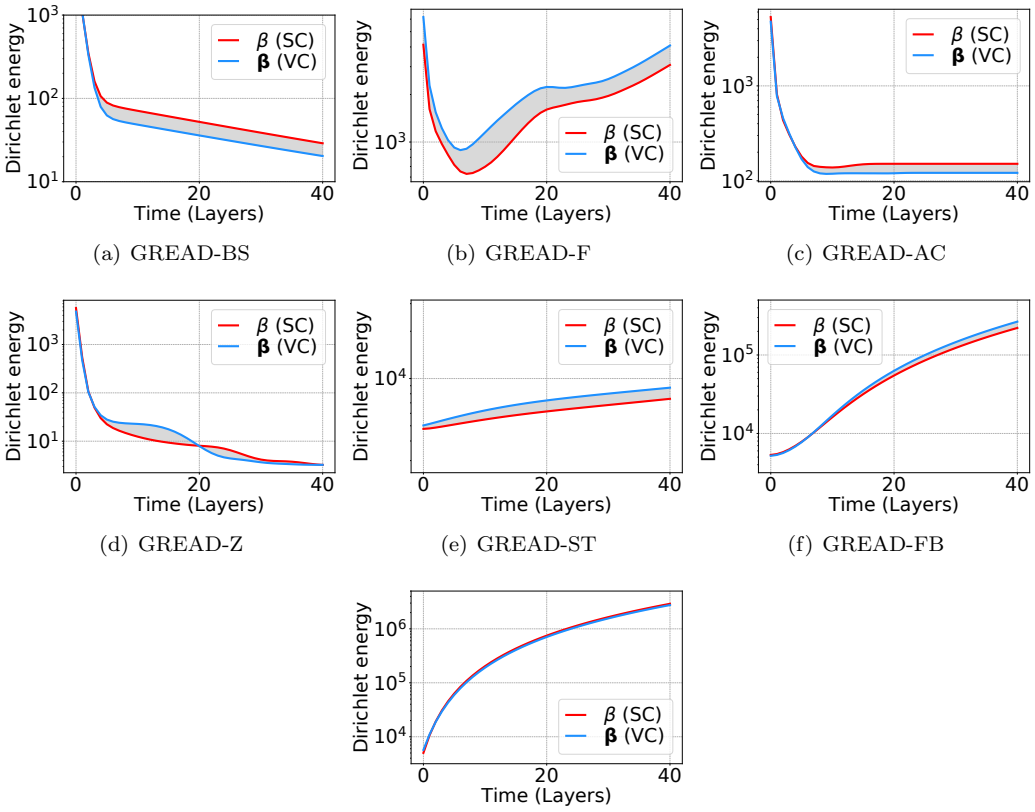
2.5.3 Different Homophily Levels

Experimental Settings To test the classification capability of GNNs, we use the synthetic Cora generator [28, 84]. We run the experiment with 3 fixed train/valid/test splits and report the mean and the standard deviation of accuracy accordingly. In Table 2-8 of Section 2.7.1, we list the hyperparameter range we consider.

Experimental Results Figure 2-7 shows the mean test accuracy on all random splits of the synthetic Cora datasets. MLP, which does not consider the connectivity of nodes, maintains its test accuracy for all homophily rates, which is obvious. GCN, GAT, and GRAND, which consider only diffusion, perform poorly at low homophily settings. H2GCN shows reasonable performance on low homophily rates, but its accuracy suddenly decreases at some homophily settings. All GREAD models have the best trend overall without sudden drops. The reaction terms of GREAD contribute to their stable accuracy for both homophily and heterophily settings compared with other models that rely on only diffusion processes, such as GCN and GRAND.

2.5.4 Training Time

We present the training time of GREAD and some selected baselines in Figure 2-8. In general, our method's training time is slightly larger than that of the existing baselines because



(a) GREAD-BS (b) GREAD-F (c) GREAD-AC

(d) GREAD-Z (e) GREAD-ST (f) GREAD-FB

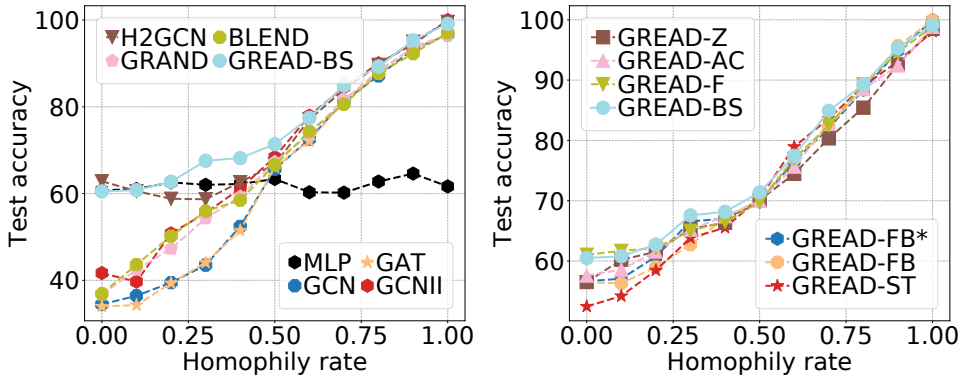
(g) GREAD-FB*

Figure. 2-6: Evolution of the Dirichlet energy on the synthetic random graph. The Y-axis is the logarithmic Dirichlet energy in each layer's output, given a GNN of 40 layers. The gray area is the Dirichlet energy difference between SC and VC.

GREAD includes an additional operation in its reaction term.

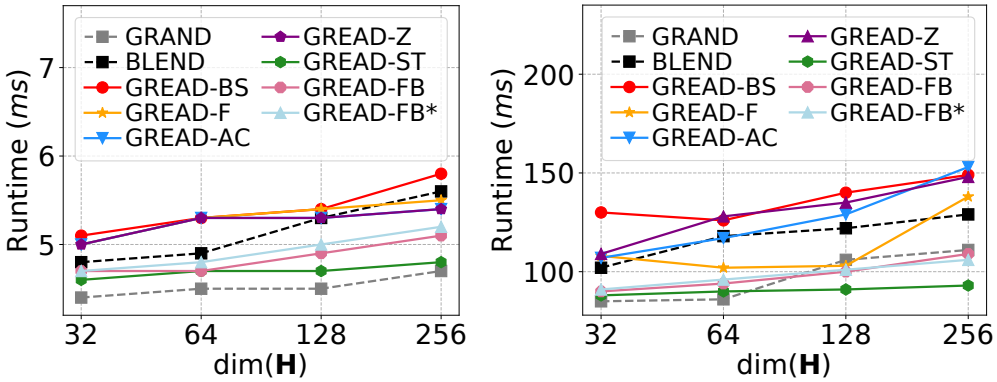
2.6 Summary

In this chapter, we presented the concept of a graph neural reaction-diffusion equation, called GREAD. Our proposed GREAD is one of the most generalized architectures considering diffusion and reaction processes. We design a reaction-diffusion layer that has three types of reaction equations widely used in natural sciences. We also added four reaction terms, including one special reaction term called Blurring-Sharpening (BS), which we designed for GNNs. Therefore, our reaction-diffusion layer has seven types. We consider a comprehensive set of 9 real-world datasets with various homophily difficulties and 28 baselines. GREAD



(a) GREAD-BS vs. baselines (b) GREAD

Figure. 2-7: Experiments on the synthetic Cora with controlled homophily rates.



(a) Euler (b) RK4

Figure. 2-8: Average running time per epoch (ms) on Cora dataset when $T = 3$, $\tau = 1.0$, SC, and OA.

marks the best accuracy in almost all cases. In our experiments with the two kinds of synthetic datasets, GREAD shows that it alleviates the over-smoothing problem and performs well on various homophily rates. This demonstrates that our proposed model is a novel framework for constructing GNNs based on the concept of the reaction-diffusion equation.

2.7 Appendix

2.7.1 Search Space of Hyperparameter

In this section, we report the range of hyperparameters considered in our experiments. Table 2-6 shows the range of hyperparameters searched for the real-world benchmark datasets. Table 2-7 shows the hyperparameters for an experiment to measure Dirichlet energy using cSBM synthetic networks, where we use fixed hyperparameters for fair comparison. In Table 2-7, continuous GNNs are considered with step size τ and time T, while discrete GNNs are considered with time T, where T corresponds to the number of GNN layers. Table 2-8 shows the hyperparameter search ranges used to compare performance in various homophily rates. Again, for discrete GNNs, the step size τ is irrelevant, and time T is the number of layers.

Table. 2-6: Hyperparameter search space for real-world datasets

Hyperparameters	Search Space
epochs	200
adjacency matrix	{OA, SA}
α	{SC, VC}
β	{SC, VC}
learning rate	$[1 \times 10^{-3}, 6 \times 10^{-2}]$
weight decay	$[0, 3 \times 10^{-2}]$
dropout	[0, 0.6]
input dropout	[0, 0.6]
dim($\mathbf{H}$)	{32, 64, 128, 256}
step size τ	[0.1, 1.5]
time T	[0.1, 6.0]
ODE solver	{Euler, RK4, DOPRI5}

2.7.2 Best Hyperparameters in Section 2.5.1

In this section, we report the best hyperparameters according to the reaction terms of our GREAD methods experimented with in Section 2.5.1. We provide the best hyperparameters for GREAD-BS in Table 2-9, the best hyperparameters for GREAD-F in Table 2-10, the best hyperparameters for GREAD-AC in Table 2-11, and the best settings for GREAD-Z in Table 2-12. Finally, GREAD-ST reports the best settings in Table 2-13, and GREAD-FB and FB* are provided in Tables 2-14 and 2-15.

Table. 2-7: Hyperparameter for the cSBM synthetic network

Hyperparameters	Value
epochs	100
adjacency matrix	OA
α	SC
β	VC
learning rate	0.001
weight decay	5×10^{-4}
dropout	0.0
input dropout	0.5
dim($\mathbf{H}$)	2
step size τ	1.0
time T	40
ODE solver	Euler

Table. 2-8: Hyperparameter search space for the synthetic Cora network

Hyperparameters	Search Space
epochs	100
adjacency matrix	{OA, SA}
α	{SC, VC}
β	{SC, VC}
learning rate	{0.001, 0.002, 0.0025, 0.005, 0.01}
weight decay	{0.01, 0.001, 0.0005, 0.0001}
dropout	0.35
input dropout	0.5
dim($\mathbf{H}$)	64
step size τ	{0.1, 0.5, 1.0}
time T	{1, 2, 3, 4}
ODE solver	Euler

Table. 2-9: Best hyperparameters of GREAD-BS

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	OA	SA	SA	SA	OA	SA	SA	SA	SA
α	SC	SC	VC	SC	VC	VC	VC	SC	VC
β	VC	VC	VC	VC	VC	VC	SC	SC	SC
learning rate	0.0100	0.0154	0.0082	0.0079	0.0171	0.0068	0.0105	0.0024	0.0108
weight decay	0.0247	0.0090	0.0280	0.0014	0.0000	0.0000	0.0060	0.0146	0.0005
input dropout	0.47	0.54	0.49	0.42	0.52	0.68	0.53	0.50	0.36
dropout	0.48	0.48	0.32	0.65	0.09	0.05	0.45	0.47	0.26
dim($\mathbf{H}$)	128	256	128	64	256	256	64	128	64
step size τ	1.0	0.25	0.2	0.1	0.75	1.5	0.25	0.5	0.8
time T	1.46	0.75	0.12	0.31	5.70	1.71	3.49	2.35	1.74
ODE solver	Euler	RK4	RK4	RK4	Euler	Euler	RK4	RK4	RK4

Table. 2-10: Best hyperparameters of GREAD-F

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	SA	SA	SA	SA	SA	SA	SA	SA	SA
α	VC	SC	VC	SC	VC	SC	SC	SC	VC
β	VC	VC	VC	VC	VC	VC	SC	VC	VC
learning rate	0.0113	0.0094	0.0092	0.0068	0.0054	0.0101	0.0048	0.0013	0.0120
weight decay	0.0079	0.0057	0.0263	0.0006	0.0011	0.0015	0.0370	0.0041	0.0003
input dropout	0.46	0.41	0.46	0.48	0.48	0.50	0.50	0.50	0.36
dropout	0.38	0.05	0.31	0.48	0.36	0.24	0.35	0.51	0.25
dim($\mathbf{H}$)	256	64	256	128	128	256	32	256	128
step size τ	1.0	0.1	1.0	0.75	1.0	1.0	0.2	0.9	1
time T	1.26	0.12	1.0	1.14	2.23	1.0	2.27	1.86	1.44
ODE solver	Euler	RK4	Euler	RK4	Euler	RK4	Euler	RK4	RK4

Table. 2-11: Best hyperparameters of GREAD-AC

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	SA	SA	SA	SA	SA	SA	SA	SA	SA
α	VC	VC	SC	SC	SC	SC	SC	SC	VC
β	VC	VC	VC	SC	VC	VC	VC	VC	VC
learning rate	0.0070	0.0083	0.0084	0.0027	0.0025	0.0038	0.0039	0.0029	0.0124
weight decay	0.0136	0.0081	0.0311	0.0001	0.0020	0.0007	0.0469	0.0140	0.0006
input dropout	0.40	0.45	0.49	0.46	0.52	0.52	0.40	0.47	0.30
dropout	0.30	0.20	0.29	0.48	0.28	0.35	0.40	0.49	0.26
dim($\mathbf{H}$)	256	128	128	128	128	256	128	64	128
step size τ	1.0	0.5	0.75	1.0	1.0	1.0	0.1	0.9	1.0
time T	1.36	0.20	0.18	1.06	1.98	2.0	3.52	2.78	1.65
ODE solver	Euler	RK4	RK4	Euler	Euler	RK4	Euler	RK4	RK4

Table. 2-12: Best hyperparameters of GREAD-Z

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	OA	SA	SA	SA	SA	SA	SA	SA	SA
α	VC	SC	SC	SC	VC	VC	VC	VC	VC
β	SC	VC	SC	VC	VC	VC	SC	VC	VC
learning rate	0.0088	0.0046	0.0048	0.0023	0.0099	0.0111	0.0045	0.0027	0.0091
weight decay	0.0462	0.0086	0.0435	0.0011	0.0007	0.0012	0.0050	0.0145	0.0004
input dropout	0.48	0.45	0.4272	0.48	0.53	0.45	0.4	0.50	0.37
dropout	0.46	0.18	0.29	0.48	0.44	0.31	0.2	0.49	0.22
dim($\mathbf{H}$)	256	128	256	64	128	256	64	64	64
step size τ	1.2	0.4	0.2	0.2	1.0	1.0	0.1	0.8	0.8
time T	1.2	0.11	0.13	0.75	2.71	1.0	3.55	2.01	1.12
ODE solver	RK4	RK4	RK4	RK4	RK4	RK4	RK4	RK4	RK4

Table. 2-13: Best hyperparameters of GREAD-ST

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	OA	SA	SA	SA	SA	SA	SA	SA	SA
α	SC	SC	SC	SC	VC	VC	SC	SC	SC
β	SC	VC	VC	SC	VC	VC	VC	SC	SC
learning rate	0.0200	0.0180	0.0050	0.0081	0.0538	0.0077	0.0074	0.0038	0.0108
weight decay	0.0295	0.0082	0.0275	0.0013	0.0000	0.0000	0.0086	0.0042	0.0004
input dropout	0.46	0.54	0.47	0.42	0.61	0.65	0.37	0.49	0.36
dropout	0.50	0.50	0.25	0.56	0.95	0.09	0.41	0.54	0.22
dim($\mathbf{H}$)	126	256	256	64	256	256	128	64	64
step size τ	0.5	0.5	0.25	0.7	1.0	1.0	0.1	0.6	0.9
time T	1.02	0.1	0.20	0.15	3.54	1.0	3.04	2.37	1.28
ODE solver	Euler	RK4	RK4	RK4	Euler	Euler	RK4	RK4	RK4

Table. 2-14: Best hyperparameters of GREAD-FB

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	OA	SA	SA	SA	SA	SA	SA	OA	SA
α	VC	SC	SC	SC	VC	VC	SC	VC	VC
β	SC	VC	VC	VC	VC	VC	VC	SC	VC
learning rate	0.0016	0.0185	0.0050	0.0133	0.0090	0.0010	0.0064	0.0012	0.0102
weight decay	0.0055	0.0113	0.0283	0.0014	0.0000	0.0000	0.0091	0.0042	0.0004
input dropout	0.52	0.50	0.36	0.51	0.62	0.64	0.47	0.45	0.35
dropout	0.48	0.53	0.23	0.60	0.06	0.05	0.50	0.54	0.21
dim($\mathbf{H}$)	64	64	256	64	128	128	256	128	64
step size τ	1.5	0.7	1.0	0.6	0.25	0.25	0.5	0.6	0.2
time T	1.4	1.0	0.1	1.3	2.4	1.8	3.1	1.5	1.0
ODE solver	Euler	RK4	RK4	RK4	Euler	Euler	RK4	RK4	Euler

Table. 2-15: Best hyperparameters of GREAD-FB*

Hyperparameters	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
adjacency matrix	OA	SA	SA	SA	SA	SA	SA	SA	SA
α	SC	SC	SC	SC	VC	VC	VC	VC	SC
β	VC	VC	SC	VC	VC	VC	VC	SC	VC
learning rate	0.0194	0.0195	0.0072	0.0144	0.0055	0.0095	0.0097	0.0020	0.0166
weight decay	0.0113	0.0142	0.0169	0.0010	0.0000	0.0000	0.0090	0.0048	0.0005
input dropout	0.45	0.51	0.36	0.52	0.63	0.65	0.50	0.57	0.35
dropout	0.52	0.48	0.19	0.59	0.05	0.14	0.39	0.39	0.22
dim($\mathbf{H}$)	64	64	128	64	128	128	64	64	128
step size τ	1.5	1.0	0.1	0.8	0.1	0.2	0.1	0.9	1.0
time T	1.4	1.0	0.2	1.9	2.0	1.5	3.3	1.7	1.4
ODE solver	Euler	Euler	Euler	RK4	Euler	Euler	Euler	RK4	RK4

Chapter 3

Over-smoothing II

Transformers, renowned for their self-attention mechanism, have achieved state-of-the-art performance across various tasks in natural language processing, computer vision, time-series modeling, etc. However, one of the challenges with deep Transformer models is the oversmoothing problem, where representations across layers converge to indistinguishable values, leading to significant performance degradation. We interpret the original self-attention as a simple graph filter and redesign it from a graph signal processing (GSP) perspective. We propose a graph-filter-based self-attention (GFSA)[1] to learn a general yet effective one, whose complexity, however, is slightly larger than that of the original self-attention mechanism. We demonstrate that GFSA improves the performance of Transformers in various fields, including computer vision, natural language processing, graph-level tasks, speech recognition, and code classification.

The materials in this chapter have been published in Choi et al. [85].

3.1 Motivation

Transformers are arguably one of the most significant achievements in the field of deep learning. They are now showing state-of-the-art performance in various fields, ranging from computer vision to natural language processing, prediction tasks on graphs, speech recognition, and so forth [64, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]. Recently, there have been several studies conducted on better understanding them [101, 102, 103]; there exists a common agreement among researchers that the self-attention is one of the keys leading to the success.

However, there also exist several studies pointing out potential limitations of the self-attention [91, 104, 101, 1]. For instance, Shi et al. [105] revealed an analogy between the self-attention and the residual GCN, showing that BERT also suffers from a notorious

[1]The source code of GFSA is available at: `https://github.com/jeongwhanchoi/GFSA`.

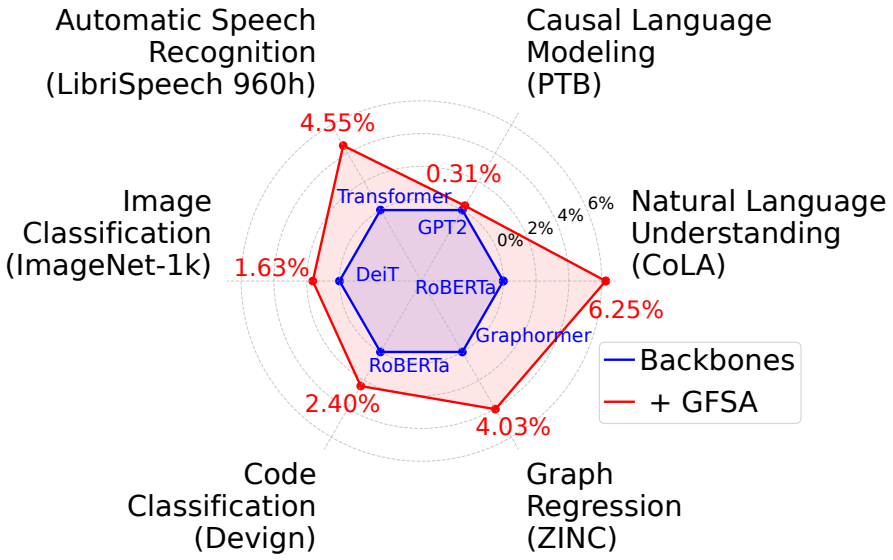
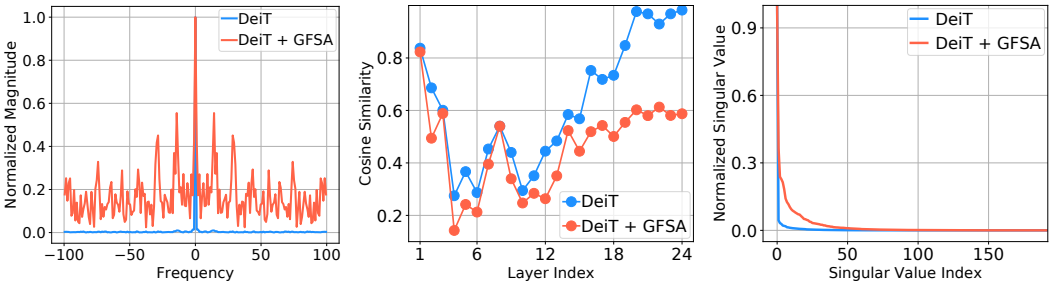


Figure. 3-1: Performance improvements (%) of our GFSA when integrated with different Transformer backbones in various domains. We achieve these results with only tens to hundreds of additional parameters to Transformers.

problem of GCNs, called *over-smoothing*, i.e., tokens' latent representations become similar to each other at the deeper layers. In every self-attention layer, value vectors are aggregated in a weighted average manner since each row-wise sum of the attention matrix is always 1. Although each self-attention layer has its own attention matrix, this aggregation method causes the over-smoothing problem, not only in Transformers but also in graph neural networks [8, 9, 106, 82, 107, 91, 101, 108, 109, 105, 103, 102, 110, 111]. However, we confine our discussion to the oversmoothing of Transformers (see Section 3.2).

Inspired by them, we redesign the self-attention from the perspective of graph signal processing (GSP) — in particular, we resort to GSP on directed graphs since the attention matrix is asymmetric. However, performing graph convolutions in the self-attention layer may incur non-trivial computational overheads. Therefore, our key design point is to learn a general but effective graph filter with minimal overhead. In general, a graph filter on a graph $\mathcal{G}$ is represented by a polynomial expression based on its adjacency or Laplacian matrix — in this regard, the existing self-attention mechanism can be understood as the simplest graph filter with $\bar{\boldsymbol{A}}$ only, where $\bar{\boldsymbol{A}} \in [0,1]^{n\times n}$ means a learned attention matrix and n is the number of input tokens.

Our proposed graph filter consists of an identity term and two matrix polynomial terms, $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{A}}^K$. One can design better filters with more polynomial terms, but we avoid it since Transformers already require very large computation. The K-th power, $\bar{\boldsymbol{A}}^K$, may also require a high computation when the number of tokens is large. To avoid this, we further



(a) Filter response (b) Cosine similarity (c) Singular value

Figure. 3-2: Filter frequency response, cosine similarity, and singular values on ImageNet-1k for DeiT-S and DeiT-S + GFSA. Details and more visualizations are in Sections 3.8.1 and 3.8.2.

approximate $\bar{\boldsymbol{A}}^K$ using the element-wise first-order Taylor approximation. Therefore, one can consider that our proposed graph filter is the next complicated filter after the one used by the original self-attention mechanism. However, its efficacy is tremendous in various fields (cf. Figure 3-1).

Our proposed filter enriches the self-attention with more diverse frequency information (see Figure 3-2(a)) — low (resp. high) frequency signals on $\mathcal{G}$ mean neighboring nodes have similar (resp. different) values. Therefore, our method cannot only effectively address the over-smoothing problem but also learn better latent representations for downstream tasks.

There exist a couple of prior works to enrich the self-attention mechanism with high-frequency information [103, 112]. In comparison with them, our proposed graph filter is distinctive in the following aspects: i) our proposed filter is more effective and shows better performance with comparable computational overheads, ii) our proposed filter is well-aligned with recent advancements in the GCN community — in other words, some graph filters used by recent advanced GCN methods are special cases of our proposed graph filter, which is not the case for prior works, and iii) other methods were typically studied for certain domains only whereas we test our method in 6 domains — for instance, DiversePatch [101] works only for Vision Transformers (ViTs).

We replace the self-attention layer of selected Transformers in various fields with our proposed graph filter-based layer without changing other parts. Therefore, the accuracy increases in them are solely by our proposed graph filter-based self-attention. These enriched Transformers increase the model performance by 1.63% for image classification, 6.25% for natural language understanding, 0.31% for causal language modeling, 4.03% for graph regression, 4.76% for speech recognition, and 2.40% for code classification (see Figure 3-1). Our core contributions are as follows:

- We provide a novel perspective on self-attention as a graph filter. This perspective

allows us to design more effective self-attention that can address the over-smoothing problem.

- We propose a graph filter-based self-attention (GFSA) mechanism, integrating an identity term and two polynomial terms for a more general yet effective self-attention mechanism than the simple self-attention mechanism (Section 3.3).
- We demonstrate that GFSA improves the performance of Transformers on a variety of tasks. GFSA achieves improved results on natural language processing, computer vision, speech recognition, graph-level tasks, and code classification (Sections 3.5.1 to 3.5.6).
- We devise a strategy to selectively apply GFSA to even-numbered layers, effectively mitigating the computational overhead while preserving GFSA's performance (Section 3.6).

3.2 Background & Related Work

3.2.1 Self-Attention in Transformers

The core building block of the Transformer architecture is the self-attention mechanism, which enables the model to learn attention patterns over its input tokens [64]. The self-attention mechanism, denoted as $\mathrm{SA} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$, can be expressed as follows:

$$\mathrm{SA}(\boldsymbol{X}) = \mathrm{softmax}\Big(\frac{\boldsymbol{X}\boldsymbol{W}_{\mathrm{qry}}(\boldsymbol{X}\boldsymbol{W}_{\mathrm{key}})^{\intercal}}{\sqrt{d}}\Big)\boldsymbol{X}\boldsymbol{W}_{\mathrm{val}} = \bar{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}_{\mathrm{val}}, \tag{3.1}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is the input feature matrix, $\boldsymbol{W}_{\mathrm{key}} \in \mathbb{R}^{d\times d}$, $\boldsymbol{W}_{\mathrm{qry}} \in \mathbb{R}^{d\times d}$, and $\boldsymbol{W}_{\mathrm{val}} \in \mathbb{R}^{d\times d}$ are the key, query, and value trainable parameters, respectively, and d is the dimension of each token. The self-attention mechanism allows the model to weigh the importance of each token in the input sequence relative to the others, enabling the model to capture long-range contextual information better. The Transformer architecture includes multiple layers, each with a multi-head self-attention layer followed by a position-wise feed-forward layer.

3.2.2 Self-Attention and Graph Convolutional Filter

The self-attention matrix used in Transformers has the form of symmetrically normalized adjacency matrix where each token become a node [105, 1] — the symmetrically normalized adjacency matrix is a special case of asymmetric (or directed) adjacency matrix where each row is normalized and is frequently used for the graph signal processing (GSP) on directed graphs [113]. A weighted graph $\mathcal{G}$ with adjacency matrix $\boldsymbol{A}$ can be constructed by using the input tokens as n nodes and the edge weights between node i and node j as $\exp((\boldsymbol{X}\boldsymbol{W}_{\mathrm{qry}})_i^{\intercal}(\boldsymbol{X}\boldsymbol{W}_{\mathrm{key}})_j)$. We can rewrite the self-attention matrix $\bar{\boldsymbol{A}}_{ij}$ as

$\frac{\exp\left((\boldsymbol{X}\boldsymbol{W}_{\text{qry}})_i^\intercal(\boldsymbol{X}\boldsymbol{W}_{\text{key}})_j\right)}{\sum_{k=1}^{d}\exp\left(\boldsymbol{X}\boldsymbol{W}_{\text{qry}})_i^\intercal(\boldsymbol{X}\boldsymbol{W}_{\text{key}})_k\right)}$. This allows $\bar{\boldsymbol{A}}$ to be interpreted as the symmetrically normalized adjacency matrix. In other words, $\bar{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$, where $\boldsymbol{D} = \text{diag}(d_1, d_2, \ldots, d_n)$ and $d_i = \sum_j \boldsymbol{A}_{i,j}$.

Our new attention method is designed on top of GSP, which has a close connection to discrete signal processing (DSP) [114, 115]. In DSP, a discrete signal with a length of n can be represented by a vector $\boldsymbol{x} \in \mathbb{R}^n$. Let $\boldsymbol{g} \in \mathbb{R}^n$ be a filter that we want to apply to $\boldsymbol{x}$. The convolution $\boldsymbol{x} * \boldsymbol{g}$ can be written as follows:

$$\boldsymbol{y}_i = \sum_{j=1}^{n} \boldsymbol{x}_j \boldsymbol{g}_{i-j}, \tag{3.2}$$

where the index, denoted as i, refers to the i-th element in each vector.

GSP can be understood as a generalized concept of DSP. Signals are defined on the nodes of a graph, and the graph's structure influences signal processing operations. In addition, the linear and shift-invariant graph convolution filter $\boldsymbol{H}$ with n nodes can be written with a shift operator $\boldsymbol{S}$ as follows — $\boldsymbol{S}$ can be from a directed graph [116]:

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} = \sum_{k=0}^{K} w_k \boldsymbol{S}^k \boldsymbol{x}, \tag{3.3}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is a 1-dimensional graph signal, K is the maximum order of polynomial, and $w_k \in [-\infty, \infty]$ is a coefficient. $\boldsymbol{S}$ is an $n \times n$ matrix where (i, j)-th element is non-zero if and only if there is an edge from node i to j. Two representative samples of $\boldsymbol{S}$ are the adjacency and Laplacian matrices. The graph filter $\boldsymbol{H}$ is the same as $\sum_{k=0}^{K} w_k \boldsymbol{S}^k$ with a large enough value of K, which is called *matrix polynomial* [116]. We note that this graph filtering operation can be extended to d-dimensional cases as in Equation (3.1). Being inspired by Zou et al. [117] and Maskey et al. [113], we rely on the singular value domain analysis to understand the low/high-pass characteristics of filters on directed graphs (cf. Figure 3-2). See more discussion in Sections 3.8.3 and 3.8.4.

In the context of the self-attention within Transformers, the core part of the self-attention in Equation (3.1), i.e., $\bar{\boldsymbol{A}}\boldsymbol{X}$, can be considered as a d-dimensional graph filter with $\bar{\boldsymbol{A}}$ only, where $\boldsymbol{H} = \bar{\boldsymbol{A}}$. Our goal in this paper is to design a simple (for computational purposes) yet effective form of $\boldsymbol{H}$.

3.2.3 Over-smoothing in GCNs and Transformers

Over-smoothing is a phenomenon observed in deep learning models, especially in GCNs [22, 23]. As information is aggregated over multiple layers for multiple nodes (tokens), latent representations tend to become similar to each other, leading to a loss of distinctiveness in the representations [8, 118, 82].

Surprisingly, an over-smoothing-like phenomenon is also observed in Transformers [103, 105]. Unlike CNNs, Transformers can not benefit from simply deepening layers after a certain

depth. Earlier studies hypothesize that this may be due to issues such as attention or feature collapse, or due to uniformity among patches or tokens [91, 101, 108]. Dong et al. [104] also point out that the output of a pure Transformer, i.e., an attention mechanism without skip connections or MLPs, tends to converge to a rank-1 matrix [104]. This analysis is followed by [109], which suggests that rank collapses incur vanishing gradients of attention queries and keys.

In this context, self-attention acts as a low-pass filter, as it calculates the weighted average of the value vectors of tokens. Wang et al. [103, Theorem 1] also reveal that the self-attention is a low-pass filter, continuously reducing high-frequency information. This nature contributes to the over-smoothing phenomenon as unique high-frequency features are lost in deeper layers of the network, further worsening the uniformity of token representations. Therefore, we extend the term "over-smoothing" to describe the degeneration challenge observed in Transformers.

Many empirical countermeasures have been proposed for ViT, such as patch diversification [119, 101], rank collapse alleviation [91, 120], and training stabilization [90, 121]. Similar alleviating methods have also been proposed in the field of NLP, such as unsupervised learning [122], and resolve the over-smoothing and the token uniformity (or information diffusion) problems [104, 108]. There are studies on utilizing high-frequency information via frequency domain analyses [103, 112], but they are not designed on top of graph filtering perspectives. Dovonon et al. [123] find that Transformers are not inherently low-pass filters, but over-smoothing depends on the eigenspectrum of the self-attention layers. They propose a reparameterization of the Transformer weights to ensure that over-smoothing does not occur.

Our paper addresses the over-smoothing problem with graph filters since the self-attention mechanism is a basic graph filtering operation, as seen in the previous subsection.

3.3 Graph Filter-based Self-Attention Layers

Let $\bar{\boldsymbol{A}} \in [0,1]^{n \times n}$, where n is the number of tokens in the input to the self-attention layer, be a self-attention matrix. Since Transformers use multi-head self-attention, there are multiple such matrices. For simplicity, but without loss of generality, we discuss only one head in one layer.

From the GSP perspective, using $\bar{\boldsymbol{A}}$ as the shift operator, a graph filter can be represented as a matrix polynomial filter, as mentioned in Section 3.2.2. We aim to design this matrix polynomial filter using the two lowest-order terms and one high-order term in Equation (3.3). The following theorem shows that, despite using the three terms, the filter can be either a low-pass filter or a high-pass filter, depending on the coefficient values.

Theorem 3.3.1 (Filter characteristics based on coefficient values)**.** *Let $\bar{\boldsymbol{A}}$ be a self-attention matrix interpreted as a graph with connected components. Consider the polynomial graph filter*

defined by $\sum_{k=0}^{K} w_k \bar{\boldsymbol{A}}^k$*, where* $w_2, w_3, \ldots, w_{K-1} = 0$ *and only* w_0*,* w_1*, and* w_K *are non-zero. If the coefficients* w_k *for* $k = 0, 1, K$ *are positive and their sum is 1, then the polynomial filter acts as a low-pass filter, attenuating high-frequency components and promoting smoothness across the graph. Conversely, if* $w_k = (-\alpha)^k$ *for* $k = 0, 1, K$ *and* $\alpha \in (0, 1)$ *with sufficient large* K*, the polynomial filter exhibits high-pass filter behavior.*

The proof of Theorem 3.3.1 is in Section 3.8.5. Based on Theorem 3.3.1, we propose to use the following graph filter, $\boldsymbol{H}_{\text{GFSA}}$, where the two lowest-order terms and one high-order term of the matrix polynomial are used:

$$\boldsymbol{H}_{\text{GFSA}} = w_0 \boldsymbol{I} + w_1 \bar{\boldsymbol{A}} + w_K \bar{\boldsymbol{A}}^K, \tag{3.4}$$

where w_0, w_1, w_K are coefficients and K is a hyper-parameter where $K \geq 2$. The coefficients can be learnable weights, which we learn with gradient descent algorithms.

Approximation of the high-order term. In Equation (3.4), it is costly to calculate $\bar{\boldsymbol{A}}^K$ when K is large, so we need a way to approximate the high-order term $\bar{\boldsymbol{A}}^K$ in GFSA. We use the first-order Taylor approximation at point $a = 1$ for this purpose:

$$f(x) \simeq f(a) + f'(a)(x - a), \tag{3.5}$$

thus, we approximate $f(K) = \bar{\boldsymbol{A}}^K$ as follows:

$$f(K) = \bar{\boldsymbol{A}}^K \simeq f(1) + f'(1)(K - 1). \tag{3.6}$$

Computing the derivative of $\bar{\boldsymbol{A}}^K$ directly at the evaluation point requires high computational costs. To overcome this problem, we adopt the forward finite difference method, which approximates derivatives with the difference term:

$$f'(K) = \frac{f(K + h) - f(K)}{h} = \frac{\boldsymbol{A}^{K+h} - \boldsymbol{A}^K}{h}, \tag{3.7}$$

where the approximation error is $\mathcal{O}(h^2)$. To balance the trade-off between computational efficiency[2] and accuracy, we set $h = 1$. This method is inspired by the approach in Brouwer et al. [125], which uses the difference term between two consecutive hidden states in discrete time to approximate the derivatives. Therefore, we approximate $\bar{\boldsymbol{A}}^K$ as:

$$\begin{aligned} f(K) = \bar{\boldsymbol{A}}^K &\simeq f(1) + (\bar{\boldsymbol{A}}^2 - \bar{\boldsymbol{A}})(K - 1) \\ &= \bar{\boldsymbol{A}} + (K - 1)(\bar{\boldsymbol{A}}^2 - \bar{\boldsymbol{A}}). \end{aligned} \tag{3.8}$$

The approximation for $\bar{\boldsymbol{A}}^K$ with $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{A}}^2$ provides a simpler computation that can significantly reduce the required computational resources and time.

[2] Calculating the power of a matrix for small h requires a high computational cost since it is calculated by diagonalizing the matrix or using Schur normal form [124].

GFSA: our graph filter-based self-attention. Our proposed graph filter-based self-attention (GFSA) is defined with the graph filter $\tilde{\boldsymbol{H}}_{\text{GFSA}}$ as follows:

$$\text{GFSA}(\boldsymbol{X}) := \tilde{\boldsymbol{H}}_{\text{GFSA}}\boldsymbol{X}\boldsymbol{W}_{\text{val}}, \tag{3.9}$$

$$\tilde{\boldsymbol{H}}_{\text{GFSA}} = w_0\boldsymbol{I} + w_1\bar{\boldsymbol{A}} + w_K(\bar{\boldsymbol{A}} + (K-1)(\bar{\boldsymbol{A}}^2 - \bar{\boldsymbol{A}})), \tag{3.10}$$

where the last term is the approximated $\bar{\boldsymbol{A}}^K$ from Equation (3.8). We replace the original self-attention layer in various Transformers with the proposed graph filter-based layer without changing other parts. Therefore, GFSA can be plugged into any Transformers that rely on self-attention. For pseudocode, see Section 3.8.7.

3.4 Properties of GFSA

This section analyzes the theoretical error of the $\bar{\boldsymbol{A}}^K$ approximation used by GFSA and how GFSA can mitigate over-smoothing. We also explain the meaning of GFSA's high-order term in the context of Transformers and provide comparisons of GFSA in other models.

Theoretical characteristics of approximation error in GFSA. We provide a theorem that provides an upper bound on the error introduced by approximating the power of a matrix, specifically using the first-order Taylor expansion. The following theorem specifically analyzes the error of matrix $\bar{\boldsymbol{A}}^K$ when approximated using a first-order Taylor expansion.

Theorem 3.4.1 (Error bound for approximated high-order term in GFSA)**.** *Define the error term, E_K, as the difference between the exact value and approximated value of $\bar{\boldsymbol{A}}^K$, which is given by $E_K = ||\bar{\boldsymbol{A}}^K - (\bar{\boldsymbol{A}} + (K-1)(\bar{\boldsymbol{A}}^2 - \bar{\boldsymbol{A}}))||_F$, where $||\cdot||_F$ denotes the Frobenius norm. Then, the error bound can be shown that $E_K \leq 2\sqrt{n}K$.*

The error bound provides an upper limit for the difference between the actual value of $\bar{\boldsymbol{A}}^K$ and its approximation. The proof of Theorem 3.4.1 is in Section 3.8.6. It shows theoretical validity for using approximations to the high-order term in the filters of our GFSA. In terms of performance, we report a difference of approximately $\bar{A}^K$ between the actual calculated values in Section 3.8.8.

How to alleviate the over-smoothing problem? The key leading to the low/high pass filtering behavior of our proposed filter is the coefficients $\{w_0, w_1, w_K\}$ — note that in the self-attention of Transformers, $w_0 = w_K = 0$ and $w_1 = 1$. Since our method can learn any appropriate values for them for a downstream task, it can be reduced to low-pass-only, high-pass-only, or combined filters. According to Theorem 3.3.1, our graph polynomial filter can be said to be a low-pass filter when w_1, w_K are positive and a high-pass filter when they

are negative. Therefore, our method can learn the appropriate coefficients $\{w_0, w_1, w_K\}$ for downstream tasks, so it can be reduced to a low-pass-only, high-pass-only, or combined filter, alleviating the over-smoothing problem of self-attention.

The meaning of the high-order term in GFSA in the context of Transformers. Existing self-attention only captures simple pairwise similarities between tokens and is limited in capturing high-order dependencies. For example, given the two sentences, "Books are more expensive than pencils" and "Books are cheaper than computers", to understand the relationship between "computers" and "pencils", we need to capture the high-order dependencies connected through the "Book" token. However, it is difficult to capture these high-order dependencies with traditional self-attention [126]. Therefore, from a Transformer perspective, the approximated $\bar{\boldsymbol{A}}^K$ in GFSA can be interpreted as being able to capture these high-order dependencies.

Comparison to Transformers. In the field of computer vision, there has been recent research on adjusting the frequency response of ViT. HAT [112] creates adversarial examples by altering clean images with high-frequency perturbations and jointly trains the ViT on clean images and adversarial examples. Through this, they aim to solve the problem of the ViT being unable to capture high-frequency by allowing us to capture the high-frequency components of the images. However, HAT has the disadvantage of requiring more epochs than the existing ViT, as it must perform adversarial training in some initial epochs and train normally in the remaining epochs. Wang et al. [103] use the concept of DSP, a special case of GSP, to isolate the lowest frequency component in the Fourier domain and use a filter learned by rescaling the low and high-frequency components. On the other hand, our GFSA extends the concept to graph signal processing and redesigns self-attention as a graph filter. While GFSA seeks to design a better graph filter by interpreting self-attention as a graph filter, Shi et al. [105] are inspired by JKNet [50], and they solve the over-smoothing problem by fusing the hidden vectors of each layer. However, their method has a limitation with memory increasing, and they only applied it to BERT.

Comparison to GCNs. Comparisons to GCNs that can be interpreted as graph filters [22, 24, 127] are inevitable. GFSA without a high-order term is analogous to ChebNet [24] with $K = 1$. In addition, GFSA reduces to the vanilla GCN [22] when $K = 1$, $w_0 = 0$, $w_1 = 1$. GPR-GNN [27], which approximates graph convolutions using the monomial basis, is identical to GFSA if it only considers up to first order and additionally uses a K-order term and learns the coefficients. When we use only a high-order term and w_K is learned to a negative value, GFSA can become similar to the reaction-diffusion layer of GREAD [15], $\bar{\boldsymbol{A}}\boldsymbol{X} + \beta(\bar{\boldsymbol{A}} - \bar{\boldsymbol{A}}^2)$, depending on the higher order terms.

3.5 Experiments

In this section, we demonstrate the effectiveness of GFSA through a series of experiments. These experiments encompass various tasks: i) language understanding and causal language modeling, ii) image classification, iii) graph-level task, and iv) code classification. We replace the self-attention of base Transformers in those fields with our GFSA. Our modification adds only tens to hundreds of parameters, which are negligible compared to the original size of the base models.

3.5.1 Experiments on Natural Language Understanding

Setting. We integrate GFSA into 3 pre-trained large language models: BERT, ALBERT, and RoBERTa. We evaluate them on the GLUE benchmark, which includes 3 categories of natural language understanding tasks: i) single-sentence, ii) similarity and paraphrasing, and iii) natural language inference tasks. For each task, we select the best hyperparameters for GFSA, and the other hyperparameters are fixed. The detailed experimental settings are in Section 3.8.9.

Results. The results are shown in Table 3-1. When GFSA was plugged into backbones, average performance scores improved across all models over pure backbones. This indicates that GFSA is effective in both large models like BERT and RoBERTa, as well as relatively smaller models like ALBERT. It is worth mentioning that in the case of RoBERTa finetuned on the CoLA dataset, there is a significant margin increase from 60.34% to 64.11%, which is a 3.77% improvement with only 144 additional parameters. When compared to ContraNorm, GFSA shows a greater performance improvement on average. Figure 3-5 in Section 3.8.1 shows that these performance enhancements can be attributed to addressing the over-smoothing issue through the designed graph filter.

Table. 3-1: Results comparison on GLUE benchmark. **Avg** denotes the average performance.

Method	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	110M	56.79	93.81	88.70	88.32	88.16	84.96/84.15	91.63	66.06	82.51
+ ContraNorm	110M	**59.89**	93.92	89.88	**88.51**	**88.36**	85.11/84.50	91.84	**69.31**	83.48
+ GFSA	110M	59.56	**94.15**	**90.60**	88.46	88.33	**85.12**/**85.06**	**91.95**	68.95	**83.58**
$\text{ALBERT}_{\text{BASE}}$ [128]	11M	57.86	92.32	91.80	85.30	90.37	**85.37**/84.37	91.76	76.90	84.01
+ ContraNorm	11M	57.45	93.00	**92.83**	87.78	90.55	85.06/84.57	**92.28**	**78.70**	84.69
+ GFSA	11M	**60.21**	**93.23**	92.47	**87.79**	**90.63**	85.29/**84.92**	92.17	**78.70**	**85.05**
$\text{RoBERTa}_{\text{BASE}}$ [129]	125M	60.34	94.84	92.28	88.86	89.99	87.94/87.30	92.57	78.70	85.87
+ ContraNorm	125M	63.06	**95.41**	93.17	88.91	90.34	87.88/87.40	92.82	**80.51**	86.61
+ GFSA	125M	**64.11**	**95.41**	**93.52**	**89.09**	**90.35**	**87.99**/**87.54**	**92.97**	80.14	**86.79**

3.5.2 Experiments on Causal Language Modeling

Setting. We also validate the effectiveness of GFSA on causal language modeling problems. We finetune GPT2 [88] on the following 3 datasets: Penn Treebank (PTB) [130], WikiText-2, and WikiText-103 [131]. Following the evaluation method in Yao et al. [132], we finetune models for 15 epochs with PTB, 4 epochs with WikiText-103, and 10 epochs with WikiText-2, and report the perplexity for the sensitivity metric. The detailed experimental settings are in Section 3.8.11.1.

Table. 3-2: Results comparison on GPT-2 finetuned with GFSA. **Avg** denotes the average performance.

Method	#Params	PTB	WikiText-2	WikiText-103	**Avg**
GPT2	117M	19.513	20.966	15.939	18.806
+ GFSA	117M	**19.450**	**20.923**	**15.919**	**18.764**

Results. Table 3-2 shows the perplexity on PTB, WitiText-2, and WikiText-103. Across all datasets, GPT2 with GFSA consistently outperforms vanilla GPT2. Our GFSA improves the average perplexity from 18.806 to 18.764. Note that performance improvements are made with only 144 additional learnable parameters for 12 layers with 12 heads.

3.5.3 Experiments on Vision Transformers

Setting. We aim to demonstrate the efficacy of our GFSA across a spectrum of ViT backbones. We choose DeiT [90], CaiT [133], and Swin [92] as the backbone, and the models are trained from scratch. When training the 12-layer DeiT, we follow the same training recipe, hyperparameters, and data augmentation from Touvron et al. [90]. For detailed experimental settings, see Section 3.8.12.1.

Results. In Table 3-3, we consider all three classes, CNN only, CNN + Transformer, and pure Transformer. We compare various models on the ImageNet-1k benchmark. The results show that the proposed GFSA successfully enhances DeiT, CaiT, and Swin across all depth settings and training methods. In particular, in the Transformer category, we only test with lightweight models with a similar number of parameters, such as ViT-S and DeiT-S. GFSA provides additional parameters less than 72 for 12-layer DeiT while improving top-1 accuracy by 1.63%. Compared to existing techniques, the improvements by GFSA already surpass LayerScale (0.7%) [133], LateInsertion (0.6%) [133], and HAT [112] (1.38%). To sum up, we observed that both shallow and deep ViTs can achieve the following benefits from GFSA: i) The filter response shows GFSA can preserve higher-frequency representation (cf. Figure 3-2 (a)) and ii) Figure 3-2 (b) shows that GFSA mitigates the increase in the cosine similarity of

Table. 3-3: Compared with state-of-the-art models on ImageNet-1k dataset. The number in (↑) indicates the performance improvement over the base model.

Category	Method	Input Size	#Layers	#Params	Top-1 Acc
CNN	ResNet-152 [134]	224	152	230M	78.1
	DenseNet-201 [135]	224	201	77M	77.6
CNN + Transformer	CVT-21 [136]	224	21	32M	82.5
	Refiner [119]	224	16	86M	81.2
Transformer	ViT-S/16 [89]	224	12	49M	78.1
	ViT-B/16 [89]	224	12	86M	79.8
	DeiT-S [90]	224	12	22M	79.8
	DeiT-S + LayerScale [133]	224	12	22M	80.5
	DeiT-S + LateInsertion [133]	224	12	22M	80.5
	DeiT-S + ClassAttention [133]	224	12	22M	80.6
	DeiT-S + AttnScale [103]	224	12	22M	80.7
	DeiT-S + FeatScale [103]	224	12	22M	80.9
	DeiT-S + HAT [112]	224	12	22M	80.9
	DeiT-S + Diverse [137]	224	12	22M	80.6
	DeiT-S + ContraNorm [1]	224	12	22M	80.4
	Swin-S [92]	224	12	50M	82.9
	T2T-ViT-24 [138]	224	24	64M	82.3
	DeepViT-24B [91]	224	24	36M	80.1
	DeiT-S [90]	224	24	43M	80.5
	CaiT-S [133]	224	24	47M	82.6
	DeiT-S + DiversePatch [101]	224	24	44M	82.2
	DeiT-S + LayerScale [133]	224	24	44M	82.4
	DeiT-S + AttnScale [103]	224	24	44M	81.1
	DeiT-S + FeatScale [103]	224	24	44M	81.3
	DeiT-S + ContraNorm [1]	224	24	43M	80.7
GFSA	DeiT-S + GFSA	224	12	22M	**81.1** (↑ 1.3)
	DeiT-S + GFSA	224	24	43M	**81.5** (↑ 1.0)
	CaiT-S + GFSA	224	24	47M	**82.8** (↑ 0.2)
	Swin-S + GFSA	224	12	50M	**83.0** (↑ 0.1)

representation as the layer gets deeper. We further compare with state-of-the-art models that use Fourier transforms rather than graph filters in Section 3.8.12.4. We also show results under the same settings as ContraNorm [1] in Section 3.8.12.5.

3.5.4 Experiments on Graph-level Tasks

Setting. To evaluate the efficacy of GFSA on graph-level tasks, we conduct experiments on a broader range of datasets. We use datasets from Long-Range Graph Benchmark (LRGB) [139]

(e.g., Peptide-func and Peptide-struct), Benchmarking GNNs [140] (e.g., ZINC, MNIST, CIFAR10), Open Graph Benchmark (OGB) dataset [141] (e.g., Molhiv and MolTox21), and OGB-LSC dataset (i.e., PCQM4M-LSC) [142]. We choose Graphormer [143], Graph-ViT [144], and GPS [95] as our backbone architectures, following their original experimental protocols for fair comparison. For GPS, we replace its self-attention module with our GFSA while maintaining its best configuration and other hyperparameters. For Graph-ViT, we apply GFSA to the Hadamard self-attention method, which He et al. [144] propose as optimal. For a detailed experimental setting, see Section 3.8.14.1.

Results. Tables 3-4 to 3-6 show consistent performance improvements when GFSA is integrated with backbone architectures. Graph-ViT + GFSA shows improvements on all datasets. On Peptide-func, it achieves a 0.65% increase in AP. Notably, in PCQM4M, incorporating GFSA improves the validation MAE by 7.20%.

Table. 3-4: Experimental results and number of parameters on ZINC

Method	#Params	MAE
Graphormer	500K	$0.1240_{\pm 0.006}$
Graphormer + GFSA	500K	$\mathbf{0.1189}_{\pm 0.002}$

Table. 3-5: Experimental results and number of parameters on PCQM4M and PCQM4Mv2

Method	#Params	PCQM4M		PCQM4Mv2	
		Train	Validate	Train	Validate
Graphormer	48.3M	$0.0535_{\pm 0.038}$	$0.1286_{\pm 0.016}$	$0.0250_{\pm 0.000}$	$0.0862_{\pm 0.000}$
Graphormer + GFSA	48.3M	$\mathbf{0.0312}_{\pm 0.001}$	$\mathbf{0.1193}_{\pm 0.000}$	$\mathbf{0.0249}_{\pm 0.000}$	$\mathbf{0.0860}_{\pm 0.000}$

3.5.5 Experiments on Automatic Speech Recognition

Setting. We conduct automatic speech recognition (ASR) experiments on the LibriSpeech [3] dataset [145], which consists of audio recordings paired with their transcriptions. We use Branchformer [97] and a pure Transformer. For implementation, we follow the recipes of SpeechBrain [146], and the detailed settings are in Section 3.8.13.1.

Results. Table 3-7 compares word error rates (WERs) on LibriSpeech 100h and 960h. For 100h, Transformer+GFSA achieves 10.30/25.30 on the test clean/other set, which is a 6.53% improvement over the Transformer for the WER of the test clean. For 960h, Transformer+GFSA shows a WER result of 2.31 in test clean, a 4.55% improvement over

[3] `http://www.openslr.org/12`

Table. 3-6: Experimental evaluation of GFSA plugged into GPS and Graph-ViT. Results marked with † indicate settings where we conducted our own experiments due to unavailable Hadamard self-attention performance in [144]'s paper.

Method	Peptide-func	Peptide-struct	MNIST	CIFAR10	Molhiv	MolTOX21	ZINC
	AP (↑)	MAE (↓)	Accuracy (↑)	Accuracy (↑)	ROCAUC (↑)	ROCAUC (↑)	MAE (↓)
GPS	$0.6535_{\pm 0.0041}$	$0.2500_{\pm 0.0005}$	$0.9805_{\pm 0.0013}$	$0.7230_{\pm 0.0036}$	–	–	$0.070_{\pm 0.004}$
+ GFSA	$\mathbf{0.6593}_{\pm \mathbf{0.0094}}$	$\mathbf{0.2496}_{\pm \mathbf{0.0013}}$	$\mathbf{0.9814}_{\pm \mathbf{0.0014}}$	$\mathbf{0.7244}_{\pm \mathbf{0.0048}}$	–	–	$\mathbf{0.069}_{\pm \mathbf{0.002}}$
Graph-ViT	$0.6919_{\pm 0.0085}$	$^{\dagger}0.2474_{\pm 0.0016}$	$0.9820_{\pm 0.0005}$	$0.6967_{\pm 0.0040}$	$0.7792_{\pm 0.0149}$	$0.7851_{\pm 0.0077}$	$0.0849_{\pm 0.0047}$
+ GFSA	$\mathbf{0.6964}_{\pm \mathbf{0.0025}}$	$\mathbf{0.2461}_{\pm \mathbf{0.0024}}$	$\mathbf{0.9826}_{\pm \mathbf{0.0004}}$	$\mathbf{0.6987}_{\pm \mathbf{0.0028}}$	$\mathbf{0.7830}_{\pm \mathbf{0.0109}}$	$\mathbf{0.7895}_{\pm \mathbf{0.0069}}$	$\mathbf{0.0845}_{\pm \mathbf{0.0032}}$

Table. 3-7: Results for ASR training on LibriSpeech 100h and 960h with GFSA

Method	#Params	LibriSpeech 100h		LibriSpeech 960h	
		test-clean WER	test-other WER	test-clean WER	test-other WER
Transformer	71.5M	11.02	25.42	2.42	5.50
+ GFSA	71.5M	**10.30**	**24.30**	**2.31**	**5.49**
Branchformer	109.8M	9.63	22.43	2.13	5.00
+ GFSA	109.8M	**9.60**	**22.25**	**2.11**	**4.94**

Transformer, and Branchformer+GFSA achieves 2.31/5.49 with an LM on the test clean/other sets. Figure 3-8 in Section 3.8.13.2 depicts the learning curves of train loss and valid loss when using GFSA, showing the effectiveness of our proposed filter.

3.5.6 Experiments on Code Classification

Setting. We conduct a code defect detection task based on the Devign dataset provided by Zhou et al. [147]. We use RoBERTa [129], CodeBERT [148], PLBART [149], and CodeT5 [150] as our backbone models. The detailed settings are in Section 3.8.15.1.

Results. Table 3-8 shows the accuracy of all models; GFSA results are better than the base models. The biggest improvement is 2.40% for RoBERTa. In the case of CodeT5-base, using GFSA shows an accuracy of 64.75, an improvement of 1.95% from 63.51. CodeT5-small+GFSA has only about 100 additional parameters compared to CodeT5-small with 60M parameters, and even more impressively, it surpasses the accuracy of CodeT5-base. The biggest improvement is 2.40% for RoBERTa. In Section 3.8.15.2, we include case studies for this task. We also report the results of the code clone detection task in Section 3.8.15.3.

Table. 3-8: Results on code classification. The number in (↑) indicates the improvement rate.

Method	Accuracy
RoBERTa	62.88
+ GFSA	**64.39** (↑ 2.40%)
CodeBERT	64.31
+ GFSA	**64.49** (↑ 0.12%)
PLBART	62.63
+ GFSA	**62.96** (↑ 0.52%)
CodeT5-small	63.25
+ GFSA	**63.69** (↑ 0.70%)
CodeT5-base	63.51
+ GFSA	**64.75** (↑ 1.95%)

3.6 Discussion on Runtime Overheads

Limitation. The introduction of our GFSA layer results in a slight increase in training and inference time. We report the runtimes when plugging GFSA in Sections 3.8.15.4 and 3.8.16. For GLUE benchmark, integrating GFSA into BERT enhances the average performance from 82.51% to 83.58% (see Table 3-1) with more overhead of less than 36 seconds per epoch based on average training time (see Table 3-20). Considering the improvements, the increases in training time are negligible.

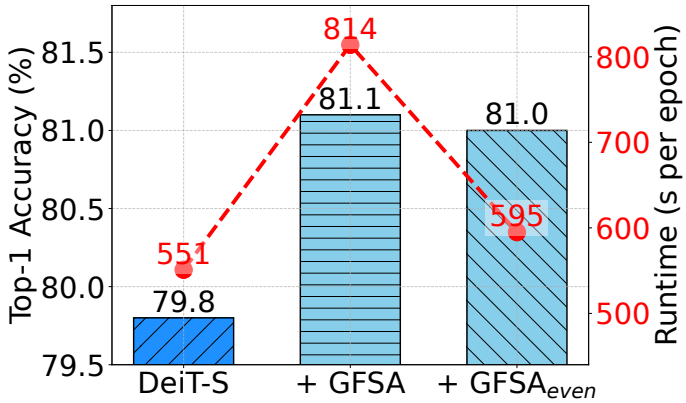


Figure. 3-3: Effectiveness of our selective layer strategy on ImageNet-1k. This shows out strategy's ability to maintain accuracy benefits while mitigating runtime increases.

GFSA in selected layers: a strategy to mitigate the limitation. As GFSA requires more calculation than the original self-attention, the runtime after using GFSA slightly increases. Our experiments initially applied GFSA across all Transformer layers (as discussed in Section 3.5); however, to reduce computational load, we propose a selective application strategy. For this purpose, GFSA is used only on even-numbered layers. In Tables 3-32 and 3-36 of Section 3.8.17, the results show that this strategy effectively reduces runtime increases while preserving comparable performance to the full-layer GFSA integration. Notably, the selective application of GFSA cuts the per-epoch runtime increase by 26.90% relative to its full-layer application, with only a 7.39% increase in runtime per epoch compared to the backbone model in Table 3-36.

GFSA in linear Transformers. Although GFSA requires additional computation for calculating $\bar{\boldsymbol{A}}^2$, we explore integrating GFSA with linear attention variants to maintain efficiency and scalability. Recent approaches [151, 152] achieve linear complexity by reformulating softmax operations and reordering matrix multiplication in self-attention. We apply similar principles to compute second-order self-attention efficiently, enabling $\tilde{\boldsymbol{H}}_{\text{GFSA}}$

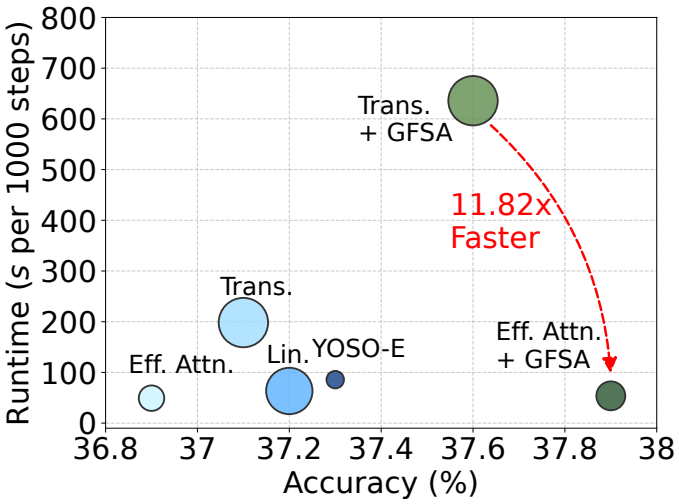


Figure. 3-4: Performance (x-axis), runtime (y-axis), and GPU usage (circle sizes) of various Transformers and integrated GFSA on Long-Range benchmark

calculation with linear complexity with respect to sequence length. Figure 3-4 shows the performance, runtime, and GPU usage changes when applying our GFSA to Transformers with linear complexity. GFSA still improves performance compared to the backbone model, while the increase in time and GPU usage is minimal. Notably, when GFSA is applied to Efficient Attention [152], the performance is improved and the runtime is 11.82 times faster than when GFSA is applied to the vanilla self-attention. This shows that GFSA can be effectively implemented with linear complexity architectures while preserving its benefits and providing a solution for addressing computational concerns.

3.7 Summary

Our proposed GFSA achieves high performance with improvements on a variety of tasks. GFSA is a simple yet effective method that enriches self-attention in Transformers with more diverse frequency information. This enables GFSA to address the over-smoothing problem and learn better latent representations for downstream tasks. However, our GFSA does not bring significant overheads in those Transformers' empirical runtime complexity. One can use more complicated graph filters to enhance accuracy further, but our goal is to find a balance between accuracy enhancements and overheads in runtime complexity.

We believe that GFSA suggests a promising new direction for improving Transformers. GFSA can be implemented only and used in conjunction with other techniques to further improve the performance of Transformers. Considering the ongoing advancements in large language models, such as GPT-4 [153] and LLaMA [154], we hope that our approach may

offer new insights for enhancing their performance and efficiency.

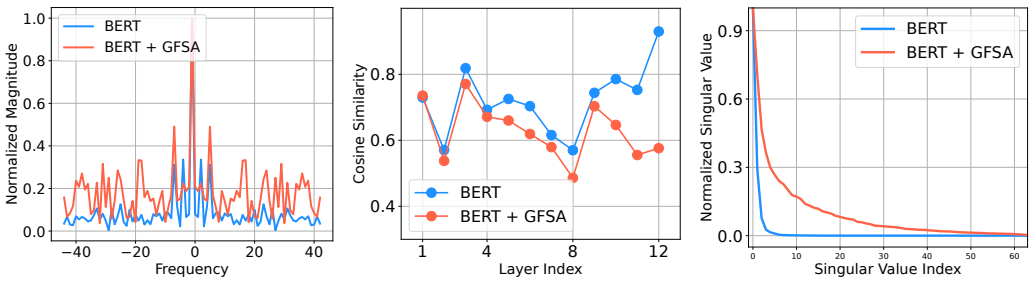
3.8 Appendix

3.8.1 Oversmoothing and Additional Visualizations

In Figure 3-2, we show the visualizations of over-smoothing characteristics in DeiT. We also provide visualizations in other domains. We show the filter response, cosine similarity, and singular value of BERT finetuned on STS-B dataset of GLUE tasks in Figure 3-5 and Graphormer finetuned on ZINC dataset in Figure 3-6.

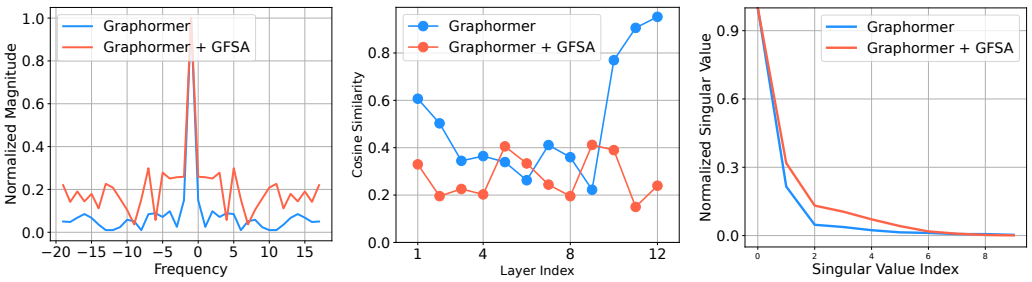
To characterize self-attention, we first analyze the filter response of self-attention in the frequency domain. We follow the method used by Wang et al. [103] for spectral visualization of the self-attention matrix. As shown in Figure 3-2 (a), DeiT has a near-zero magnitude for the high frequencies, which is characteristic of a low-frequency filter and is likely to result in oversmoothing when applied multiple times.

We follow the calculation method of Guo et al. [1] for cosine similarity. As shown in Figure 3-2 (b), the higher similarity as the layers of the model get deeper is related to the oversmoothing problem. To further analyze this issue, we also consider the dimensionality collapse in Transformer-based models. We plot the singular value distribution of the feature in the last block. As shown in Figure 3-2 (c), insignificant, near-zero values dominate the feature distribution. As layers get deeper, the similarity of features increases, and dimensional collapse occurs. The oversmoothing problem is the same in BERT and Graphormer, as shown in Figure 3-5 and Figure 3-6.



(a) Filter reponse (b) Cosine Similarity (c) Singular Value

Figure. 3-5: Filter frequency response, cosine similarity, and singular values on STS-B for BERT and BERT+GFSA



(a) Filter reponse (b) Cosine Similarity (c) Singular Value

Figure. 3-6: Filter frequency response, cosine similarity, and singular values on ZINC for Graphormer and Graphormer+GFSA

3.8.2 Analysis of Frequency Responses with Visualizations

We analyze the frequency responses, representing the impact of learned coefficients, for all 12 layers of $\text{BERT}_{\text{BASE}}$ with and without GFSA. From Figure 3-7, our analysis reveals that GFSA learns various filter types between layers. In early layers, we observe a tendency towards low-pass filtering, with prominent peaks at low frequencies. This aligns with the need for broader feature extraction in initial layers. The middle layers show low-pass and high-pass characteristics, with more complex frequency responses. This suggests GFSA is learning to balance between feature extraction and refinement. In deep layers, there is a noticeable shift towards higher frequency responses, indicating a move towards high-pass filtering. This shift supports our claim that GFSA can mitigate oversmoothing in deeper layers. $\text{BERT}_{\text{BASE}}$+GFSA shows a consistently higher magnitude response at higher frequencies, especially in deeper layers, than vanilla BERT. In other words, vanilla self-attention works primarily as a low-pass filter, while GFSA utilizes a wider range of frequencies.

3.8.3 Frequency Analyses in the Singular Value Domain

Graph signal processing (GSP) [114, 115] can be understood as a generalized concept of DSP — in other words, DSP is a special case of GSP where a *line graph with n nodes* is used. Therefore, the graph Fourier transform (GFT) of the line graph is identical to the discrete Fourier transform.

In the definition of GFT, we assume that the graph shift operator (GSO) $\boldsymbol{S}$ is diagonalizable. Considering the eigendecomposition of the GSO $\boldsymbol{S} = \boldsymbol{V}^\intercal \boldsymbol{\Lambda} \boldsymbol{V}$ with eigenvector $\boldsymbol{V}$, we can write the graph filter output as follows:

$$\boldsymbol{y} = \sum_{k=0}^{K} w_k \boldsymbol{S}^k \boldsymbol{x} = \sum_{k=0}^{K} \boldsymbol{V}^\intercal w_k \boldsymbol{\Lambda}^k \boldsymbol{V} \boldsymbol{x} = \boldsymbol{V}^\intercal \Big(\sum_{k=0}^{K} w_k \boldsymbol{\Lambda}^k \Big) \boldsymbol{V} \boldsymbol{x} = \boldsymbol{V}^\intercal g(\boldsymbol{\Lambda}) \boldsymbol{V} \boldsymbol{x}, \tag{3.11}$$

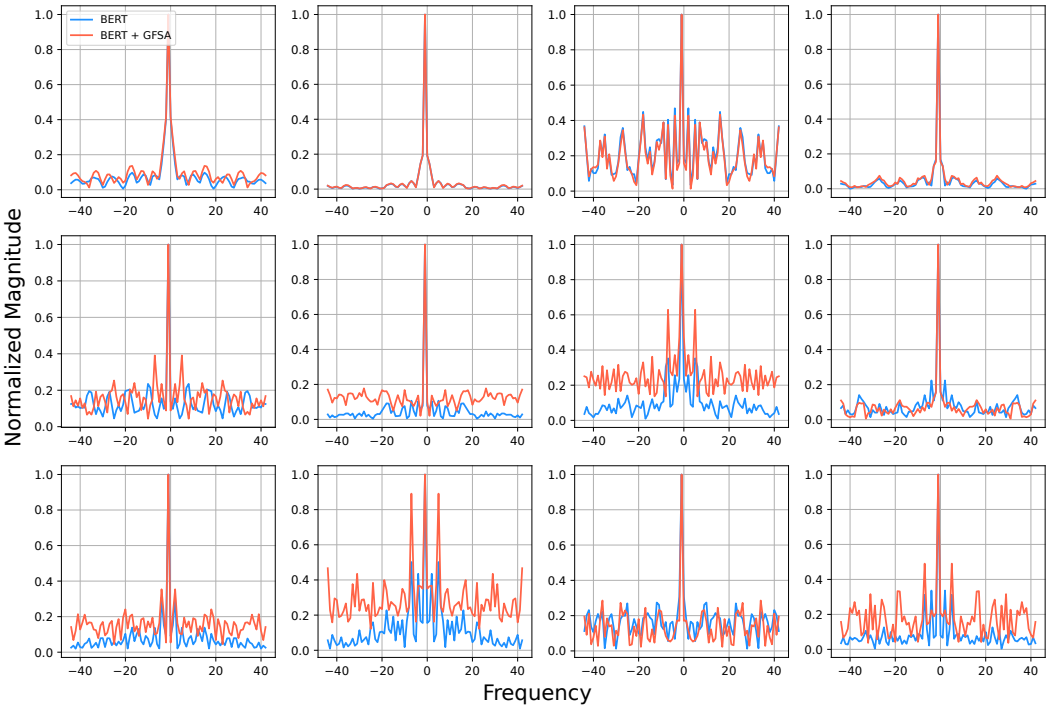


Figure. 3-7: Visualization of the frequency responses for all 12 layers of BERT trained on the STS-B dataset. The top-left figure corresponds to the first layer, and the bottom-right figure corresponds to the last layer.

where $\boldsymbol{x} \in \mathbb{R}^n$ is a 1-dimensional graph signal, $\boldsymbol{\Lambda}$ is a diagonal matrix with eigenvalues, and $w_k \in [-\infty, \infty]$ is a coefficient.

However, one can use the singular value decomposition when the GSO is not diagonalizable but symmetrically normalized, instead of the eigendecomposition [113]. Both the singular value decomposition and the eigendecomposition project the original signal onto a set of bases, but they use different basis sets. In the singular value decomposition, we sort the set of bases in ascending order of their eigenvalues, and perform frequency domain-like analyses [117, 113].

Since the self-attention matrix's row-wise sum is always 1, the following is the case: $\bar{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A} = \frac{1}{n}\boldsymbol{A}$, where n is the number of tokens. Maskey et al. [113] define the following symmetrically normalized adjacency matrix (SNA): $\boldsymbol{D}_{in}^{-1/2}\boldsymbol{A}\boldsymbol{D}_{out}^{-1/2}$. Since the degree of every node is n in the self-attention matrix, the following is the case: $\boldsymbol{D}_{in}^{-1/2}\boldsymbol{A}\boldsymbol{D}_{out}^{-1/2} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} = \frac{1}{\sqrt{n}}\boldsymbol{A}\frac{1}{\sqrt{n}} = \frac{1}{n}\boldsymbol{A} = \bar{\boldsymbol{A}}$. Therefore, the self-attention matrix is a special case of SNAs.

3.8.4 Matrix Polynomial vs. Graph Fourier Transform

There are two paradigms of implementing graph filters: i) matrix polynomial, which does not require diagonalizability, and ii) graph Fourier transform, which uses the eigendecomposition for diagonalizable adjacency matrices or uses the Jordan decomposition or the singular value decomposition for non-diagonalizable adjacency matrices.

Those two paradigms have their own weaknesses: i) the matrix polynomial approach requires explicit matrix multiplications, and ii) the graph Fourier transform approach requires expansive spectral decompositions. The matrix polynomial is preferred when there are not many matrix multiplications. Otherwise, the graph Fourier transform approach may be better since the matrix multiplication can be simplified after the decomposition.

Among those two, we use the first matrix polynomial approach with only three non-zero coefficients $\{w_0, w_1, w_K\}$ since it does not require the complicated spectral decomposition. Since we do not rely on any explicit spectral decomposition but on the matrix polynomial, any adjacency matrix can be used.

3.8.5 Proof of Theorem 3.3.1

Theorem 4.3.1 (Filter characteristics based on coefficient values)**.** *Let $\bar{\boldsymbol{A}}$ be a self-attention matrix interpreted as a graph with connected components. Consider the polynomial graph filter defined by $\sum_{k=0}^{K} w_k \bar{\boldsymbol{A}}^k$, where $w_2, w_3, \ldots, w_{K-1} = 0$ and only w_0, w_1, and w_K are non-zero. If the coefficients w_k for $k = 0, 1, K$ are positive and their sum is 1, then the polynomial filter acts as a low-pass filter, attenuating high-frequency components and promoting smoothness across the graph. Conversely, if $w_k = (-\alpha)^k$ for $k = 0, 1, K$ and $\alpha \in (0, 1)$ with sufficient large K, the polynomial filter exhibits high-pass filter behavior.*

Note that without filtering, the singular value ratio is $\left|\sigma_i^0\right| / \left|\sigma_1^0\right| = 1$. In the case where $|g(\sigma_i)/g(\sigma_1)| < 1 \; \forall i \geq 2$, it implies that after applying the graph filter g, the lowest frequency component further dominates, indicating that the graph filter acts as a low-pass filter. Conversely, in the case where $|g(\sigma_i)/g(\sigma_1)| > 1 \; \forall i \geq 2$, it implies that after applying the graph filter g, the lowest frequency component σ_i no longer dominates, indicating that the graph filter acts as a high-pass filter.

Proof. We prove the low-pass filter result. For the case where w_0, w_1, and w_K are positive and their sum is 1, we show that

$$|g(\sigma_1)| = |w_0 + w_1 + w_K| = 1 \tag{3.12}$$

Hence, proving Section 3.8.5 is equivalent to show $|g(\sigma_i)| < 1$.

$$|g(\sigma_i)| = \left|w_0 + w_1\sigma_i + w_K(\sigma_i + (K-1)(\sigma_i^2 - \sigma_i))\right| < |w_0 + w_1\sigma_i + w_K\sigma_i| = \sigma_i < 1 \tag{3.13}$$

since $\sigma_i + (K-1)(\sigma_i^2 - \sigma_i) = \sigma_i((K-1)\sigma_i - (K-2)) < \sigma_i((K-1) - (K-2)) = \sigma_i$

For the high-pass filter result, when $w_k = (-\alpha)^k/(k+1)$ where $k = 0, 1, K$ and $\alpha \in (0, 1)$, then we show that when $w_0 = 1, w_1 = -\alpha/2$,

$$\left|\frac{\lim_{K\to\infty} g(\sigma_i)}{\lim_{K\to\infty} g(\sigma_1)}\right| = \left|\frac{\lim_{K\to\infty} w_0 + w_1\sigma_i + w_K(\sigma_i + (K-1)(\sigma_i^2 - \sigma_i))}{\lim_{K\to\infty} w_0 + w_1 + w_K(1 + (K-1)(1-1))}\right| \tag{3.14}$$

$$= \left|\frac{\lim_{K\to\infty} 1 - \frac{\alpha}{2}\sigma_i + \frac{(-\alpha)^K}{(K+1)}(\sigma_i + (K-1)(\sigma_i^2 - \sigma_i))}{\lim_{K\to\infty} 1 - \frac{\alpha}{2} + \frac{(-\alpha)^K}{(K+1)}}\right| \tag{3.15}$$

$$= \left|\frac{1 - \frac{\alpha}{2}\sigma_i + (\lim_{K\to\infty} \frac{(-\alpha)^K}{(K+1)}(\sigma_i + (K-1)(\sigma_i^2 - \sigma_i)))}{1 - \frac{\alpha}{2}}\right| \tag{3.16}$$

$$= \left|\frac{1 - \frac{\alpha}{2}\sigma_i}{1 - \frac{\alpha}{2}}\right| > 1 \tag{3.17}$$

since

$$\lim_{K\to\infty} \frac{(-\alpha)^K}{(K+1)}(\sigma_i + (K-1)(\sigma_i^2 - \sigma_i)) = \lim_{K\to\infty} \frac{(-\alpha)^K}{(K+1)}(K-1)(\sigma_i^2 - \sigma_i) \tag{3.18}$$

$$= \lim_{K\to\infty} (-\alpha)^K \frac{(K+1)}{(K-1)}(\sigma_i^2 - \sigma_i) = 0 \tag{3.19}$$

It shows that the graph filter with $w_k = (-\alpha)^k/(k+1)$ for $k = 1, 2, K$ emphasizes high-frequency components and acts as a high-pass filter.

This proof supports that the behavior of the polynomial filter as either a low-pass or high-pass filter directly depends on the sign and values of the coefficients, as specified in Theorem 3.3.1. □

3.8.6 Proof of Theorem 3.4.1

Proof. The Frobenius norm of the self-attention is directly related to how far the softmax probabilities are from being uniform. For any matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$, we have

$$\|\text{softmax}(\boldsymbol{M})\|_F = \sqrt{\frac{m + \sum_{i=1}^m d_{\chi^2}(S_i, U_n)}{n}}, \tag{3.20}$$

where S_i is the i-th row of softmax(M), U_n is the uniform distribution over n elements, and $d_{\chi^2}(p, q) = \sum_i q_i(p_i/q_i - 1)^2$ is the χ^2-divergence between p and q [155]. The Frobenius norm is maximized when the whole mass of the probabilities is on one element, which is a case for $d_{\chi^2}(S_i, U_n) = n - 1$ and $\|\text{softmax}(\boldsymbol{M})\|_F = \sqrt{m}$. Therefore, we can calculate the upper bound of the Frobenius norm for $\bar{\boldsymbol{A}}$ as follows:

$$\|\bar{\boldsymbol{A}}\|_F \leq \sqrt{n}. \tag{3.21}$$

Note that $\bar{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$ is a right-stochastic matrix normalized with row-wise softmax: i) all the elements of $\bar{\boldsymbol{A}}$ lie within [0, 1], and ii) the row-sum in $\bar{\boldsymbol{A}}$ is equal to 1. Since the self-attention matrix is right-stochastic, the power of the self-attention is also a right-stochastic matrix. Therefore, Equation (3.21) is also hold for $\bar{\boldsymbol{A}}^K$ as follows:

$$\|\bar{\boldsymbol{A}}^K\|_F = \sqrt{\sum_{i,j} \bar{\boldsymbol{A}}^K_{i,j}} \leq \sqrt{\sum_{i,j} \bar{\boldsymbol{A}}_{i,j}} = \|\bar{\boldsymbol{A}}\|_F \leq \sqrt{n}. \tag{3.22}$$

Now, considering the error term E_K as given by Theorem 3.4.1, and applying the triangle inequality for matrix norms:

$$\begin{aligned} E_K &= \|\bar{\boldsymbol{A}}^K - (\bar{\boldsymbol{A}} + (K-1)(\bar{\boldsymbol{A}}^2 - \bar{\boldsymbol{A}}))\|_F && (3.23)\\ &\leq \|\bar{\boldsymbol{A}}^K\|_F + \|\bar{\boldsymbol{A}}\|_F + (K-1)(\|\bar{\boldsymbol{A}}^2\|_F + \|\bar{\boldsymbol{A}}\|_F) && (3.24)\\ &\leq \sqrt{n} + \sqrt{n} + (K-1)(\sqrt{n} + \sqrt{n}) = 2\sqrt{n}K. && (3.25) \end{aligned}$$

□

3.8.7 Implementation of GFSA

The pseudo code of our GFSA is shown in Algorithm 3.1. For implementation, w_0 and w_1 can be set as hyperparameters optionally.

Algorithm 3.1 PyTorch-style pseudocode for GFSA

```
w_0 = torch.zeros(h)
w_1 = torch.ones(h)
w_K = torch.zeros(h)
I = torch.eyes(n)[None, None, ...]

def GFSA (att, K)
  att: original self-attention
  att_K: high order term
  att_K = att + (K-1) * (torch.mm(att,att)-att)
  gf_att: GFSA attention
  gf_att = w_0[None, :, None, None] * I
           + w_1[None, :, None, None] * att
           + w_K[None, :, None, None] * att_K
return gf_att
```

3.8.8 Comparison with Actual and Approximated High-order Terms

To compare the impact of the actual $\bar{\boldsymbol{A}}^K$ and the approximated $\bar{\boldsymbol{A}}^K$ in terms of accuracy, we experimented with BERT on GLUE and the results are summarized in Table 3-9. $\text{BERT}_{\text{BASE}}$+$\bar{\boldsymbol{A}}^K$ denotes using the exactly calculated $\bar{\boldsymbol{A}}^K$ instead of the approximated $\bar{\boldsymbol{A}}^K$.

Table. 3-9: Comparison of performance using the exactly calculated $\bar{\boldsymbol{A}}^K$ vs. the approximated $\bar{\boldsymbol{A}}^K$ for GLUE tasks

Datasets	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	56.79	93.81	88.70	88.32	88.16	84.96/84.15	91.63	66.06	82.51
+ GFSA (approx. $\bar{\boldsymbol{A}}^K$)	59.56	94.15	90.60	88.46	88.33	85.12/85.06	91.95	68.95	83.58
+ GFSA (actual $\bar{\boldsymbol{A}}^K$)	59.85	94.27	89.80	88.43	88.32	84.95/84.89	91.76	68.23	83.39

3.8.9 Natural Language Understanding

Detailed Experimental Settings. We integrate GFSA into 3 pre-trained large language models: BERT, ALBERT, and RoBERTa. We evaluate them on the GLUE benchmark, which includes 3 categories of natural language understanding tasks: i) single-sentence tasks CoLA and SST-2; ii) similarity and paraphrasing tasks MRPC, QQP, and STS-B; iii) natural language inference tasks MNLI, QNLI, and RTE. For the MNLI task, we experiment on both the matched (MNLI-m) and mismatched (MNLI-mm) versions. Following Devlin et al. [86], we report Matthews correlation for CoLA, F1 scores for QQP and MRPC, Spearman correlations for STS-B, and accuracy scores for the other tasks. For each task, we select the best hyperparameters for GFSA, and the other hyperparameters are fixed. We compare our GFSA with ContraNorm [1], one of the related methods that address oversmoothing. We fine-tune ContraNorm with the recommended hyperparameters in Guo et al. [1]. We initialize with a pre-trained language model and fine-tune with added GFSA for 5 epochs.

Dataset. The benchmark datasets we used are listed below.

- **CoLA.** The Corpus of Linguistic Acceptability [156] consists of English acceptability judgments drawn from books and journal articles. The target task is a binary classification task, and each sentence is determined to be grammatically acceptable or not.
- **SST-2.** The Stanford Sentiment Treebank [157] is a dataset in which each sentence is sourced from movie reviews and accompanied by human annotations of its sentiment. The target task is to classify binary sentiments for a single sentence.
- **MRPC.** The Microsoft Research Paraphrase Corpus [158] is a corpus of sentence pairs, which are automatically extracted from online news sources and annotated by humans. The target is to determine whether the sentences in the pair are semantically equivalent.
- **QQP.** The Quora Question Pairs [159] dataset is a collection of question pairs from the community question-answering website Quora. The target is to determine whether the questions in the pair are semantically equivalent.

- **STS-B.** The Semantic Textual Similarity Benchmark [160] is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data with human annotation. The target is a regression task to predict a similarity score from 0 to 5.
- **MNLI.** The Multi-Genre Natural Language Inference Corpus [161] is a crowdsourced collection of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral). The standard test set consists of private labels from the authors and evaluates both the matched (in-domain) and mismatched (cross-domain) sections.
- **QNLI.** The Stanford Question Answering [162] dataset is a question-answering dataset consisting of question-paragraph pairs, where one of the sentences in the paragraph contains the answer to the corresponding question written by an annotator. The task is to determine whether the context sentence contains the answer to the question.
- **RTE.** The Recognizing Textual Entailment [163] dataset comes from a series of annual textual entailment challenges. The target task is a binary entailment classification task.

BERT. BERT [86] consists of 12 layers, 12 heads, 768 hidden size, 512 maximum sequence length, and an MLP dimension of 3072.

ALBERT. ALBERT [128] consists of 12 layers, 12 heads, 768 hidden dimensions, 512 maximum sequence length, 128 embedding dimensions, and an MLP dimension of 3072.

RoBERTa. RoBERTa [129] consists of 12 layers, 12 heads, 768 hidden size, 514 maximum sequence length, and an MLP dimension of 3072.

Training. For implementation, we adopt the HuggingFace framework. We trained all models with 5 epochs and 32 batch sizes. The linear learning rate decay is used, and the initial learning rate is set to 2×10^{-5}. We use AdamW [164] optimizer, and weight decay is set to 0. All models are trained on 1 GPU of NVIDIA RTX A5000 24GB.

3.8.10 Sensitivity to K

In this section, we explore the influence of the polynomial order, denoted as K, in our GFSA, conducting experiments on $\text{BERT}_{\text{BASE}}$ finetuned with GLUE tasks. We search for values of K from 2 to 10, and the results are presented in Table 3-10. For each dataset, there is an optimal K, and the performance of models using GFSA is generally robust to changes in K.

Table. 3-10: Sensitivity results on various K with $\text{BERT}_{\text{BASE}}$ finetuned on GLUE tasks

K	CoLA	SST2	MRPC	QQP	STSB	MNLI-m	MNLI-mm	QNLI	RTE
2	57.83	**94.15**	**90.60**	88.41	88.27	84.96	84.90	91.74	68.23
3	58.56	93.46	89.77	88.41	**88.33**	85.08	84.75	91.78	68.59
4	**59.56**	93.46	89.77	88.45	88.29	85.06	**85.06**	91.76	68.59
5	58.10	93.58	90.07	88.39	88.29	84.87	84.99	91.85	68.95
6	59.40	93.58	90.40	88.29	88.27	84.93	84.97	91.69	68.23
7	59.12	94.04	90.48	88.43	88.26	**85.12**	84.94	91.82	68.94
8	58.58	93.69	90.12	**88.46**	88.24	84.92	84.81	**91.95**	**68.95**
9	58.88	93.46	89.54	88.41	88.26	85.06	91.67	67.87	85.04
10	59.31	93.35	89.98	88.41	88.30	84.84	91.73	68.59	84.93

3.8.11 Causal Language Modeling

3.8.11.1 Detailed Experimental Settings

Dataset. The benchmark datasets we used are listed below.

- **PTB.** Penn Treebank [130] dataset is a collection of text documents that have been extensively annotated with linguistic information, primarily syntactic and grammatical structures.
- **WikiText.** WikiText [131] dataset is a collection of over 100 million tokens extracted from the set of verified good and featured articles on Wikipedia. Compared to the preprocessed version of Penn Treebank (PTB), WikiText-2 is over 2 times larger, and WikiText-103 is over 110 times larger.

GPT2. GPT2 [88] is a Transformer pretrained on a very large corpus of English data in a self-supervised fashion without any human labeling on the dataset. It automatically generates inputs and labels from those texts and is trained to guess the next word in sentences. For implementation, we adopt the HuggingFace Framework [4]. For all experiments, GPT2 has 12 layers with 12 attention heads, 768 hidden size, and 1024 maximum sequence length, resulting in a total of 117 million parameters.

Training. We finetune GPT2 with 4 batch size, 5×10^{-5} learning rate and linear learning weight decay using adamW [164] optimizer. We also apply dropout with a probability of 0.1. Following [132], we train models for 15 epochs with PTB, 4 epochs with WikiText-103, and 10 epochs with WikiText-2. We use a sensitivity metric, i.e., perplexity, which is a commonly used metric to evaluate the performance of language models, particularly in

[4] `https://github.com/huggingface/transformers`

language modeling and text generation tasks. Perplexity measures how well a language model can predict a sequence of words in a given text or a test dataset. All the experiments are conducted on 1 GPU of NVIDIA RTX 3090 24GB.

3.8.11.2 Sensitivity to K

We conducted a sensitivity study on K of GPT-2 across all datasets, and the results are presented in Table 3-11. For PTB and WikiText-2, GFSA exhibits the best performance when K is high, typically around 8 or 9. However, for WikiText-103, GFSA achieves the best perplexity when K is small, specifically when K is 3 or 4.

Table. 3-11: Sensitivity to K on GPT-2 finetuned with GFSA

Method	#Params	PTB	WikiText-2	WikiText-103
GPT2 [88]	117M	19.513	20.966	15.939
GPT2 + GFSA($K = 2$)	117M	19.459	20.929	15.920
GPT2 + GFSA($K = 3$)	117M	19.456	20.927	**15.919**
GPT2 + GFSA($K = 4$)	117M	19.453	20.927	**15.919**
GPT2 + GFSA($K = 5$)	117M	19.452	20.925	15.920
GPT2 + GFSA($K = 6$)	117M	19.451	20.925	15.920
GPT2 + GFSA($K = 7$)	117M	**19.450**	20.925	15.921
GPT2 + GFSA($K = 8$)	117M	**19.450**	20.924	15.921
GPT2 + GFSA($K = 9$)	117M	**19.450**	**20.923**	15.921

3.8.12 Image Classification

3.8.12.1 Detailed Experimental Settings

Our code is implemented based on the timm library [165]. In the case of our training recipe, it is the same as the experimental setting of Wang et al. [103] that follows the training recipes of Touvron et al. [90] and Touvron et al. [133]. To apply our GFSA to existing base models such as DeiT, Cait, and Swin, we consider a range of K between 2 and 5. For 12-layer DeiT, we follow the same hyperparameters from Wang et al. [103]. We set the dropout rate to 0 and 0.2 for 12-layer and 24-layer DeiT, respectively. For CaiT, we apply our GFSA only to the patch embedding layer. All other hyperparameters are kept consistent with the original papers of DeiT [90], CaiT [133], and Swin [92]. All models are trained on NVIDIA RTX 3090 24GB.

3.8.12.2 FLOPs & Throughput

In Table 3-12, we report the number of FLOPs and throughput. With GFSA plugged in, the FLOP count is either the same or not different. For DeiT-S with 24 layers, there is a slight

FLOP increase with GFSA plugged in. However, for the rest of the settings, the models have the same number of Flops. For throughput, it tends to decrease because calculating the high-order term is an additional cost.

Table. 3-12: Experimental evaluation of GFSA plugged into DeiT-S, CaiT-S, and Swin-S

Backbone	Method	Input Size	#Layers	#Params	#FLOPs	#Throughput	Top-1 Acc
DeiT	DeiT-S	224	12	22.0M	4.57G	856.07	79.8
	+ GFSA	224	12	22.0M	4.57G	614.54	**81.1** (↑ 1.3)
	DeiT-S	224	24	43.3M	9.09G	423.68	80.5
	+ GFSA	224	24	43.3M	9.10G	314.75	**81.5** (↑ 1.0)
CaiT	CaiT-S	224	24	46.9M	9.34G	574.66	82.6
	+ GFSA	224	24	47.0M	9.34G	406.96	**82.8** (↑ 0.2)
Swin	Swin-S	224	24	49.6M	8.75G	912.38	82.9
	+ GFSA	224	24	49.6M	8.75G	714.60	**83.0** (↑ 0.1)

3.8.12.3 Sensitivity to K

We also perform the sensitivity analysis for K. Tables 3-13 and 3-14 show the results of the sensitivity analysis for DeiT-S and CaiT-S with GFSA plugged in. For 12-layer DeiT-S, the GFSA performance of 81.12 is highest when $K = 3$. When the GFSA has a K of 2, the performance is worse than the original DeiT-S, but when the K is 3 or higher, the performance is better than the original DeiT-S, and most surprisingly, the performance is better than the 24-layer DeiT-S.

CaiT-S shows the highest performance of 82.84 when $K = 4$. For CaiT-S, the accuracy is slightly lower than that of the original CaiT-S when $K = 2$, but it starts to exceed the accuracy of CaiT-S when K is 3 or higher.

Table. 3-13: Sensitivity to K for 12-layer DeiT-S + GFSA

K	2	3	4	5
Top-1 Acc (%)	79.27	**81.12**	80.86	81.07

Table. 3-14: Sensitivity to K for 24-layer CaiT-S + GFSA

K	2	3	4
Top-1 Acc (%)	82.54	82.65	**82.84**

3.8.12.4 Additional Comparison with SOTA Models

Our main experiment aims to determine whether introducing the GFSA layer would help improve performance in a base model, such as DeiT. We also compare our method with the recent models: SpectFormer [166], SVT [167], NeuTRENO [168], FNet [169], and GFNet [170]. We use a 12-layer setup to ensure a fair comparison in Table 3-15.

GFNet [170] can reduce the number of parameters, but there is a performance penalty. However, the performance improvement of DeiT-S+GFSA is relatively greater than that of DeiT-S compared to other models. SpectFormer [166] and SVT [167] have advantages in calculation amount and model complexity, and performance is improved over DeiT-S, but Top-1 and Top-5 accuracies are lower than those using GFSA. Additionally, NeuTRENO [168] also improves as much as GFSA compared to DeiT-S, but GFSA still has higher Top-1 accuracy.

Table. 3-15: Compared with state-of-the-art models on ImageNet-1k

Method	Input Size	#Layers	#Params	Top-1 Acc	Top-5 Acc
DeiT-S	224	12	22M	79.8	95.0
Fnet-XS [169]	224	12	20M	71.2	-
GFNet-XS [170]	224	12	16M	78.6	94.2
SpectFormer-XS [166]	224	12	20M	80.2	94.7
SVT-XS [167]	224	12	20M	79.9	94.5
DeiT-S + NeuTRENO [168]	224	12	20M	80.7	95.4
DeiT-S + GFSA	224	12	22M	**81.1**	95.4

3.8.12.5 Additional Experiments with Guo et al. [1]'s setting

To make a fair comparison with ContraNorm [1], one of the related studies that mitigates oversmoothing, we run additional experiments to match their experimental setup.

Setting. We follow the training recipe used by Guo et al. [1], a slightly modified version of Touvron et al. [90]'s recipe. Guo et al. [1] use AdamW optimizer with cosine learning rate decay. We select the DeiT-T and DeiT-S for ImageNet-1k. "T" and "S" denote tiny and small model sizes, respectively. For all experiments, the image size is set to be 224x224. We train each model for 300 epochs, and the batch size is 1024. For ContraNorm, we train with their recommended hyperparameters. All models are trained on 4 GPUs of NVIDIA RTX A6000 48GB.

Results. In Table 3-16, DeiT-T and DeiT-S with GFSA outperform vanilla DeiT-T and DeiT-S in all layer settings. GFSA improves the performance of DeiT-T with 12 layers by 1.52%. The largest gain is a 4.88% improvement on 16-layer DeiT-T. This shows that

the effect of GFSA is larger than the effect of ContraNorm. For DeiT-S with 16 layers, surprisingly, GFSA can increase the performance by 80.83%, meaning that GFSA brings benefits with a 3.23% improvement.

Table. 3-16: Experiment results on ImageNet-1k

Method	#Layers=12	#Layers=16	#Layers=24
DeiT-T	76.52	75.34	76.76
DeiT-T + ContraNorm	77.03	78.72	78.12
DeiT-T + GFSA	**77.68**	**79.02**	**78.64**
DeiT-S	77.32	78.25	77.69
DeiT-S + ContraNorm	77.80	79.04	78.67
DeiT-S + GFSA	**79.86**	**80.83**	**79.15**

Sensitivity to K. In Table 3-17, we experiment with a sensitivity analysis for K. For DeiT-T, the performance of GFSA generally improves when K is 4 or 5. On the other hand, GFSA performs better at lower K for settings that are layers 16 and 24 for DeiT-S.

Table. 3-17: Varying K for DeiT-T and DeiT-S

Method	K	#Layers=12	#Layers=16	#Layers=24
DeiT-T + GFSA	2	76.92	78.14	78.40
DeiT-T + GFSA	3	77.41	77.76	78.09
DeiT-T + GFSA	4	77.01	**79.02**	**78.64**
DeiT-T + GFSA	5	**77.68**	78.14	**78.64**
DeiT-S + GFSA	2	79.84	**80.83**	**79.15**
DeiT-S + GFSA	3	79.85	79.39	79.07
DeiT-S + GFSA	4	**79.86**	79.44	79.10

3.8.13 Automatic Speech Recognition

3.8.13.1 Detailed Experimental Settings

Dataset. We conduct experiments on the LibriSpeech [5] dataset [145], which consists of audio recordings paired with their transcriptions. The LibriSpeech dataset has approximately 1,000 hours of read English speech with a sampling rate of 16 kHz. We keep the original 16,000Hz sampling rate and compute 80-dim log-Mel filterbanks for a 25ms sliding window, strided by 10ms. The filterbank features are then normalized to zero mean and unit variance per input sequence. For implementation, we follow the recipes of SpeechBrain [146].

[5] `http://www.openslr.org/12`

Evaluation Metric. Word error rate (WER (%)) is derived from the Levenshtein distance and compares a reference to a hypothesized word-level transcription. It is calculated by summing the number of word insertions, deletions, and substitutions and dividing it by the total number of words in the reference transcription.

Vanilla Transformer. We use a vanilla Transformer to apply our GFSA. For implementation, we use a SpeechBrain [146] framework. The vanilla Transformer consists of i) 1D convolution to perform striding, ii) Transformer encoder with 12 layers, 4 heads, embedding dimension of 512, MLP dimension of 2048, and post-LayerNorm iii) decoder with 6 layers, 4 heads, embedding dimension of 512, MLP dimension of 2048, joint beamsearch, and iv) external Transformer language model with 12 layers, 12 heads, embedding dimension of 768, and MLP dimension of 3072.

Branchformer. We use one of the SOTA models, Branchformer [97], to plug into our GFSA. Branchformer has two parallel branches, one for capturing global interactions using attention and the other for more localized context using a convolutional gating MLP. The Branchformer architecture for speech recognition consists of i) 1D convolution to perform striding, ii) Branchformer encoder with 18 layers, 8 heads, embedding dimension of 512, and MLP dimension of 3072, iii) decoder with 6 layers, 8 heads, embedding dimension of 512, a convolutional spatial gating unit (CSGU) dimension of 3072, joint beamsearch, and iv) external Transformer language model with 12 layers, 12 heads, embedding dimension of 768, and MLP dimension of 3072.

Training. We follow a training recipe from SpeechBrain [146]. The standard LibriSpeech validation sets (dev-clean and dev-other) are used to tune all parameters and select the best models. Test sets (test-clean and test-other) are used only to report final WER performance. We train the pure Transformer for 100 epochs and the Branchformer for 120 epochs with a batch size of 16. We use a data augmentation method on all models using SpecAugment [171]. SpecAugment applies time and frequency masking as well as time warping to the input spectrum. For Branchformer, we use AdamW [164] optimizer with 0.9 and 0.98 coefficients for computing running averages of gradient and its square. The learning rate and weight decay in all models are 0.0008 and 0.01, respectively. We use a connectionist temporal classification (CTC) loss [172, 173]. We also apply dropout with a probability of 0.1 and label smoothing with a weight of 0.1 to mitigate overfitting. We fix the random seed to 74,443 on all experiments. All models are trained on 1 GPU of NVIDIA RTX A6000 48GB.

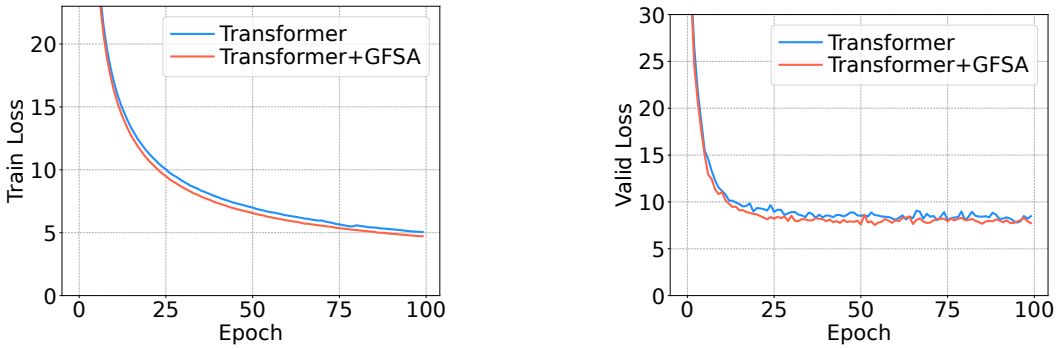
Hyperparameters. In Table 3-18, we describe the main hyperparameters used in the automatic speech recognition task. For Transformer+GFSA and Branchformer+GFSA, we also report the best K hyperparameter.

Table. 3-18: Main hyperparameters used in ASR

Model	**Experimental Setting**
Transformer	Encoder: Transformer (12 layers) Decoder: Transformer (6 layers) + (CTC/ATT joint) beamsearch + TransformerLM Augmentation: SpecAugment Features: 40 fbanks Pretraining: no Dropout: 0.1 Batchnorm: yes Number of epochs: 100 Batch size: 32 Learning rate: 0.0008 LR scheduler: Noam Optimizer: Adam Loss: CTC + KLdiv (Label Smoothing loss) CTC weight: 0.3
Transformer+GFSA	Encoder: Transformer (12 layers) Decoder: Transformer (6 layers) + (CTC/ATT joint) beamsearch + TransformerLM Augmentation: SpecAugment Features: 40 fbanks Pretraining: no Dropout: 0.1 Batchnorm: yes Number of epochs: 100 Batch size: 32 Learning rate: 0.0008 LR scheduler: Noam Optimizer: Adam Loss: CTC + KLdiv (Label Smoothing loss) CTC weight: 0.3 K: 2
Branchformer	Encoder: Branchformer Decoder: Transformer (6 layers) + (CTC/ATT joint) beamsearch + TransformerLM Augmentation: SpecAugment Features: 40 fbanks Pretraining: no Dropout: 0.1 Batchnorm: yes Number of epochs: 120 Batch size: 16 Learning rate: 0.0008 LR scheduler: Noam Optimizer: AdamW with coefficients 0.9 and 0.98 Loss: CTC + KLdiv (Label Smoothing loss) CTC weight: 0.3
Branchformer+GFSA	Encoder: Branchformer Decoder: Transformer (6 layers) + (CTC/ATT joint) beamsearch + TransformerLM Augmentation: SpecAugment Features: 40 fbanks Pretraining: no Dropout: 0.1 Batchnorm: yes Number of epochs: 120 Batch size: 16 Learning rate: 0.0008 LR scheduler: Noam Optimizer: AdamW with coefficients 0.9 and 0.98 Loss: CTC + KLdiv (Label Smoothing loss) CTC weight: 0.3 K: 3

3.8.13.2 Training Curve

We compare the training and validation curves for LibriSpeech 100h in Figure 3-8. The training loss curve of GFSA is lower than that of the pure Transformer. GFSA stabilizes the loss curve of the pure Transformer slightly earlier.



(a) Training loss curve

(b) Validation loss curve

Figure. 3-8: Training curve on LibriSpeech 100h

3.8.14 Graph-level Tasks

3.8.14.1 Detailed Experimental Settings

Experimental settings for Graphormer. We describe benchmark datasets and Graphormer as the backbone model we used. ZINC [174] is the most popular real-world molecular dataset to predict graph property regression for constrained solubility, an important chemical property for designing generative GNNs for molecules. Uniform sampling is adopted for data splitting. We use a ZINC-subset of a small-scale dataset. PCQM4M-LSC [142] is a 2D molecular graph, one of the most practically relevant quantum chemical properties of molecular science. The task is to predict the density functional theory (DFT)-calculated HOMO-LUMO energy gap of molecules given their graphs. PCQM4M-LSC is unprecedentedly large in scale compared to other labeled graph-level prediction datasets, which contain more than 3.8M graphs. We use PCQM4M and PCQM4Mv2 large-scale datasets.

Following Graphormer [143], we use Graphormer for PCQM4M and $\text{Graphormer}_{\text{SLIM}}$ for ZINC. Graphormer consists of 12 encoder layers, 80 encoder embedding dimensions, and 768 MLP dimensions. It employs 32 encoder heads and 24 hidden dimensions for each head. $\text{Graphormer}_{\text{SLIM}}$ consists of 12 encoder layers, 80 encoder embedding dimensions, and 80 MLP dimensions. It employs 8 encoder heads and 10 hidden dimensions for each head. We use adamW [164] optimizer with 0.9 and 0.999 coefficients for running averages of gradient and its square, and use Mean Absolute Error (MAE) as a loss function. We use polynomial

learning rate decay, with initial learning rate set to 2×10^{-4} and end learning rate set to 1×10^{-9}. For ZINC, we set batch size as 256, max epochs as 10k, and warm-up stage step as 40k. For PCQM4M and PCQM4Mv2, we set the batch size as 1024, the max epochs as 300, and the warm-up stage step as 60k. All models are trained on 1 GPU of NVIDIA RTX 3090 24GB. We conduct experiments with 4 different seeds.

Experimental settings for GPS and Graph-ViT. We use various benchmark datasets to experiment with our GFSA on GPS [95] and Graph-ViT [144]: Peptide-func and Peptide-struct from Long-Range Graph Benchmark (LRGB) [139], MNIST, CIFAR10, and ZINC from Benchmarking GNNs [140], and Moltox21 and Molhiv from OGB [141]. We set all hyperparameters of GPS and Graph-ViT to the values recommended in their respective papers to ensure a fair comparison. To plug GFSA into GPS, we replace the self-attention module of GPS with GFSA. For Graph-ViT, we apply GFSA instead of the Hadamard self-attention mechanism and compare it with this self-attention method. We conduct experiments on GPS and Graph-ViT in their open code frameworks for a fair comparison:

- GPS: `https://github.com/rampasek/GraphGPS`
- Graph-ViT: `https://github.com/XiaoxinHe/Graph-ViT-MLPMixer`

3.8.15 Code Defect Detection

3.8.15.1 Detailed Experimental Settings

Dataset. We use the Devign dataset provided by [147], which is a binary classification task to evaluate whether a C language code is vulnerable to software systems or not.

Implementation. We build our experiments on top of the open-sourced code [6] and recipes provided by Wang et al. [150].

RoBERTa. RoBERTa [129] is an encoder-only model trained with masked language modeling on code. All hyperparameters are consistent with the training method in the source code of Wang et al. [150].

PLBART. PLBART [149] is an encoder-decoder model based on BART [175] architecture. PLBART can support understanding and generation tasks. All hyperparameters are consistent with the training method in the source code of Wang et al. [150].

[6] `https://github.com/salesforce/CodeT5`

CodeBert. CodeBERT [148] is a model trained on masked language modeling and replaced token detection. CodeBERT is a bimodal pretrained model based on the Transformer with 12 layers for programming language and natural language. All hyperparameters are consistent with the training method in the source code of Wang et al. [150].

CodeT5. CodeT5 is an encoder-decoder framework with the same architecture as T5 [176]. It aims to derive generic representations for programming languages and natural languages via pre-training on unlabeled source code. CodeT5-small has 6 encoder layers, 6 decoder layers, 8 attention heads, 512-dimensional hidden states, and 60M parameters. The other models have 12 encoder layers, 12 decoder layers, 12 attention heads, 768-dimensional hidden states, and 220M parameters. All hyperparameters are consistent with the training method in the source code of Wang et al. [150].

Training. The pre-trained models mentioned above are applied to this downstream task. We add GFSA directly on top of self-attention. We fine-tune baselines and GFSA models for 10 epochs with a batch size of 16. We use an early stopping strategy with a patience of 2. Models generate binary labels from unigram sequences at the decoder for the defect detection task. We employ accuracy when evaluating the code defect detection task. All models are trained on 1 GPU of NVIDIA RTX A6000 48GB.

3.8.15.2 Case Study

In Listing 1, we show one of the cases for code snippets of defects in QEMU [7] that CodeT5-base does not predict correctly, but that *only* CodeT5-base+GFSA predicts. The commit message [8] for this case is as follow:

> Needed for changing `cpu_has_work()` argument type to CPUState, used in `h_cede()`.

`h_cede()` is the hypercall that asks the hypervisor to shut down the CPU. Previously, this hypercall passed the CPUID, so the hypervisor did not know what state the CPU was in. This change allows the hypervisor to know whether the CPU is performing work. If the CPU performs a task, the hypervisor waits for the CPU to complete the task.

In this context, accurately predicting defects like the one above is very important, and applying GFSA to CodeT5-base helps in terms of performance improvement.

```
1 @@ -204,7 +204,7 @@ static target_ulong put_tce_emu(sPAPRTCETable *
    tcet, target_ulong ioba,
2 - static target_ulong h_put_tce(CPUPPCState *env, sPAPREnvironment *
    spapr
```

[7] https://www.qemu.org
[8] https://github.com/qemu/qemu/commit/b13ce26d3e8c6682044ae84920f2417b30ce356b

```
+ static target_ulong h_put_tce(PowerPCCPU *cpu, sPAPREnvironment *
    spapr
                               , target_ulong opcode, target_ulong *
    args)
{
  target_ulong liobn = args[0];
  target_ulong ioba = args[1];
  target_ulong tce = args[2];
  VIOsPAPRDevice *dev = spapr_vio_find_by_reg(spapr->vio_bus, liobn);
  VIOsPAPR_RTCE *rtce;
  if (!dev) {
    hcall_dprintf("LIOBN 0x" TARGET_FMT_lx " does not exist\n", liobn)
    ;
    return H_PARAMETER;
  }
  ioba &= ~(SPAPR_VIO_TCE_PAGE_SIZE - 1);
  #ifdef DEBUG_TCE
    fprintf(stderr, "spapr_vio_put_tce on %s ioba 0x" TARGET_FMT_lx "
    TCE 0x" TARGET_FMT_lx "\n", dev->qdev.id, ioba, tce);
  #endif
  if (ioba >= dev->rtce_window_size) {
    hcall_dprintf("Out-of-bounds IOBA 0x" TARGET_FMT_lx "\n", ioba);
    return H_PARAMETER;
  }
  rtce = dev->rtce_table + (ioba >> SPAPR_VIO_TCE_PAGE_SHIFT);
  rtce->tce = tce;
  return H_SUCCESS;
}
```

Listing 3.1: An example commit history for defects in Devign dataset

3.8.15.3 Code Clone Detection

Dataset. Code clone detection aims to measure the similarity between two code snippets and predict whether they have the same functionality. We experiment with the Java data provided by Wang et al. [177].

Implementation. We build our experiments on top of the open-sourced code [9] and recipes provided by Wang et al. [150].

[9] https://github.com/salesforce/CodeT5

Training. We fine-tune both CodeT5 and CodeT5+GFSA for one epoch with a batch size 16. We also use early stopping with a patience of 2. CodeT5 and CodeT5+GFSA encode source code and take the representation to calculate the similarity of two code snippets. We employ the F1 score to evaluate this task. All models are trained on 1 GPU of NVIDIA RTX A6000 48GB.

Experiment Result. Table 3-19 shows results comparing CodeT5 and CodeT5 with GFSA. The result shows that by using our GFSA, CodeT5 models improve their performance. CodeT5-small+GFSA provides a 0.61% improvement over Code5T-small.

Table. 3-19: Results on the code clone detection task

Method	Clone F1
CodeT5-small [150]	94.36
CodeT5-small + GFSA	**94.94** (↑ 0.61%)
CodeT5-base [150]	94.31
CodeT5-base + GFSA	**94.92** (↑ 0.64%)

3.8.15.4 Time Complexity and Empirical Runtime Analysis

Time Complexity. The time complexity of the original self-attention is $\mathcal{O}(n^2 d)$. But our GFSA has a high-order term. Therefore, the time complexity of GFSA has $\mathcal{O}(n^2 d + n^3)$. If n is smaller than d, the time complexity approaches $\mathcal{O}(n^2 d)$, which is the complexity of the original self-attention.

Empirical Runtime Analysis. We report the training time of various methods with GFSA in Tables 3-20 and 3-21 to 3-25. Generally, the training time of methods using GFSA is slightly longer than that of existing methods. For example, the Transformer for the automatic speech recognition task increases from 190.5 seconds to 191.6 seconds on the Librispeech 100h dataset, as an increase of only 1 second. Instead of computing higher-order polynomial terms, our GFSA approximates them, with only a small increase in runtime, which is insignificant.

Table. 3-20: Training time (seconds per epoch) on GLUE tasks. s denotes the abbreviation for second. **Avg** denotes the average training time across all tasks.

Datasets	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	17s	182s	17s	1483s	24s	2004s	580s	18s	541s
$\text{BERT}_{\text{BASE}}$ + GFSA	19s	192s	19s	1571s	25s	2147s	621s	20s	577s
$\text{ALBERT}_{\text{BASE}}$ [128]	15s	188s	20s	1506s	25s	2072s	612s	19s	557s
$\text{ALBERT}_{\text{BASE}}$ + GFSA	16s	197s	21s	1604s	26s	2219s	659s	21s	595s
$\text{RoBERTa}_{\text{BASE}}$ [129]	17s	190s	18s	1492s	25s	2012s	593s	18s	546s
$\text{RoBERTa}_{\text{BASE}}$ + GFSA	19s	200s	19s	1580s	26s	2151s	634s	20s	581s

Table. 3-21: Training time (seconds per epoch) on causal language modeling tasks.

Method	#Params	PTB	WikiText-2	WikiText-103	**Avg**
GPT2 [88]	117M	89.1s	195.7s	9638.4s	3307.8s
GPT2 + GFSA	117M	160.3s	354.2s	17424.6s	5979.7s

Table. 3-22: Training time (seconds per epoch) on ImageNet-1k

Backbone	Method	#Layers	#Params	#FLOPs	#Throughput	Runtime
DeiT	DeiT-S	12	22.0M	4.57G	856.07	551s
	DeiT-S + GFSA	12	22.0M	4.57G	614.54	814s
	DeiT-S	24	43.3M	9.09G	423.68	1508s
	DeiT-S + GFSA	24	43.3M	9.10G	314.75	1798s
CaiT	CaiT-S	24	46.9M	9.34G	574.66	1530s
	CaiT-S + GFSA	24	47.0M	9.34G	406.96	1624s
Swin	Swin-S	24	49.6M	8.75G	912.38	1897s
	Swin-S + GFSA	24	49.6M	8.75G	714.60	1970s

Table. 3-23: Training time (seconds per epoch) on graph-level tasks

Method	ZINC	PCQM4M	PCQM4Mv2
Graphormer [143]	9s	740s	817s
Graphormer + GFSA	9s	896s	955s

Table. 3-24: Training time (seconds per epoch) on LibriSpeech datasets

Method	LibriSpeech 100h	LibriSpeech 960h
Transformer	190.5s	3049.3s
Transformer + GFSA	191.6s	3398.4s
Branchformer [97]	248.5s	4990.1s
Branchformer + GFSA	254.3s	4999.3s

Table. 3-25: Training time (seconds per epoch) on the code defect prediction task

Method	Runtime
RoBERTa [129]	543.96s
RoBERTa + GFSA	537.79s
CodeBERT [148]	555.28s
CodeBERT + GFSA	561.43s
PLBART [149]	467.80s
PLBART + GFSA	470.19s
CodeT5-small [150]	301.11s
CodeT5-small + GFSA	309.04s
CodeT5-base [150]	362.28s
CodeT5-base + GFSA	373.22s

3.8.16 Inference Time Analysis

We report the inference time of various methods with GFSA in Tables 3-26 to 3-31.

Table. 3-26: Inference time on GLUE tasks. *s* denotes the abbreviation for second. **Avg** denotes the average training time across all tasks.

Datasets	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	110M	1.0*s*	1.4*s*	1.2*s*	48.7*s*	1.9*s*	15.5*s*	10.1*s*	1.2*s*	10.0*s*
$\text{BERT}_{\text{BASE}}$ + GFSA	110M	1.1*s*	1.4*s*	1.2*s*	52.3*s*	2.0*s*	16.8*s*	11.0*s*	1.3*s*	11.0*s*
$\text{ALBERT}_{\text{BASE}}$ [128]	11M	1.1*s*	1.6*s*	1.4*s*	58.4*s*	2.2*s*	18.4*s*	12.1*s*	1.3*s*	12.0*s*
$\text{ALBERT}_{\text{BASE}}$ + GFSA	11M	1.2*s*	1.7*s*	1.4*s*	62.1*s*	2.3*s*	19.7*s*	13.1*s*	1.4*s*	13.0*s*
$\text{RoBERTa}_{\text{BASE}}$ [129]	125M	1.0*s*	1.4*s*	1.1*s*	47.0*s*	1.9*s*	15.0*s*	9.9*s*	1.2*s*	10.0*s*
$\text{RoBERTa}_{\text{BASE}}$ + GFSA	125M	1.1*s*	1.4*s*	1.2*s*	50.4*s*	2.0*s*	16.3*s*	10.8*s*	1.2*s*	11.0*s*

Table. 3-27: Inference time on causal language modeling tasks.

Method	#Params	PTB	WikiText-2	WikiText-103	**Avg**
GPT2 [88]	117M	3.2*s*	7.4*s*	7.4*s*	6.0*s*
GPT2 + GFSA	117M	5.5*s*	12.9*s*	12.9*s*	10.4*s*

Table. 3-28: Inference time on ImageNet-1k

Backbone	Method	#Layers	Inference Time
DeiT	DeiT-S	12	52*s*
	DeiT-S + GFSA	12	53*s*
	DeiT-S	24	68*s*
	DeiT-S + GFSA	24	69*s*
CaiT	CaiT-S	24	105*s*
	CaiT-S + GFSA	24	107*s*
Swin	Swin-S	24	17*s*
	Swin-S + GFSA	24	17*s*

Table. 3-29: Inference time on graph-level tasks

Method	ZINC	PCQM4M	PCQM4Mv2
Graphormer [143]	8*s*	99*s*	31*s*
Graphormer + GFSA	8*s*	117*s*	29*s*

Table. 3-30: Inference time on LibriSpeech datasets

Method	LibriSpeech 100h	LibriSpeech 960h
Transformer	328.1s	323.7s
Transformer + GFSA	329.5s	343.3s
Branchformer [97]	299.4s	328.7s
Branchformer + GFSA	305.5s	354.1s

Table. 3-31: Inference time on the code defect prediction task

Method	Inference Time
RoBERTa [129]	22.4s
RoBERTa + GFSA	23.9s
CodeBERT [148]	23.8s
CodeBERT + GFSA	24.1s
PLBART [149]	37.7s
PLBART + GFSA	39.3s
CodeT5-small [150]	78.2s
CodeT5-small + GFSA	82.5s
CodeT5-base [150]	83.2s
CodeT5-base + GFSA	88.5s

3.8.17 Results of the Strategy for Efficiency

In Tables 3-32 to 3-36, the results show that this strategy can reduce the increase in runtime and maintain performance compared to using GFSA for all layers.

Table. 3-32: Comparison of performance using GFSA on all layers vs. $\text{GFSA}_{\text{even}}$ on even layers for GLUE tasks

Datasets	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	110M	56.79	93.81	88.70	88.32	88.16	84.96/84.15	91.63	66.06	82.51
+ GFSA	110M	59.56	94.15	90.60	88.46	88.33	85.12/85.06	91.95	68.95	83.58
+ $\text{GFSA}_{\text{even}}$	110M	58.80	93.69	90.50	88.47	88.27	85.13/85.02	91.65	70.76	83.59

Table. 3-33: Comparison of training time (seconds per epoch) using GFSA on all layers vs. $\text{GFSA}_{\text{even}}$ on even layers for GLUE tasks

Datasets	#Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI-m/mm	QNLI	RTE	**Avg**
$\text{BERT}_{\text{BASE}}$ [86]	110M	17s	182s	17s	1483s	24s	2004s	580s	18s	541s
+ GFSA	110M	19s	192s	19s	1571s	25s	2147s	621s	20s	577s
+ $\text{GFSA}_{\text{even}}$	110M	17s	185s	18s	1506s	24s	2061s	595s	18s	553s

Table. 3-34: Comparison of performance using GFSA on all layers vs. $\text{GFSA}_{\text{even}}$ on even layers for causal language modeling tasks

Method	#Params	PTB	WikiText-2	WikiText-103	**Avg**
GPT2 [88]	117M	19.513	20.966	15.939	18.806
+ GFSA	117M	19.450	20.923	15.919	18.764
+ $\text{GFSA}_{\text{even}}$	117M	19.453	20.926	15.923	18.767

Table. 3-35: Comparison of training time (seconds per epoch) using GFSA on all layers vs. $\text{GFSA}_{\text{even}}$ on even layers for causal language modeling tasks

Method	#Params	PTB	WikiText-2	WikiText-103	**Avg**
GPT2 [88]	117M	89.1s	195.7s	9638.4s	3307.8s
+ GFSA	117M	160.3s	354.2s	17424.6s	5979.7s
+ $\text{GFSA}_{\text{even}}$	117M	127.4s	279.1s	13761.4s	4722.6s

Table. 3-36: Comparison of using GFSA on all layers vs. $\text{GFSA}_{\text{even}}$ on even layers for ImageNet-1k

Method	Input Size	#Layers	#Params	Top-1 Acc	Runtime
DeiT-S	224	12	43M	79.8	551s
+ GFSA	224	12	43M	81.1	814s
+ $\text{GFSA}_{\text{even}}$	224	12	43M	81.0	595s

3.8.18 GFSA in Linear Transformers

Table 3-37 shows the accuracy, runtime, and GPU usage results on Long Range Arena benchmark using ListOps and Image datasets.

Table. 3-37: Comparison of accuracy (%), runtime (s per 1,000 steps) and GPU usage (GB) on Long Range Arena benchmark

Method	ListOps (2K)			Image (4K)		
	Accuracy	Runtime	GPU usage	Accuracy	Runtime	GPU usage
Transformer [64]	37.1	198.3	5.50	38.2	345.1	5.88
Transformer+GFSA	**37.6**	635.8	10.87	**40.2**	737.2	11.20
Linformer [178]	37.3	63.4	1.73	37.8	158.5	3.45
YOSO-E [179]	37.3	85.7	0.37	39.8	114.2	1.42
Effic. Attention [152]	36.9	49.2	0.57	40.2	121.1	1.14
Effic. Attention + GFSA	**37.9**	53.8	0.67	**40.4**	135.8	1.33

Chapter 4

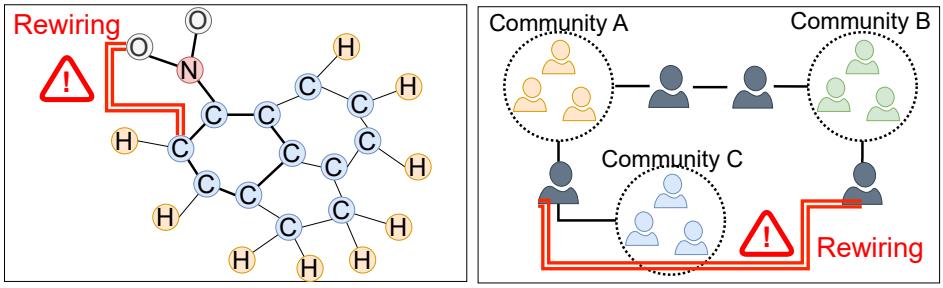
Over-squashing I

Recent research in the field of GNN has identified a critical issue known as "over-squashing," resulting from the bottleneck phenomenon in graph structures, which impedes the propagation of long-range information. Previous works have proposed a variety of graph rewiring concepts that aim to optimize the spatial or spectral properties of graphs to promote signal propagation. However, such approaches inevitably deteriorate the original graph topology, which may lead to distortion of the information flow. To address this, we introduce an ex**pand**ed width-**a**ware (**PANDA**) message passing, a new message passing paradigm where nodes with high centrality, a potential source of over-squashing, are selectively expanded in width to encapsulate the growing influx of signals from distant nodes. Experimental results show that our method outperforms existing rewiring methods, suggesting that selectively expanding the hidden state of nodes can be a compelling alternative to graph rewiring for addressing the over-squashing.

The materials in this chapter have been published in Choi et al. [180].

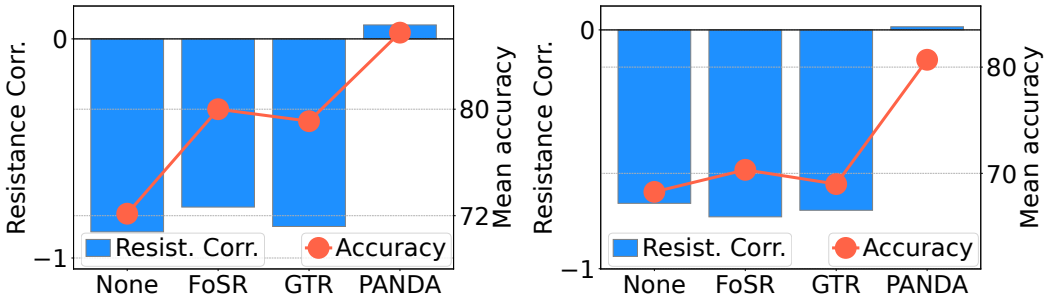
4.1 Motivation

Graph Neural Networks (GNNs) have emerged as a powerful tool for graph data processing. They are being studied extensively in various domains, as evidenced by significant research contributions [24, 23, 26, 27, 5, 34, 60, 181, 15, 182, 183]. A key subclass within GNNs is Message Passing Neural Networks (MPNNs), which excel in propagating information through neighbouring nodes. As MPNNs propagate only for a 1-hop distance within a single layer, long-range interactions can only be captured when the depth of the network is comparable to the distance. However, increasing the number of layers for capturing long-range dependencies results in exponential growth of the receptive field and excessive aggregations of repeated messages that are subsequently packed into a feature vector of fixed length, which causes the problem called "over-squashing" [10, 184]. This phenomenon severely decreases the expressivity of the networks, and thus impairs the model's performance. Previous studies



(a) Molecular graph (b) Social graph

Figure. 4-1: Potential pitfalls of rewiring in domain-specific graphs: (a) In a molecular graph, rewiring the edge in red to a benzene ring violates the domain knowledge. (b) In a social graph, connecting a user to his/her enemy may lead to a totally different meaning.



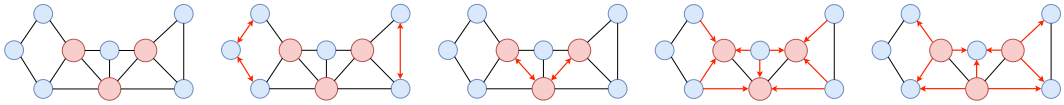
(a) Mutag (b) Reddit-Binary

Figure. 4-2: Comparison of resistance correlation and mean accuracy across different methods for GCN. A large negative correlation reflects that a higher total effective resistance is associated with reduced signal propagation (see Section 4.4).

have attempted to identify topological bottlenecks in a graph and directly modify graph connectivity with various optimization targets, including curvature, effective resistance, and spectral gap [10, 185, 186, 187, 188, 12, 189, 190, 191, 192, 193, 194].

Limitations of current rewiring methods. Existing rewiring methods that alter the graph topology to resolve over-squashing, while potentially beneficial, can inadvertently introduce inaccuracies within domain-specific contexts. Figure 4.1(a) demonstrates how modifying the molecular graph of a benzene ring could contradict chemical principles. In contrast, Figure 4.1(b) reveals that rewiring in social networks has the potential to distort the underlying community structures. These examples highlight the necessity to preserve the original graph structure to maintain the validity of domain-specific semantic information.

A number of studies have attempted to identify the various factors associated with over-



(a) Expanded nodes (b) Low-to-low (c) High-to-high (d) Low-to-high (e) High-to-low

Figure. 4-3: Examples of PANDA's message passing mechanism. The size of the node indicates the size of the hidden dimension.

squashing through the lens of sensitivity analysis of the Jacobian of node features [185, 14]. As one of these trials, Di Giovanni et al. [14], an insightful analytical paper, provides a theoretical justification that increasing the width of the model (i.e., hidden dimension) can also contribute to improving the sensitivity of the model. However, it points out a significant concern regarding expanding the width of the model to address over-squashing; increasing the hidden dimension globally affects the whole network at the expense of the generalization capacity of the model.

Inspired by the limitations of existing rewiring methods and the promising role of width expansion for addressing the over-squashing, our object is to design a novel message passing paradigm that mitigates the over-squashing by targeting the nodes that are mainly involved in creating bottlenecks in the signal flow and selectively expanding their width without risking a change in the underlying graph topology.

Main idea. It is possible to define graph bottlenecks as nodes with high centrality, such as betweenness centrality [195, 185]. Nodes with high centrality can show hub-like behavior in that they connect to a relatively larger set of nodes lying on different parts of the graph, thus receiving excessive information from them. Increasing the width of the feature vector of these nodes provides more space for them to process information flowing from other nodes. Then, we can take advantage of their enhanced capacities to alleviate over-squashing. This requires exchanging messages between nodes with different hidden dimensions. This raises a natural question:

"Is it possible to design a message passing that enables information exchange between nodes with different widths only in a way that mitigates over-squashing without rewiring?"

We propose a new message passing framework that addresses the above question. First, the entire node set is divided into two subsets, expanded and non-expanded nodes, which are colored in red (◯) and blue (◯), respectively in Figure 4-3. Then, we classify four different edge types depending on the expansion state of the nodes at the source (edge tail) and target (edge head) positions: low-to-low, high-to-high, low-to-high, and high-to-low. Each edge type requires a distinct message passing scheme.

The standard message passing between nodes with equal widths corresponds to the conventional message passing (see Figures 4.3(b) and 4.3(c)). For high-to-low message passing (see Figure 4.3(d)), a low-dimensional node selects and receives a subset of the information

sent by a high-dimensional node. The low-to-high message passing involves augmenting the hidden vectors of low-dimensional nodes to match the higher dimension size for propagation (see Figure 4.3(e)).

Our framework can potentially enhance the capacity of nodes from the perspective of signal propagation. As shown in Figure 4-2, our framework maintains constant signal propagation w.r.t. effective resistance and *simultaneously* improves the accuracy compared to the existing rewiring methods. We use the correlation coefficient to quantify how signal propagation decreases as effective resistance increases.

Contributions and outline. We introduce an ex**pand**ed width-**a**ware message passing (**PANDA**)[1], a novel paradigm for message passing with expanded widths for nodes that are potentially bottlenecks. Our contributions can be summarized as follows:

- In Section 4.3, we propose **PANDA** that enables the signal propagation across the nodes of different widths.
- In Section 4.4, we discuss how our **PANDA** can alleviate over-squashing in terms of various perspectives. Verifying a higher feature sensitivity compared to other models, we show **PANDA** can overcome the topological bottleneck, maintaining consistent signal propagation even under large effective resistance. Finally, we show that our method solves the limitation of existing rewiring methods, e.g., over-smoothing.
- In Section 4.5, we empirically demonstrate that our **PANDA** outperforms existing rewiring methods.

4.2 Preliminaries

We first examine the notation used in this paper. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we use $\mathcal{V}$ and $\mathcal{E}$ to denote its nodes and edges, respectively. The nodes are indexed by v and u such that $v, u \in \mathcal{V}$, and an edge connecting nodes v and u is denoted by $(v, u) \in \mathcal{E}$. The connectivity is encoded in the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ where n is the number of nodes. p denotes the width (hidden dimension size), while ℓ is the number of layers. The feature of node v at layer ℓ is written as $\mathbf{h}_v^{(\ell+1)}$.

[1] Our source code is available here: `https://github.com/jeongwhanchoi/panda`.

4.2.1 Message Passing Neural Networks

We consider the case where each node v has a feature $\mathbf{h}_v^{(0)} \in \mathbb{R}^p$. MPNNs iteratively update node representations using the following equations:

$$\mathbf{m}_v^{(\ell)} = \psi^{(\ell)}(\{\mathbf{h}_u^{(\ell)} : u \in \mathcal{N}(v)\}), \tag{4.1}$$

$$\mathbf{h}_v^{(\ell+1)} = \phi^{(\ell)}(\mathbf{h}_v^{(\ell)}, \mathbf{m}_v^{(\ell)}). \tag{4.2}$$

where $\mathbf{m}_v^{(\ell)}$ is the aggregated node feature of v's neighbourhood. The aggregation function $\psi^{(\ell)}$ and the update function $\phi^{(\ell)}$ are learnable, and their different definitions result in different architectures [22, 196, 23]. By default, all nodes have the same hidden dimension size. We will introduce a method that enables message passing among nodes with different hidden dimensions in Section 4.3.

4.2.2 Over-squashing Problem

Initially identified by Alon and Yahav [10], over-squashing has become a significant challenge in GNNs when dealing with long-range dependencies. It mainly occurs when the information aggregated from too many neighbours is squashed into a fixed-sized node feature vector, resulting in a substantial loss of information [184]. We will review the literature, including rewiring methods, to address this problem, in Section 4.6. We now describe the Jacobian sensitivity and effective resistance widely used to estimate the over-squashing. In Section 4.4, we justify our method in terms of the two metrics.

Sensitivity bound. Recent theoretical insights on GNNs have provided a formal understanding of over-squashing via the lens of the sensitivity analysis [185]. The sensitivity bound theorem, as proposed by [14, Theorm 3.2], posits that the impact of over-squashing can be quantified by examining the following 3 factors: i) the Lipschitz continuity of the non-linearity in the model, ii) the width of the model, and iii) the topology of a graph. The sensitivity of $\mathbf{h}_v^{(\ell)}$ to $\mathbf{h}_u^{(0)}$, assuming that there exists a ℓ-path between nodes v and u, is bounded by

$$\left\| \frac{\partial \mathbf{h}_v^{(\ell)}}{\partial \mathbf{h}_u^{(0)}} \right\|_1 \leq (\underbrace{zwp}_{\text{model}})^{\ell} \underbrace{(\mathbf{S}^{\ell})_{vu}}_{\text{topology}}. \tag{4.3}$$

The impact of the model to over-squashing is bounded by the Lipschitz constant z, the width p, and the maximum weight w, while the topology of an input graph affects the sensitivity by ℓ-th power of the graph shift matrix $\mathbf{S}$. Hereinafter, $\mathbf{S}$ can be any (normalized) adjacency matrix (with self-loops).

A small Jacobian norm indicates that the final representations of a node learned by a model are not much affected by the variations in its neighbouring inputs, implying that the network suffers from over-squashing. This condition typically arises when the information

has not been sufficiently propagated over the graph due to the excessive aggregation of information, i.e., compression into a small hidden vector, through several layers of message passing.

Effective resistance and signal propagation. Derived from the field of electrical engineering, the effective resistance between two nodes u and v in an electrical network is defined as the potential difference induced across the edges when a unit current is injected at one of the ends [197]. The effective resistance between the nodes u and v in the graph is given by

$$R_{u,v} = (\mathbf{1}_u - \mathbf{1}_v)^\top \mathbf{L}^+ (\mathbf{1}_u - \mathbf{1}_v), \tag{4.4}$$

where $\mathbf{1}_v$ and $\mathbf{1}_u$ are indicator vectors for node u and v, respectively. Rayleigh's monotonicity principle, which says that adding paths or shortening existing paths can only decrease the effective resistance between two nodes [198], leads to the following interpretation: more and shorter disjoint paths connecting the nodes u and v lead to a lower resistance between them [189, 199]. Therefore, edges with high effective resistance struggle to propagate information, creating bottlenecks. Total effective resistance, R_{tot}, is defined as the sum of effective resistance between all pairs of nodes [197, 189]:

$$R_{tot} = \sum_{u>v} R_{u,v} = n \cdot \mathrm{Tr}(\mathbf{L}^+) = n \sum_{i}^{n} \frac{1}{\lambda_i}, \tag{4.5}$$

where λ_i is the i-th eigenvalues of $\mathbf{L}$ and $\mathbf{L}^+$ is the pseudo-inverse of $\mathbf{L}$. R_{tot} is a key measure for measuring the overall degree of over-squashing in a graph. It is known that the signal propagation of GNNs is inversely proportional to R_{tot} [14], which will be covered in detail in Section 4.4 with a comparison to our method.

Over-smoothing problem. Over-smoothing is another well-known problem that reduces the expressiveness of GNNs in deeper layers, resulting in a loss of discriminative power in the representation of node features [45, 81, 8]. Rewiring methods to address the over-squashing problem focus on improving graph connectivity to ease the signal flow. However, one limitation of the rewiring method is that adding too many edges potentially leads to the over-smoothing issue [12]. To explore this trade-off between over-squashing and over-smoothing, FoSR [12] analyzes the change in Dirichlet energy upon adding edges. Their results demonstrate that rewiring the graph to enhance information propagation leads to the over-smoothing problem.

4.3 Proposed Method

We first introduce our motivation and design rationale. We then propose our PANDA message-passing framework.

4.3.1 Motivation and Design Rationale

Sensitivity bound. Our design rationale is anchored in the insight derived from the sensitivity bound theorem [14, Theorem 3.2]. Given the sensitivity bound in Equation (4.3), we consider two scenarios: one with a standard width p and the other with an increased width p_{high}, where $p_{\text{high}} > p$. For nodes with the increased width, the sensitivity bound is improved, demonstrating a potentially higher sensitivity to input features. Regarding this, we provide a Proposition 4.4.1 in Section 4.4 and show empirical results well aligned with our argument in Section 4.4. We do not increase the width of all nodes, but *selectively*.

Design rationale for selecting node centrality. We consider node centrality metrics to identify a node as a potential source of the bottleneck in a graph. Betweenness centrality has been considered as a significant indicator of the bottleneck, which measures the number of shortest paths going through a node [200]. The more frequently a node appears on the shortest paths of pairs of different nodes, the more likely it is to cause the bottleneck [185, Definition 9.]. However, a hub-like node with inter-community edges also has eccentric properties that differ from those of the high betweenness centrality node, such as a high degree. We apply various centrality metrics in our framework to understand how each is related to the over-squashing problem. Section 4.9.3 describes the centralities we consider.

4.3.2 Expanded Width-Aware Message Passing

Node expansion criteria. In our proposed method, we stratify nodes into two categories: low-dimensional nodes (○) and high-dimensional nodes (○) (see Figure 4-4). This stratification is based on a specific centrality that quantifies the importance of each node. The centrality, denoted as $C(\mathcal{G})$, can be one of the following: degree [201], betweenness [200], closeness [202], PageRank [203], or load centrality [204]. We represent the centrality values for all nodes as a vector $\mathbf{c} \in \mathbb{R}^n$ and then sort the elements of $\mathbf{c}$ in descending order. Based on the sorted $\mathbf{c}$, we select the top-k nodes, resulting in a binary mask vector, $\mathbf{b} \in \{0,1\}^n$:

$$\mathbf{b}_v = \begin{cases} 1 & \text{if } v \text{ is among the top } k \text{ nodes in } \mathbf{c}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

Define subsets of neighbourhoods. Based on a centrality measure, we define four subsets of neighbouring nodes. For a node v classified as low-dimensional, denoted by $\mathbf{b}_v = 0$:

$$\mathcal{N}_{\text{low}\leftrightarrow\text{low}}(v) = \{u \in \mathcal{N}(v) \mid \mathbf{b}_v = 0 \wedge \mathbf{b}_u = 0\}, \tag{4.7}$$

$$\mathcal{N}_{\text{high}\rightarrowtail\text{low}}(v) = \{u \in \mathcal{N}(v) \mid \mathbf{b}_v = 0 \wedge \mathbf{b}_u = 1\}. \tag{4.8}$$

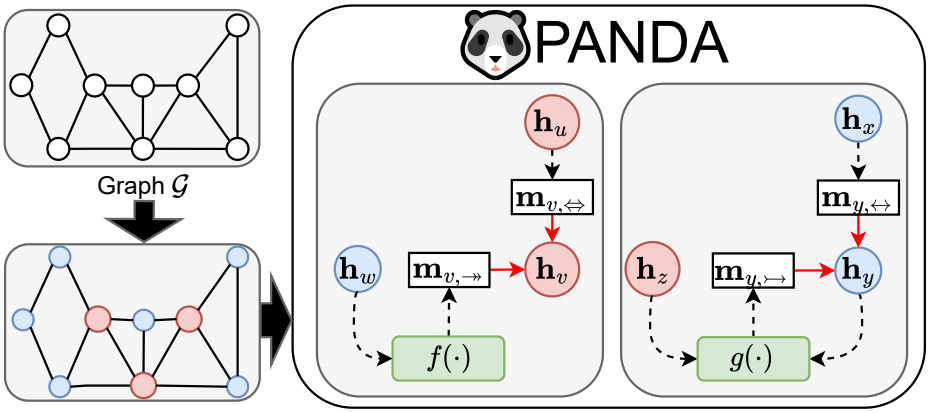


Figure. 4-4: Our proposed **PANDA** message passing framework. First, we selectively expand widths (i.e., hidden dimension sizes) according to a centrality in $\mathcal{G}$, and our **PANDA** message passing enables signal propagation among nodes with different widths (low-dimensional nodes and high-dimensional nodes).

In addition, for a node v classified as high-dimensional:

$$\mathcal{N}_{\text{high}\Leftrightarrow\text{high}}(v) = \{u \in \mathcal{N}(v) \mid \mathbf{b}_v = 1 \wedge \mathbf{b}_u = 1\}, \tag{4.9}$$

$$\mathcal{N}_{\text{low}\twoheadrightarrow\text{high}}(v) = \{u \in \mathcal{N}(v) \mid \mathbf{b}_v = 1 \wedge \mathbf{b}_u = 0\}. \tag{4.10}$$

The comprehensive union of these subsets across all nodes in $\mathcal{V}$ encompasses the entire edge set $\mathcal{E}$, thereby preserving the original graph structure.

A new framework: PANDA message passing. The main contribution of our framework is that, given any MPNNs, we can make them work with different widths for nodes. To this end, we replace the standard neighbour aggregator $\psi^{(\ell)}$ with 4 different aggregators, each applied to the corresponding type of neighbours: $\psi^{(\ell)}_{\leftrightarrow}$, $\psi^{(\ell)}_{\Leftrightarrow}$, $\psi^{(\ell)}_{\twoheadrightarrow}$, and $\psi^{(\ell)}_{\rightarrowtail}$. Then, we use a common update function for all types. The final message passed from layer ℓ to layer $\ell+1$ is defined as follows:

$$\mathbf{m}^{(\ell)}_{v,\leftrightarrow} = \psi^{(\ell)}_{\leftrightarrow}(\{\mathbf{h}^{(\ell)}_u : u \in \mathcal{N}_{\text{low}\leftrightarrow\text{low}}(v)\}), \tag{4.11}$$

$$\mathbf{m}^{(\ell)}_{v,\Leftrightarrow} = \psi^{(\ell)}_{\Leftrightarrow}(\{\mathbf{h}^{(\ell)}_u : u \in \mathcal{N}_{\text{high}\Leftrightarrow\text{high}}(v)\}), \tag{4.12}$$

$$\mathbf{m}^{(\ell)}_{v,\twoheadrightarrow} = \psi^{(\ell)}_{\twoheadrightarrow}(\{f(\mathbf{h}^{(\ell)}_u) : u \in \mathcal{N}_{\text{low}\twoheadrightarrow\text{high}}(v)\}), \tag{4.13}$$

$$\mathbf{m}^{(\ell)}_{v,\rightarrowtail} = \psi^{(\ell)}_{\rightarrowtail}(\{g(\mathbf{h}^{(\ell)}_v, \mathbf{h}^{(\ell)}_u) : u \in \mathcal{N}_{\text{high}\rightarrowtail\text{low}}(v)\}), \tag{4.14}$$

$$\mathbf{h}^{(\ell+1)}_v = \phi(\mathbf{h}^{(\ell)}_v, \mathbf{m}^{(\ell)}_{v,\leftrightarrow}, \mathbf{m}^{(\ell)}_{v,\Leftrightarrow}, \mathbf{m}^{(\ell)}_{v,\twoheadrightarrow}, \mathbf{m}^{(\ell)}_{v,\rightarrowtail}). \tag{4.15}$$

Each aggregator $\psi^{(\ell)}_*$ is tailored to handle a specific type of interaction among nodes. In Equation (4.15), the hidden vector of a node v is updated from the four messages. For

instance, $\psi_{\twoheadrightarrow}^{(\ell)}(\cdot)$ and $\psi_{\rightarrowtail}^{(\ell)}(\cdot)$ are designed for interactions with different widths by using functions $f(\cdot)$ and $g(\cdot)$.

$f(\cdot)$ is a linear transformation that projects node features to the expanded dimensional space and is defined as:

$$f(\mathbf{h}_u^{(\ell)}) := \mathbf{W}_f^{(\ell)}\mathbf{h}_u^{(\ell)}, \tag{4.16}$$

where $\mathbf{W}_f^{(\ell)} \in \mathbb{R}^{p_{\text{high}} \times p}$. For $\psi_{\rightarrowtail}^{(\ell)}(\cdot)$, we define $g(\cdot)$ as a dimension selector as follows:

$$g(\mathbf{h}_v^{(\ell)}, \mathbf{h}_u^{(\ell)}) := \text{gather}\big(\mathbf{h}_u^{(\ell)}, \text{topk}(\mathbf{s}, p)\big), \tag{4.17}$$

$$\mathbf{s} = \text{softmax}\left(\sigma\big(\eta(\mathbf{h}_u^{(\ell)} \oplus \mathbf{h}_v^{(\ell)})\big)\right), \tag{4.18}$$

where η is a non-linear neural network that computes a score vector $\mathbf{s} \in \mathbb{R}^{p_{\text{high}}}$ that gives the relative significance of each dimension for each message. This score vector is then used to select the top p dimensions for the message aggregation.

In our **PANDA**, it is important to emphasize that nodes with low and high-dimensional features are processed in their respective dimensional spaces throughout the network layers. This means that until the final layer, these nodes maintain their distinct dimensionalities.

Instance of our framework. To better understand our framework, we show how to integrate **PANDA** into two classical MPNNs: GCN [22] and GIN [196]. We will use these variations for our experiments. For the PANDA-GCN variant, the layer-update is expressed as follows for low and high-dimensional nodes. First, if the node v is low-dimensional:

$$\mathbf{h}_v^{(\ell+1)} = \mathbf{h}_v^{(\ell)} + \sigma\Big(\sum_{u \in \mathcal{N}_{\text{low}\leftrightarrow\text{low}}(v)} \frac{1}{\sqrt{d_u d_v}} \mathbf{W}_{\text{low}}^{(\ell)} \mathbf{h}_u^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{low}\twoheadrightarrow\text{high}}(v)} \frac{1}{\sqrt{d_u d_v}} \mathbf{W}_{\text{low}}^{(\ell)} g(\mathbf{h}_v^{(\ell)}, \mathbf{h}_u^{(\ell)})\Big), \tag{4.19}$$

then if node v is high-dimensional:

$$\mathbf{h}_v^{(\ell+1)} = \mathbf{h}_v^{(\ell)} + \sigma\Big(\sum_{u \in \mathcal{N}_{\text{high}\Leftrightarrow\text{high}}(v)} \frac{1}{\sqrt{d_u d_v}} \mathbf{W}_{\text{high}}^{(\ell)} \mathbf{h}_u^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{high}\rightarrowtail\text{low}}(v)} \frac{1}{\sqrt{d_u d_v}} \mathbf{W}_{\text{high}}^{(\ell)} f(\mathbf{h}_u^{(\ell)})\Big), \tag{4.20}$$

where σ is a ReLU activation function, $\mathbf{W}_{\text{low}}^{(\ell)} \in \mathbb{R}^{p \times p}$ and $\mathbf{W}_{\text{high}}^{(\ell)} \in \mathbb{R}^{p_{\text{high}} \times p_{\text{high}}}$ are the weight matrices. The messages are normalized by their degrees, d_v and d_u.

For the PANDA-GIN variant, we define the layer-update of PANDA-GIN. If the node v is low-dimensional:

$$\mathbf{h}_v^{(\ell+1)} = \text{MLP}_{\text{low}}^{(\ell)}\Big((1+\epsilon)\mathbf{h}_v^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{low}\leftrightarrow\text{low}}(v)} \mathbf{h}_u^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{high}\rightarrowtail\text{low}}(v)} g(\mathbf{h}_v^{(\ell)}, \mathbf{h}_u^{(\ell)})\Big), \tag{4.21}$$

where $\text{MLP}_{\text{low}}^{(\ell)}$ is one or more linear layers separated by ReLU activation and ϵ is a weight parameter. For the high-dimensional nodes, the layer-update is defined as follows:

$$\mathbf{h}_v^{(\ell+1)} = \text{MLP}_{\text{high}}^{(\ell)}\Big((1+\epsilon)\mathbf{h}_v^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{high}\Leftrightarrow\text{high}}(v)} \mathbf{h}_u^{(\ell)} + \sum_{u \in \mathcal{N}_{\text{low}\twoheadrightarrow\text{high}}(v)} f(\mathbf{h}_u^{(\ell)})\Big). \tag{4.22}$$

Implementation. For implementation, the hidden vectors of different widths are separated and entered into the layer. These low and high-dimensional nodes are output with the same dimension in the last layer and integrated again.

4.3.3 Properties of PANDA

Directivity. Nodes with an equal width remain undirected, while edges among nodes with different widths can be directed. $\psi_{\rightarrow}^{(\ell)}$ and $\psi_{\leftarrow}^{(\ell)}$ have independent sets of parameters — Equations (4.19) and (4.20) have different weight parameters for low-dimensional and high-dimensional nodes. By ensuring the directivity only for edges with different widths, our method can be more expressive than its undirected counterpart [205, Theorems 4.1 and 4.2].

Relational graph. Our **PANDA** can be considered as a relational graph convolutional network (R-GCN) [206] applied to an augmented relational graph that incorporates two types of relations. Not only is message propagation performed according to the relationship of the neighbour set, but different weight matrices are also used for low-dim nodes and high-dim nodes, as mentioned earlier. In this sense, our **PANDA** is similar to R-GCN. However, R-GCN is not a model designed to alleviate over-squashing.

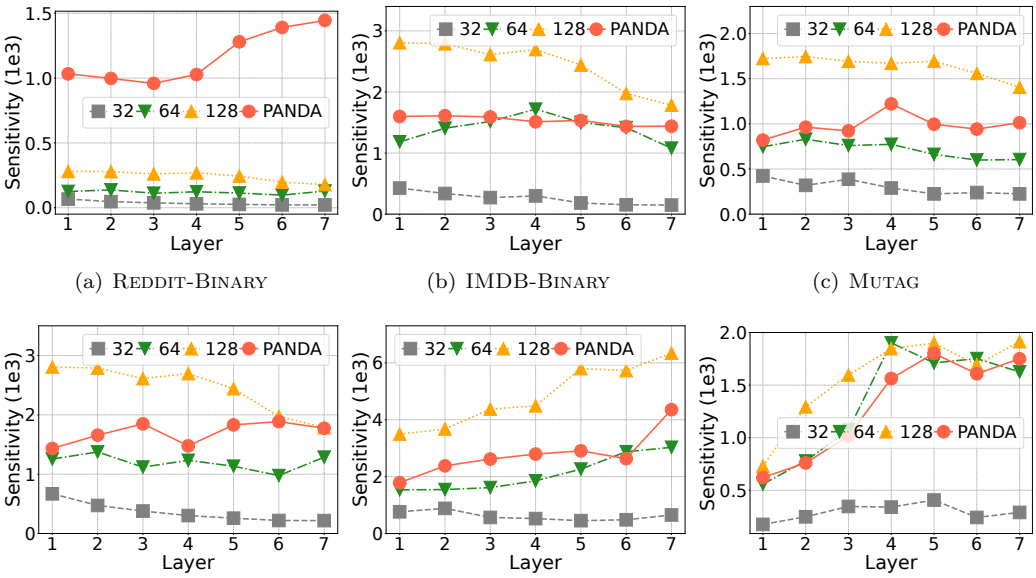
Computational complexity. Compared to existing rewiring methods, **PANDA** adds complexity from centrality calculation and 4 different message passings instead of graph rewiring. The specific complexity depends on the algorithm used to calculate centrality. For example, the time complexity of degree centrality is $\mathcal{O}(|\mathcal{E}|)$, and for betweenness centrality it is $\mathcal{O}(|\mathcal{V}|^3)$. We will discuss the empirical runtime of **PANDA** in Section 4.5.

4.4 Why Does PANDA Mitigate Over-squashing?

This section discusses why our **PANDA** alleviates over-squashing from empirical perspectives, i.e., feature sensitivity, signal propagation, and Dirichlet energy.

Empirical sensitivity analysis. To verify that our method improves the feature sensitivity among nodes, we analyze the sensitivity given by Equation (4.3) on benchmark datasets. As shown in Figure 4-5, the sensitivity improves as the width p increases. Our method benefits from selectively having 64 and 128-dimensional nodes, yielding sensitivity that lies between these two dimensions — 16.7% of nodes have 128 dimensions in Figure 4-5. As shown in Figure 4-6, compared to other rewiring techniques, **PANDA** shows higher sensitivity that is maintained even in deeper layers, while others exhibit decreasing trends.

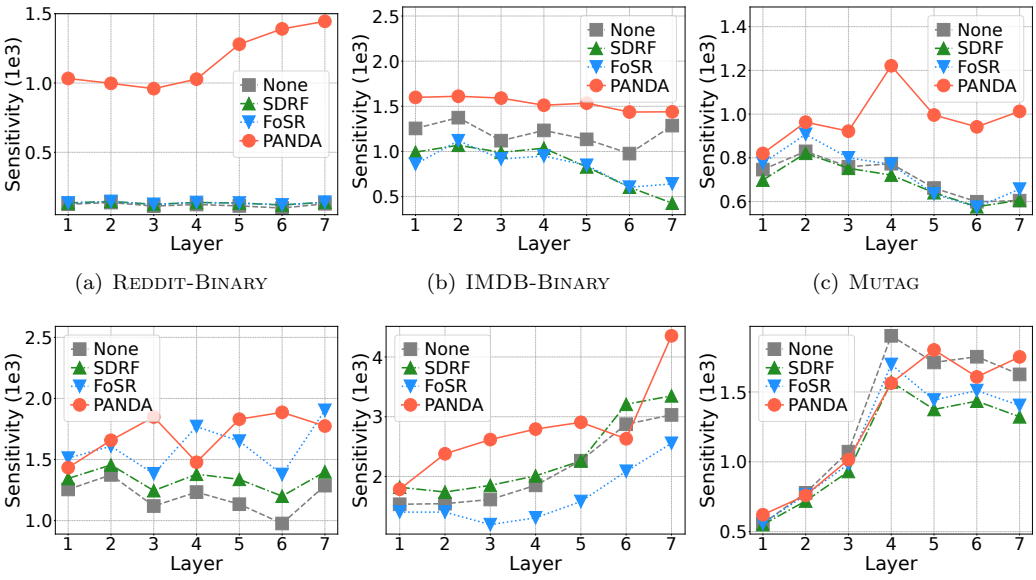
Theoretical sensitivity analysis. We provide a proposition based on Equation (4.3) that the introduction can alleviate over-squashing.



(a) Reddit-Binary (b) IMDB-Binary (c) Mutag

(d) Enzymes (e) Proteins (f) Collab

Figure. 4-5: Empirical sensitivity w.r.t p across layers for GCN on all datasets.



(a) Reddit-Binary (b) IMDB-Binary (c) Mutag

(d) Enzymes (e) Proteins (f) Collab

Figure. 4-6: Empirical sensitivity w.r.t methods across layers for GCN on all datasets.

Proposition 4.4.1 (Informal)**.** *Given the sensitivity bound in Equation* (4.3)*, we consider two scenarios: one with a standard width* p *and the other with an increased width* p_{high}*, where* $p_{\text{high}} > p$*. For nodes with increased width, the sensitivity bound is augmented, demonstrating a potentially higher sensitivity to input features:*

$$\left\| \frac{\partial \mathbf{h}_v^{(\ell)}}{\partial \mathbf{h}_u^{(0)}} \right\|_1 \leq (zwp)^\ell \left(\mathbf{S}^\ell\right)_{vu} < (zwp_{\text{high}})^\ell \left(\mathbf{S}^\ell\right)_{vu}. \tag{4.23}$$

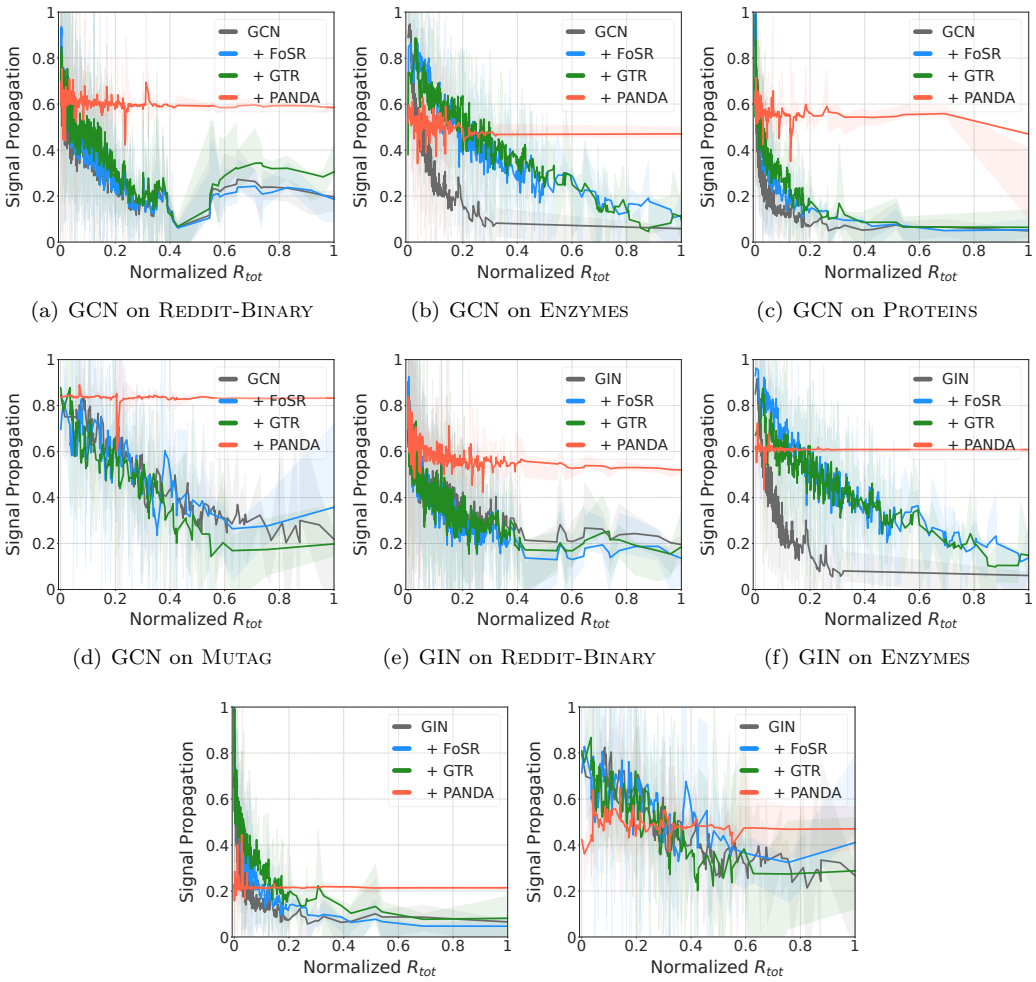
This increased sensitivity can potentially reduce over-squashing, facilitating better signal propagation.

Proposition 4.4.1 provides the theoretical foundation for our approach. We are inspired by the insights gained from increasing the upper bound with increased width, but we emphasize that we do not increase the width of all nodes, but rather *selectively*.

Signal propagation w.r.t. effective resistance. Di Giovanni et al. [14] provide an empirical evidence that the information propagation of general GNNs is inversely proportional to the total effective resistance R_{tot}, which motivates us to check if our **PANDA** maintains the magnitude of signal flow in a graph with high R_{tot}. We further analyze whether the signal flow improves after applying various rewiring methods. The detailed setting on the analysis is in Section 4.9.1.

In Figure 4-7, all methods except **PANDA** present decaying signal propagation trends as the resistance of graphs increases. In contrast, **PANDA** shows consistent signal propagation even for graphs with higher effective resistance. To verify the linear relationship between total effective resistance and signal propagation, we also quantify the correlation coefficient between them in Figure 4-2. This result demonstrates the powerful effect of the width expansion in mitigating the over-squashing problem. Our **PANDA** maintains continuous information flows even under high bottleneck conditions, which cannot be overcome even by the direct modification of graph topology through various rewiring strategies.

Trade-off between rewiring and over-smoothing. As mentioned in Section 4.2.2, graph rewiring can cause over-smoothing. Since **PANDA** does not modify the original graph connectivity, we analyze whether our method can avoid the over-smoothing problem induced by graph rewiring using the Dirichlet energy as a measure of over-smoothing. The theoretical relation between Dirichlet energy and over-smoothing is given in Section 4.9.2. Figure 4-8 compares the Dirichlet energies of the final hidden representations learned from GCN and GIN, having both a rewiring technique and **PANDA**. The result shows that **PANDA** yields a higher Dirichlet energy for both GCN and GIN compared to others, highlighting the strength of our model, mitigating the over-squashing while preserving the original graph connectivity.



(a) GCN on Reddit-Binary (b) GCN on Enzymes (c) GCN on Proteins

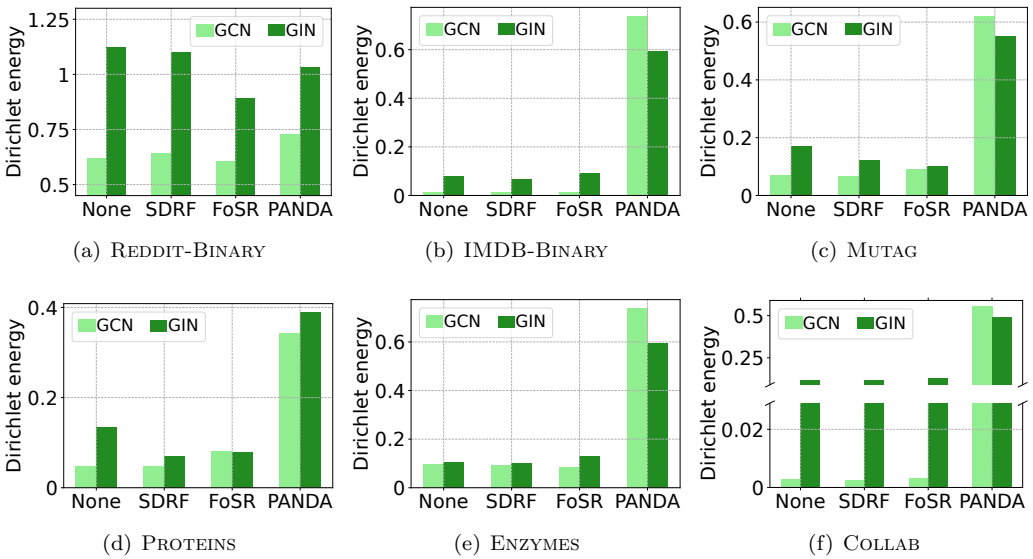
(d) GCN on Mutag (e) GIN on Reddit-Binary (f) GIN on Enzymes

(g) GIN on Proteins (h) GIN on Mutag

Figure. 4-7: The amount of signal propagated across the graph w.r.t. the normalized total effective resistance (R_{tot}) for all datasets.

4.5 Experiments

In this section, we empirically verify that **PANDA** can significantly improve the performance of GNN over other rewiring methods. We experiment with graph classification and node classification, as well as tasks on Long-Range Graph Benchmark (LRGB) [139]. We cover the experiments on TUDataset [207] in the main content and the node classification task and LRGB dataset in Section 4.9.5 and Section 4.9.6.



(a) Reddit-Binary (b) IMDB-Binary (c) Mutag

(d) Proteins (e) Enzymes (f) Collab

Figure. 4-8: Dirichlet energy of baselines and PANDA.

Datasets. For graph classification, we consider the Reddit-Binary (2,000 graphs), IMDB-BINARY (1,000 graphs), Mutag (188 graphs), Enzymes (600 graphs), Proteins (1,113 graphs), and Collab (5.000 graphs) tasks from TUDatasets [207]. These datasets were chosen by Karhadkar et al. [12], under the claim that they require long-range interactions.

Experimental setting. For graph classification, we compare **PANDA** to no graph rewiring and 7 other state-of-the-art rewiring methods:

- DIGL: Diffusion Improves Graph Learning (DIGL) [11] is a diffusion-based rewiring scheme that computes a kernel evaluation of the adjacency matrix, followed by sparsification.
- SDRF: Stochastic Discrete Ricci Flow (SDRF) [185] surgically rewires a graph by adding edges to support other edges with low curvature, which are locations of bottlenecks. For SDRF, we include results for both the original method (configured to only add edges), and our relational method (again only add edges and include the added edges with their own relation).
- FA: Fully adjacent (FA) layers [10] rewire the graph by adding all possible edges. We include results for rewiring only the last layer (last layer FA) and every layer (every layer FA).

- FOSR: First-order Spectral Rewiring (FoSR) [12] is a graph rewiring method for preventing over-squashing based on iterative first-order maximization of the spectral gap.
- BORF: Batch Ollivier-Ricci Flow (BORF) [12] is a novel curvature-based rewiring method designed to mitigate the over-smoothing and over-squashing issues simultaneously.
- GTR: Greedy Total Resistance (GTR) [189] is a rewiring method based on total effective resistance.
- CT-Layer. Commute Time (CT)-Layer [188] is a method that learns the commute time and rewires the input graph accordingly. In case of CT-Layer, we isolated the CT-Layer that consists of CT pooling and CT rewiring steps from the DiffWire [188], where the input graph is rewired and re-weighted based on the learnable commute time embeddings. Then we subsequently add a series of GCN and GIN layers identical to the ones used in other rewiring methods and **PANDA**.

In our experiments, we prioritize fairness and comprehensiveness rather than aiming to obtain the best possible performance for each dataset. For backbone GNNs, we use GCN, GIN [196], R-GCN [206], and R-GIN [208]. To faithfully compare the performance of the rewiring techniques, we follow the same setting as in Karhadkar et al. [12]. Each configuration is evaluated using the validation set. The testset accuracy of the configuration with the best validation performance is then recorded. For each experiment, we accumulate the result in 100 random trials with an 80%/10%/10% train/val/test split and report the mean test accuracy, along with the 95% confidence interval. Further experiment details are in Section 4.9.4.

Performance comparison results of general GNNs and PANDA. Table 4-1 shows the results of different methods applied to the GCN and GIN models across benchmark datasets. Our **PANDA** method shows the highest accuracy across most of the datasets for both models, significantly outperforming the baseline and other rewiring methods. Surprisingly, for Reddit-Binary, our **PANDA** shows a substantial 22.30% improvement over the baseline method. In particular, in the case of BORF, it is impractical to evaluate for Reddit-Binary and Collab due to the rewiring time. In the case of GIN, our **PANDA** also leads to the highest accuracies, with 91.055 on Reddit-Binary and 88.75 on Mutag. These results underscore the effectiveness of our **PANDA** in enhancing the performance of GNN.

Performance comparison results of relational GNNs and PANDA. Considering the similarities between PANDA-GCN and R-GCN discussed in the previous section, we empirically evaluate their performance. Table 4-2 shows the experimental results for R-GCN and R-GIN. Our **PANDA** method shows the highest accuracy on most datasets for both R-GCN and R-GIN, significantly outperforming the baseline and other rewiring methods. For

Table. 4-1: Results of PANDA and baselines for GCN and GIN. We show the best three in red (first), blue (second), and purple (third).

Method	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN	68.255±1.098	49.770±0.817	72.150±2.442	27.667±1.164	70.982±0.737	33.784±0.488
+ Last-FA	68.485±0.945	48.980±0.945	70.050±2.027	26.467±1.204	71.018±0.963	33.320±0.435
+ Every-FA	48.490±1.044	48.170±0.801	70.450±1.960	18.333±1.038	60.036±0.925	51.798±0.419
+ DIGL	49.980±0.680	49.910±0.841	71.350±2.391	27.517±1.053	70.607±0.731	15.530±0.294
+ SDRF	68.620±0.851	49.400±0.904	71.050±1.872	28.367±1.174	70.920±0.792	33.448±0.472
+ FoSR	70.330±0.727	49.660±0.864	80.000±1.574	25.067±0.994	73.420±0.811	33.836±0.584
+ BORF	Time-out	50.100±0.900	75.800±1.900	24.700±1.000	71.000±0.800	Time-out
+ GTR	68.990±0.610	49.920±0.990	79.100±1.860	27.520±0.990	72.590±2.480	33.050±0.400
+ CT-Layer	51.580±1.019	50.320±0.944	75.899±3.024	17.383±1.030	60.357±1.060	52.146±0.415
+ **PANDA**	80.690±0.721	63.760±1.012	85.750±1.396	31.550±1.230	76.000±0.774	68.400±0.452
GIN	86.785±1.056	70.180±0.992	77.700±0.360	33.800±0.115	70.804±0.827	72.992±0.384
+ Last-FA	90.220±0.475	70.910±0.788	83.450±1.742	47.400±1.387	72.304±0.666	75.056±0.406
+ Every-FA	50.360±0.684	49.160±0.870	72.550±3.016	28.383±1.052	70.375±0.910	32.984±0.390
+ DIGL	76.035±0.774	64.390±0.907	79.700±2.150	35.717±1.198	70.759±0.774	54.504±0.410
+ SDRF	86.440±0.590	69.720±1.152	78.400±2.803	35.817±1.094	69.813±0.792	72.958±0.419
+ FoSR	87.350±0.598	71.210±0.919	78.400±2.803	29.200±1.367	75.107±0.817	73.278±0.416
+ BORF	Time-out	71.300±1.500	80.800±2.500	35.500±1.200	74.200±0.800	Time-out
+ GTR	86.980±0.660	71.280±0.860	77.600±2.840	30.570±1.420	73.130±0.690	72.930±0.420
+ CT-Layer	54.589±1.757	50.000±0.974	56.850±4.253	16.583±0.907	61.107±1.184	52.304±0.605
+ **PANDA**	91.055±0.402	72.560±0.917	88.750±1.570	46.200±1.410	75.759±0.856	75.110±0.210

example, in the RGIN model, **PANDA** is shown to be the most effective method, achieving an accuracy of 91.36 on REDDIT-BINARY and 88.2 on MUTAG. Surprisingly, for ENZYMES, **PANDA** shows a significant 36.09% improvement over the baseline method. In R-GCN, GTR shows higher performance than **PANDA**. However, overall, **PANDA** shows high effectiveness in R-GCN and R-GIN.

Effect of centrality metrics. It is natural to ask how different the performance of **PANDA** varies depending on the centrality metrics. Table 4-3 compares the results obtained using different kinds of centrality metrics. In IMDB-BINARY, PageRank centrality is better than other metrics, indicating that larger node widths effectively receive messages from influential nodes. Closeness centrality shows the best performance on PROTEINS and COLLAB. This indicates that it effectively expands the width of nodes that require many connections to connect to other distant nodes. As shown in Table 4-4, for PANDA-GIN, the PageRank centrality performs best on REDDIT-BINARY, IMDB-BINARY, and MUTAG. This shows that in social networks such as REDDIT-BINARY and IMDB-BINARY, the dimension of the node that receives links from highly influential nodes must be large, as is the characteristic of PageRank.

Table. 4-2: Results of our method and rewiring methods for R-GCN and R-GIN. We show the best three in red (first), blue (second), and purple (third).

Method	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
R-GCN	49.850±0.653	50.012±0.917	69.250±2.085	28.600±1.186	69.518±0.725	33.602±1.047
+ Last-FA	49.800±0.626	50.650±0.964	70.550±1.810	28.233±1.138	69.527±0.815	34.732±1.194
+ Every-FA	49.950±0.593	50.500±0.891	70.500±1.836	33.400±1.142	71.670±0.882	33.616±0.978
+ DIGL	49.995±0.619	49.670±0.843	73.400±2.007	28.283±1.213	68.232±0.851	19.926±1.441
+ SDRF	58.620±0.647	53.640±1.043	72.300±2.215	33.483±1.245	69.107±0.759	67.990±0.386
+ FoSR	76.590±0.531	64.050±1.123	84.450±1.517	35.633±1.151	73.795±0.692	70.650±0.482
+ GTR	80.180±0.600	65.090±0.930	85.500±1.470	41.330±1.280	75.780±0.760	74.340±0.410
+ **PANDA**	80.200±0.913	66.790±1.088	90.050±1.466	43.900±1.176	76.000±0.802	71.400±0.376
R-GIN	87.965±0.564	68.889±0.872	83.050±1.439	39.017±1.166	70.500±0.809	75.544±0.323
+ Last-FA	89.995±0.647	69.710±1.025	80.600±1.639	48.183±1.401	70.304±0.844	75.434±0.491
+ Every-FA	56.855±0.943	71.480±0.876	83.050±1.518	54.950±1.331	71.045±0.909	75.432±0.475
+ DIGL	74.425±0.723	63.930±0.947	81.450±1.488	37.600±1.198	71.312±0.757	54.714±0.416
+ SDRF	86.825±0.523	70.210±0.806	82.700±1.782	39.583±1.333	70.696±0.815	76.480±0.388
+ FoSR	89.665±0.416	71.810±0.880	86.150±1.492	45.550±1.258	74.670±0.692	76.480±0.390
+ GTR	90.410±0.410	71.490±0.930	86.100±1.762	50.030±1.320	75.640±0.740	77.450±0.390
+ **PANDA**	91.360±0.372	72.090±0.936	88.200±1.513	53.100±1.344	76.170±0.776	77.800±0.355

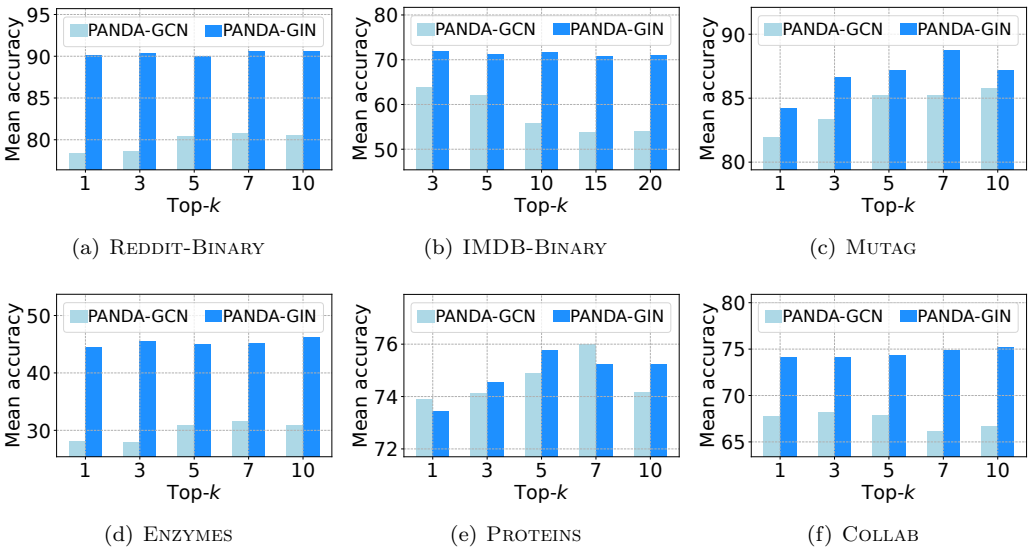
Table. 4-3: Performance comparison by various centrality measures for PANDA-GCN

$C(\mathcal{G})$	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
Degree	80.690±0.721	62.100±1.043	85.200±1.568	31.117±1.258	75.375±0.800	68.162±0.471
Between.	80.000±0.659	59.630±1.152	85.750±1.396	29.600±1.208	74.589±0.791	67.844±0.547
Closeness	79.700±0.664	61.160±0.992	84.700±1.554	29.967±1.231	76.000±0.774	68.400±0.452
PageRank	80.340±0.826	63.760±1.012	85.450±1.569	31.550±1.230	74.098±0.851	67.540±0.500
Load	79.500±0.732	59.840±1.153	85.700±1.549	28.167±1.090	74.188±0.814	67.802±0.506

Table. 4-4: Results of our method and rewiring methods for PANDA-GIN

$C(\mathcal{G})$	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
Degree	90.640±0.414	71.900±0.929	86.200±2.263	37.450±1.376	75.759±0.856	74.950±0.402
Between.	90.700±0.378	71.390±0.857	86.150±1.999	44.133±1.363	75.027±0.871	74.644±0.491
Closeness	90.080±0.434	71.310±0.963	88.400±1.612	46.200±1.410	74.741±0.845	74.956±0.412
PageRank	91.055±0.402	72.560±0.917	88.750±1.570	41.483±1.237	74.223±0.702	74.500±0.464
Load	90.625±0.417	71.900±0.967	86.050±1.774	44.150±1.455	74.812±0.878	75.110±0.210

Sensitivity on top-k nodes. Figure 4-9 shows a sensitivity study w.r.t. top-k nodes. For PROTEINS, both PANDA-GCN and PANDA-GIN show an increase in mean accuracy as the top-k value increases from 3 to 7. However, the accuracy does not increase after the k value is 7. For MUTAG, PANDA-GCN and PANDA-GIN have the highest accuracy for k values of 10 and 7, respectively.



(a) Reddit-Binary (b) IMDB-Binary (c) Mutag

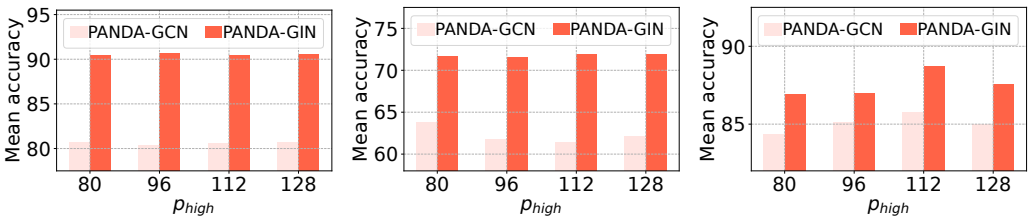
(d) Enzymes (e) Proteins (f) Collab

Figure. 4-9: Sensitivity on top-k for all datasets.

Sensitivity on p_{high}. Figure 4-10 shows a sensitivity study of the **PANDA** w.r.t. p_{high}. In Proteins, using PANDA-GCN and PANDA-GIN with larger p_{high} improves performance. In Mutag, on the other hand, both models achieve their best performance when using a p_{high} value of 112.

Empirical runtime analysis. We compare the runtime against the centrality metric used by PANDA and the rewiring method FoSR. As can be seen in Table 4-5, using centrality metrics for IMDB-Binary, Mutag, and Proteins is faster than the rewiring method FoSR. However, when the size of the graph such as Reddit-Binary or Collab is large, metrics such as betweenness, closeness, and load centrality that calculate the shortest path take a long time.

In Table 4-6, we also measure and compare the runtime of the forward pass of the original GCN and PANDA-GCN. **PANDA** takes more time than the original GCN due to the amount of computation used in Equations (4.16) and (4.17). However, in our design, it is meaningful to select the different dimensions of high-dimensional nodes that are learned to be more significant for each message (edge) in $g(\cdot)$ of Equation (4.17). In future work, we aim to design an efficient method for selecting the dimensions.



(a) Reddit-Binary (b) IMDB-Binary (c) Mutag

(d) Enzymes (e) Proteins (f) Collab

Figure. 4-10: Sensitivity on p_{high} for all datasets.

Table. 4-5: Empirical runtime comparison of rewiring methods and centrality metrics (in seconds)

	Method	Reddit-Binary	IMDB-Binary	Mutag	Enzymes	Proteins	Collab
	FoSR	11.97	3.36	3.43	3.28	3.47	6.55
PANDA	Degree	3.33	0.24	0.03	0.14	0.27	21.05
	Betweenness	965.31	1.16	0.06	1.20	5.58	349.68
	Closeness	238.32	0.38	0.43	0.40	1.53	55.60
	PageRank	9.65	1.50	0.58	1.09	1.93	43.56
	Load	953.12	1.03	0.11	1.21	5.17	331.34

Table. 4-6: Empirical runtime: PANDA-GCN forward pass duration (in milliseconds).

Method	Reddit-Binary	IMDB-Binary	Mutag	Enzymes	Proteins	Collab
GCN	11.5	10.6	7.32	19.2	12.1	38.5
+**PANDA**	41.3	33.9	27.3	68.4	45.5	98.7

4.6 Related Work

This section introduces related work to alleviate the over-squashing problem. Before the link between over-squashing and graph curvature was established by [185], DIGL was developed to sparsify the adjacency matrix using the graph heat kernel and personalized PageRank [11]. After Alon and Yahav [10] empirically observed the over-squashing problem, research is

active to find indicators for over-squashing and propose rewiring methods. Topping et al. [185] initially connect over-squashing with graph Ricci curvature, showing that edges with highly negative Ricci curvature contribute to the over-squashing and proposes an SDRF. Since then, rewiring methods with indicators based on curvatures have been studied [13, 193]. In addition to using curvatures as indicators of the over-squashing, Black et al. [189] measure the over-squashing through the concept of effective resistance. Banerjee et al. [186] propose rewiring methods to handle the original topology of the graph to prevent it from being disconnected [186]. To prevent over-smoothing, Karhadkar et al. [12] propose a rewiring method that optimizes the spectral gap. Deac et al. [187] propose to interleave message propagation on the original graph with message passing on an expander graph to alleviate over-squashing. Gutteridge et al. [209] propose a layer-dependent rewiring method that provides a variety of rewiring graphs for feature propagation. Barbero et al. [194] propose a LASER to rewire the graph to better preserve locality. Finkelshtein et al. [210] propose a CO-GNN with flexible and dynamic message passing that can perform effective graph rewiring at each layer of the GNN. Recently, Di Giovanni et al. [14] analyzed the factors contributing to over-squashing, and Southern et al. [211] analyzed the role of virtual nodes in over-squashing. Also, rewiring methods, as well as more advanced and flexible message-passing paradigms, are being studied [212, 194, 213, 214, 215, 216].

Most studies to mitigate over-squashing focus on rewiring methods that change the topology. However, PANDA is different in that it addresses the over-squashing problem by moving away from the rewiring method.

4.7 Limitations and Future Work

We only focus on redesigning the GNN based on sensitivity (Equation (4.3)) for the width of the nodes that contribute to over-squashing. Despite our empirical evidence for mitigating over-squashing, more research is needed to prove the effects of higher widths on over-squashing. For future work, we will aim to reduce the complexity of our method and explore a much wider range of tasks to study the pros and cons of higher widths.

4.8 Summary

In this chapter, we presented **PANDA**, a novel message passing framework, effectively addressing the over-squashing problem in GNNs without rewiring edges. By selectively expanding the width of high-centrality nodes, **PANDA** promotes signal propagation to alleviate over-squashing. Our empirical evaluations show that **PANDA** outperforms existing graph rewiring methods in graph classification. This research contributes a pioneering approach to selectively expanding node widths in message passing and lays the groundwork for the future exploration of width-aware strategies in GNNs.

4.9 Appendix

4.9.1 Signal Propagation

Signal propagation w.r.t. effective resistance. Here, we provide experimental details for measuring signal propagation with respect to the normalized total effective resistance of the graphs. First, fix a source node $v \in \mathcal{V}$ and assign p-dimensional feature vector to it, while assign the rest of the nodes zero vectors. Then, the amount of signal that has been propagated over the graph by the randomly initialized model with ℓ layers is given by

$$\mathbf{h}_{\odot}^{(\ell)} = \frac{1}{p \max_{u \neq v} k_{\mathcal{G}}(u,v)} \sum_{t=1}^{p} \sum_{u \neq v} \frac{\mathbf{h}_u^{(\ell),t}}{\|\mathbf{h}_u^{(\ell),t}\|} k_{\mathcal{G}}(u,v), \tag{4.24}$$

where $\mathbf{h}_u^{(\ell),t}$ is the t-th feature of p-dimensional feature vector of node u at layer ℓ and $k_{\mathcal{G}}(u,v)$ is the distance between two nodes u and v, computed as a shortest path. Every unitary signal $\mathbf{h}_u^{(\ell),t}/\|\mathbf{h}_u^{(\ell),t}\|$ propagated across the graph G from the source node v is weighted by the normalized propagation distance $k_{\mathcal{G}}(u,v)/\max_{u \neq v} d_G(u,v)$ for all nodes $u \neq v$ and then averaged over entire p output channels. 10 nodes are randomly sampled from each graph, and the total effective resistance of the graph is estimated for each source node. The final $\mathbf{h}_{\odot}^{(\ell)}$ and total resistance of the graph are calculated as the mean of all 10 nodes. The experiment is repeated for every graph in the dataset, and the signal propagation measured for each graph is plotted with respect to the normalized total effective resistance of the corresponding graph.

4.9.2 Dirichlet Energy and Over-smoothing

Let $\mathcal{G}$ be a graph with adjacency matrix $\mathbf{A}$ and normalized Laplacian $\widetilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Given a vector field $\mathbf{H} \in \mathbb{R}^{n \times p}$ defined on nodes in $\mathcal{G}$, Dirichlet energy is defined as

$$E_{\text{Dir}}(\mathbf{H}) := \text{Tr}(\mathbf{H}^{\mathsf{T}}\widetilde{\mathbf{L}}\mathbf{H}) = \frac{1}{2} \sum_{v,u,w} \mathbf{A}_{v,u} \left(\frac{\mathbf{H}_{v,u}}{\sqrt{d_v}} - \frac{\mathbf{H}_{u,w}}{\sqrt{d_u}} \right)^2 \tag{4.25}$$

Essentially, the Dirichlet energy quantifies how much a function on the graph deviates from being constant between connected node pairs, which thereby indicates how non-smooth the signals on a graph are [217, 188]. Hence, a metric of Dirichlet energy has been widely applied to measure the amount of over-smoothing of the graph representations.

4.9.3 Graph Centrality Metrics

We next introduce five graph centrality metrics:

- Degree centrality [201] measures the number of direct connections a node has to other nodes in the network, e.g., the degree of nodes. The centrality value of node v is the

fraction of nodes connected to it. This centrality measures a local characteristic and does not take into account global topology or neighbor connectivity.

- PageRank centrality [203] measures the importance of a node based on the importance of its neighbors. It is an iterative algorithm that assigns a node a high score if it is connected to many nodes that themselves have high scores. High PageRank centrality indicates that a node is not only connected to many nodes but also to nodes that are themselves central or important within the network. This centrality uses a damping factor to control the neighbours' effect on your node while measuring its importance. We set the damping factor as 0.85.

- Closeness centrality [202] is a metric that measures how close a particular node is to all other nodes within a network. This metric is calculated based on the shortest path length between nodes in a network and indicates how efficiently a node is connected to other nodes in the network on average. Closeness centrality of a node v is the reciprocal of the average shortest path distance to u over all $n-1$ reachable nodes. Nodes with high closeness centrality are located relatively close to all other nodes in the network. Because a node with high closeness centrality plays a central role through the shortest paths to many other nodes, the amount of information focused on this node can be very large.

- Betweenness centrality [200] measures the extent to which a node lies on the shortest paths between other nodes. It highlights nodes that serve as bridges or points of control within the network. A node with high betweenness centrality significantly influences the flow of information or resources within the network, as it can affect the transfer between other nodes by facilitating or constraining it. For this reason, in studies that aim to measure over-squashing [185], it is also used to measure the bottleneck of the graph. The centrality score of a node is calculated by the number of shortest paths that include the node. However, this metric is notorious for its high complexity in calculating all-pairs shortest paths.

- Load centrality [204] is similar to betweenness centrality but considers the number of times a node is traversed by all shortest paths in the network. It is a measure of the load or traffic that a node will handle. Nodes with high load centrality are critical for the flow of information or resources, as they are likely to be bottlenecks or critical points of failure within the network. Like other centralities, it is useful for finding nodes where bottlenecks occur.

4.9.4 Experimental Details for Graph Classification

Hyperparameters. For each task and baseline model, we used the same settings of GNN and optimization hyperparameters across all methods to rule out hyperparameter tuning as a

source of performance gain. Table 4-7 shows common hyperparameters. Table 4-8 shows the search range for hyperparameters of **PANDA**, and Table 4-9 shows the best hyperparameters used by **PANDA**.

Table. 4-7: Common hyperparameters

Common Hyperparameters	
Dropoout	0.5
Number of layers	4
Hidden dimension	64
Learning rate	0.001
Stopping patience	100 epochs

Table. 4-8: Hyperparameter search space of PANDA for benchmark datasets

Hyperparameters	Search Space
p_{high}	{80, 96, 112, 128}
top k	{1, 3 ,5, 7, 10, 15, 20}
Centrality	{Degree, Betweenness, Closeness, PageRank, Load }

Hardware specifications and libraries. The following software and hardware environments were used for all experiments: Ubuntu 18.04 LTS, Python 3.7.13, PyTorch 1.11.0, PyTorch Geometric 2.0.4, Numpy 1.21.6, NetworkX 2.6.3, CUDA 11.3, and NVIDIA Driver 465.19, and i9 CPU, and NVIDIA RTX 3090. We implemented our **PANDA** message passing framework in PyTorch Geometric.

4.9.5 Experiments on Node Classification

Datasets. For node classification, we use the following datasets: Cora, Citeseer [76], Texas, Cornell, Wisconsin [46], and Squirrel [73].

Experimental setting. Even if we allow different hidden sizes for the nodes, the output dimension of the last layer is the same for all nodes. Thus, it can also be applied to the task of node classification. For node classification, we compare **PANDA** to no graph rewiring and 4 other state-of-the-art rewiring methods: DIGL [11], SDRF [185], FoSR [12], and BORF [12]. We use the same GCN settings across all methods to rule out hyperparameter tuning as a source of performance improvement. We accumulate results over 100 randomized trials, the same as Nguyen et al. [13]'s setup, and report the average test accuracy with 95% confidence intervals. We use the hyperparameters reported in Table 4-10 as the best hyperparameters for **PANDA**.

Table. 4-9: Best hyperparameter

Hyperparameter	Method	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
Top-k	GCN	7	3	10	7	7	3
	GIN	10	3	10	10	5	10
	R-GCN	5	5	5	10	10	3
	R-GIN	15	5	5	15	5	10
p_{high}	GCN	80	80	112	128	128	128
	GIN	96	128	96	128	112	96
	R-GCN	128	128	128	96	112	128
	R-GIN	96	128	128	112	128	128
$C(\mathcal{G})$	GCN	Degree	PageRank	Betweenness	PageRank	Closeness	Degree
	GIN	PageRank	PageRank	PageRank	Closeness	Degree	Load
	R-GCN	Degree	PageRank	PageRank	Betweenness	Betweenness	Degree
	R-GIN	Degree	Degree	PageRank	Betweenness	Betweenness	Betweenness

Table. 4-10: Best hyperparameter for node classification

Hyperparam.	Texas	Cornell	Wisconsin	Squirrel	Cora	Citeseer
Top-k	50	50	50	25	50	50
p_{high}	96	96	96	96	96	96
$C(\mathcal{G})$	Betweenness	Betweenness	Betweenness	Degree	Betweenness	Betweenness

Results. As shown in Table 4-11, in almost all cases, **PANDA** shows higher accuracy than existing rewiring methods. In cases like Cornell and Cora, BORF still performs the best, but it is comparable to **PANDA**. **PANDA** effectively improves the classification accuracy, especially in heterophilic graphs, such as Texas, Squirrel, and Wisconsin. In the case of Wisconsin, it achieves higher average accuracy than both DIGL and BORF, with a 12.57% improvement over DIGL.

Table. 4-11: Node classification accuracies of GCN with None, DIGL, SDRF, FoSR, BORF rewiring, and **PANDA** on various node classification datasets. We show the best three in red (first), blue (second), and purple (third).

Method	Texas	Cornell	Wisconsin	Squirrel	Cora	Citeseer
GCN	44.2±1.5	41.5±1.8	44.6±1.4	42.5±2.7	86.7±0.3	72.3±0.3
+ DIGL	53.6±1.5	44.5±1.2	51.7±1.3	35.2±3.0	83.2±0.2	74.5±0.3
+ SDRF	43.9±1.6	42.2±1.5	46.2±1.2	42.4±2.5	86.3±0.3	72.6±0.3
+ FoSR	46.0±1.6	40.2±1.6	48.3±1.3	42.5±1.9	85.9±0.3	72.3±0.4
+ BORF	49.4±1.2	50.8±1.1	50.3±0.9	42.6±2.8	87.5±0.2	73.8±0.2
+ **PANDA**	55.4±1.6	47.5±1.4	58.2±1.4	43.1±0.9	87.0±0.3	75.3±0.3

4.9.6 Experiments on Long Range Graph Benchmark

Datasets. We use the Peptides (15,535 graphs) dataset from the Long Range Graph Benchmark (LRGB) [139]. There are two tasks related to peptides: i) Peptides-func, a peptide feature classification task, and ii) Peptides-struct, a peptide structure regression task.

Experimental setting. We compare **PANDA** to no graph rewiring and 4 other state-of-the-art rewiring methods: DIGL [11], SDRF [185], FoSR [12], and BORF [12]. On the Peptide-func dataset, we use two experimental setups: the same experimental setup as Nguyen et al. [13] and the commonly used experimental setup of Dwivedi et al. [139] for a fair comparison between our method and the rewiring method. The setting for Nguyen et al. [13] uses a lower hidden dimension of 64 instead of 300, while the setting for Dwivedi et al. [139] uses a hidden dimension of 300. In Peptide-struct dataset, we use the same

experimental settings as those of Dwivedi et al. [139] after fixing the number of layers to 5. In particular, no additional positional features are used to confirm the effectiveness of the rewiring method and the effectiveness of **PANDA**. We only replace the last layer of the backbone GCN with **PANDA** message passing for performance and efficiency.

Results. In Table 4-12, high average precision (AP) and low mean absolute error (MAE) values are preferred. On the PEPTIDE-FUNC dataset, for the setting of Nguyen et al. [13], FoSR and BORF slightly improve the average precision over GCN, but **PANDA** outperforms them. In the case of Dwivedi et al. [139]'s setting, existing rewiring methods show trivial improvement compared to GCN, while **PANDA** improves by 1.65%. Additionally, in PEPTIDES-STRUCT, FoSR achieves a performance improvement of 0.658%, but **PANDA** achieves a performance improvement of 6.407%. This shows that the **PANDA** message passing is effective even on LRGB datasets.

Table. 4-12: Results of GCN with None, SDRF, FoSR, BORF, and PANDA on PEPTIDES-FUNC and PEPTIDES-STRUCT. We show the best three in red (first), blue (second), and purple (third).

Method	PEPTIDES-FUNC (AP ↑)		PEPTIDES-STRUCT (MAE ↓)
	Nguyen et al. [13]'s setup	Dwivedi et al. [139]'s setup	Dwivedi et al. [139]'s setup
GCN	44.2±1.5	59.30±0.23	0.3496±0.0013
+ SDRF	43.9±1.6	59.47±1.26	0.3478±0.0013
+ FoSR	46.0±1.6	59.47±0.35	0.3473±0.0007
+ BORF	45.6±1.5	59.94±0.37	0.3514±0.0009
+ **PANDA**	55.4±1.6	60.28±0.31	0.3272±0.0001

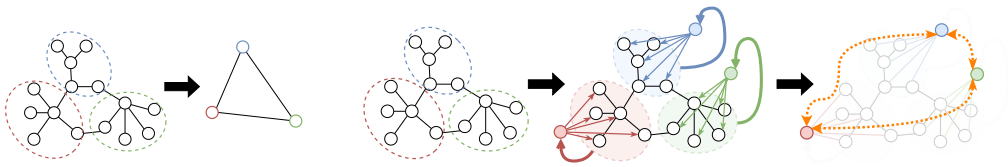
Chapter 5

Over-squashing II

Graph Neural Networks (GNNs) have emerged as powerful tools for learning on graph-structured data, but often struggle to balance local and global information. While graph Transformers aim to address this by enabling long-range interactions, they often overlook the inherent locality and efficiency of Message Passing Neural Networks (MPNNs). We propose a new concept called '*fractal nodes*', inspired by the fractal structure observed in real-world networks. Our approach is based on the intuition that graph partitioning naturally induces a fractal structure, where subgraphs often reflect the connectivity patterns of the full graph. Fractal nodes are designed to coexist with the original nodes and adaptively aggregate subgraph-level feature representations, thus enforcing feature similarity within each subgraph. We show that fractal nodes alleviate the over-squashing problem by providing direct shortcut connections that enable long-range propagation of subgraph-level representations. Experiment results show that our method improves the expressive power of MPNNs and achieves a performance comparable to or better than that of the graph Transformers, while maintaining the computational efficiency of MPNN by improving the long-range dependencies of MPNN.

5.1 Motivation

GNNs have become powerful tools for learning on graph-structured data, in various domains such as social network analysis, molecular property prediction, and recommendation systems [24, 23, 26, 34]. At the core of this field are MPNN [17], which propagates information by iteratively exchanging messages between neighboring nodes. However, MPNNs face limitations, particularly over-smoothing [81] and over-squashing [10]. To address these challenges, Transformer architectures [64] have been adapted for graph learning, applying self-attention mechanisms to enable long-range interactions by treating all nodes as tokens [218, 219, 220]. While graph Transformers effectively capture global context, they often overlook the inherent locality of MPNNs [221]. Although approaches such as GraphGPS [222] attempt to combine MPNN and Transformer node representations to balance local and global information, the



(a) Coarsened (renormalized) graph (b) Graph where our fractal nodes are connected.

Figure. 5-1: Conceptual comparison of renormalization and our fractal node process. (a) In renormalization, the original graph is replaced by a single node according to each box-covering method, resulting in a coarsened network. (b) After partitioning the original graph into subgraphs, we aggregate the low and high-frequency information of each subgraph to create *fractal nodes* (🔴, 🔵, 🟢). Then, we propagate the information to the original nodes (our proposed FN). We also support the long-distance interactions (orange dashed lines) between fractal nodes (our proposed FN_M).

computational complexity of Transformers remains a challenge.

Limitations of MPNN and graph Transformers. The limitations of both MPNN and graph Transformers motivate us to seek a novel approach that balances local and global information while preserving scalability. Our inspiration comes from the concept of fractals [223], which describes systems where similar patterns recur on scales. Many real-world graphs show this fractal structure, where structural patterns repeat across different parts of the graph [224, 225, 226]. Fractal structures in graphs are commonly studied through renormalization [227] (see Figure 5.1(a)), where groups of nodes are coarsened into "super-nodes" to analyze global properties emerging from local structures. In contrast to such coarsening, one may consider partitioning the graph into subgraphs and learning representative features for each group — without changing the original graph topology. This alternative perspective leads us to the following question:

Can fractal-inspired subgraph representations enhance message passing by improving long-range information flow, while preserving the locality and efficiency of MPNNs?

Our answer is "yes", and we introduce our main idea: fractal nodes for enforcing feature similarity.

Main idea: graph partitioning causes *fractal structure*, and each *fractal node* enforces feature similarity in each partition. We propose a novel concept called '*fractal nodes*', inspired by the fractal structure observed in real-world networks. We build on the view that graph partitioning naturally causes fractal structure, where subgraphs reflect the connectivity patterns of the full graph. Rather than replacing groups of nodes with super nodes as in renormalization, we retain the original graph and introduce fractal nodes (🔴, 🔵, 🟢) that coexist with the original nodes (see Figure 5.1(b)). Each fractal node serves as a representative for its subgraph and is connected to all nodes in its subgraph. These nodes

aggregate subgraph-level information and enforce *feature similarity* among the nodes within the same subgraph. To construct these representations, each fractal node adaptively combines both low-frequency (global) and high-frequency (local) components of the features in its subgraph. This goes beyond simple mean pooling, which only retains the lowest-frequency signal. These one-hop shortcut connections between fractal nodes and their subgraph nodes effectively reduce the information propagation distance. This theoretical reduction in effective resistance between nodes empirically supports the mitigation of over-squashing [10] and improves signal propagation. Furthermore, we apply an MLP-Mixer [228] at the final layer to flexibly mix the representations of the fractal nodes. This allows long-range interactions between subgraphs without relying on multi-hop message passing, which leads to signal degradation in standard MPNNs.

Contributions. We propose a novel paradigm, *fractal nodes*[1], which improves message passing in GNNs by enforcing feature similarity within subgraphs, by enforcing feature similarity within subgraphs, using the fractal structure caused by graph partitioning. Our main contributions are as follows:

- **A new architectural module for MPNNs.** We introduce *fractal nodes*, a plug-in component that captures subgraph-level information, and integrate them into MPNNs (Section 5.3). Our approach is inspired by the fractal nature of networks and discusses the properties of *fractal nodes* (Section 5.4).
- **Theoretical and empirical analysis.** We analyze the role of fractal nodes in reducing effective resistance and demonstrate that they mitigate the over-squashing problem (Section 5.5.1). We also show improved expressive power compared to standard MPNNs (Section 5.5.2).
- **Strong empirical performance with efficiency.** Extensive experiments on various benchmark datasets show that MPNNs augmented with fractal nodes achieve performance comparable to or better than state-of-the-art graph Transformer-based models (Section 5.5.3), while improving the scalability and maintaining efficiency of standard MPNNs (Section 5.5.4).

5.2 Background & Related Work

In this section, we discuss MPNNs, their limitations, graph Transformers, augmented MPNNs, and fractality and self-similarity in networks.

[1] Our source code is available here: `https://sites.google.com/view/fractalnode/`

Message passing neural network. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we use $\mathcal{V}$ and $\mathcal{E}$ to denote its nodes and edges, respectively. The nodes are indexed by v and u such that $v, u \in \mathcal{V}$, and an edge connecting nodes v and u is denoted by $(v, u) \in \mathcal{E}$. We consider the case where each node v has a hidden vector $h_v^{(\ell)} \in \mathbb{R}^d$, where d is the size of the hidden dimension, and ℓ is the number of layers. MPNNs iteratively update node representations:

$$h_v^{(\ell+1)} = \varphi(h_v^{(\ell)}, \psi^{(\ell)}(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\})), \tag{5.1}$$

where $\psi^{(\ell)}$ and $\varphi^{(\ell)}$ are aggregation function and update function.

Limitations of MPNNs. In several studies, MPNNs have been investigated for their *expressive power* limitations and *over-squashing* problems. Simple MPNNs are known to be only as powerful as the 1-Weisfeler-Leman graph isomorphism test [196]. The over-squashing problem occurs when MPNNs struggle to propagate information along long paths, resulting in loss of information when aggregating from too many neighbors into a fixed-sized node feature vector [10, 14]. This issue arises because local information spreading along the network is insufficient to fully capture the local and global context. This leads to the development of graph Transformers, which use self-attention to connect "everything is connected to everything", thereby solving the over-squashing problem.

Graph Transformers. Because of the success of Transformers, previous works have adapted this architecture for graph learning [218, 229]. Dwivedi and Bresson [218] proposed the use of graph Laplacian eigenvectors as node positional encodings. GraphGPS [222] provides a general framework combining MPNNs with Transformer components, while Graphormer[143] uses attention mechanisms to estimate several types of encoding, such as centrality, spatial, and edge encodings. Wu et al. [219] applies the MPNN directly to all nodes and then applies a Transformer, which is computationally intensive. He et al. [144] generalize ViT [89] to graphs and Ma et al. [230] show that adding inductive biases to graph Transformers removes the need for MPNN modules in GraphGPS. Exphormer [231] improves GraphGPS by using self-attention on expander graphs. While graph Transformers effectively capture long-range dependencies and mitigate over-squashing problems [10, 14], they typically scale quadratically with the number of nodes $\mathcal{O}(|\mathcal{N}|^2)$ compared to the linear scaling of MPNNs $\mathcal{O}(|\mathcal{E}|)$, motivating our work to develop more efficient architectures that preserve global information capture.

Augmented MPNNs. To improve information flow and address the limitations of standard MPNNs, various strategies have been proposed [14, 184, 180]. One approach is to incorporate additional global graph features during representation learning [17, 141]. Another effective method is rewiring the input graph to enhance connectivity and alleviate structural bottlenecks [11, 189, 12, 13]. These adjustments allow for more effective information flow within the network. Another example of graph augmentation is a virtual node, which adds a

new node to the graph to enhance information exchange between all pairs of nodes. This heuristic, introduced by Gilmer et al. [17], has been observed to improve performance on various tasks. Further analyses by Hwang et al. [232] and Cai et al. [233] have explored the role of virtual nodes in mitigating under-reaching and over-smoothing issues.

Fractality and self-similarity in networks. The concept of fractals, introduced by Mandelbrot [223], describes systems that exhibit self-similarity, where similar patterns recur across different scales. This concept has been widely applied in network science, where many real-world networks have been shown to exhibit fractal structures and scale-free properties [227, 234, 235]. For instance, social networks, web networks, and even protein interaction networks have been found to have fractal properties [226]. A common approach to analyzing fractality in networks involves coarsening techniques such as renormalization or box-covering techniques [234], which group nodes into coarser units and study the coarsened structures. However, these methods are not directly applicable in graph learning settings, where node coordinates or geometric information is often unavailable. These observations suggest the potential to incorporate fractality into graph learning through scalable and learnable mechanisms. Instead of relying on explicit structural coarsening methods, box-covering, and renormalization, we explore how self-similar patterns can be captured and utilized directly within the message passing process.

5.3 Fractal-Inspired Message Passing with Fractal Nodes

In this section, we propose fractal nodes and explain how they contribute to overcoming the limitations of existing MPNNs. We describe how to create fractal nodes, how to augment the MPNNs, and how to implement interactions between intra- and inter-subgraphs.

Notation. Let $\{\mathcal{V}_1, \ldots, \mathcal{V}_C\}$ be the set of node subsets corresponding to C subgraphs, where C is the number of subgraphs. $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ is the induced subgraph of $\mathcal{G}$. We define $h_{v,c}^{(\ell)}$ as the hidden vector of node v of the c-th subgraph in layer ℓ, and $f_c^{(\ell)}$ as the hidden vector of the fractal node of the c-th subgraph in the ℓ-th layer.

Message passing with *fractal nodes*. We first introduce the message passing process, including fractal nodes. The message passing process proceeds as follows:

$$\widetilde{h}_{v,c}^{(\ell+1)} = \varphi^{(\ell)}(h_{v,c}^{(\ell)}, \psi^{(\ell)}(h_{u,c}^{(\ell)} : u \in \mathcal{N}_v)), \tag{5.2}$$

$$f_c^{(\ell+1)} = \varphi_{\mathsf{FN}}^{(\ell)}(f_c^{(\ell)}, \psi_{\mathsf{FN}}^{(\ell)}(\widetilde{h}_{u,c}^{(\ell+1)} : u \in \mathcal{N}_v)), \tag{5.3}$$

$$h_{v,c}^{(\ell+1)} = \widetilde{\varphi}^{(\ell)}(\widetilde{h}_{v,c}^{(\ell+1)}, f_c^{(\ell+1)}), \tag{5.4}$$

where $\mathcal{N}(v)$ is the set of neighbors of node v. Equation (5.2) performs standard message passing at the node level. Equation (5.3) updates the fractal node representations. It aggregates

hidden vectors from all nodes in the subgraph, $\mathcal{V}_c$, using the $\widetilde{h}_{u,c}^{(\ell+1)}$, and then updates the fractal node representation. $\psi_{\mathsf{FN}}^{(\ell)}$ and $\varphi_{\mathsf{FN}}^{(\ell)}$ are aggregate and update functions for fractal nodes, which will be explained in more detail. The update function $\widetilde{\varphi}^{(\ell)}$ is the step that shows that the message $f_c^{(\ell+1)}$ is propagated to $h_{v,c}^{(\ell)}$.

How to create *fractal nodes*. As shown in Figure 5.1(b), fractal nodes are created from partitioned subgraphs. To partition into subgraphs, we consider the METIS [236] algorithm for its scalability and efficiency. How we use METIS is provided in Section 5.8.3. In our design, each fractal node plays a dual role: (i) structurally, it originates from a subgraph that reflects the recurring connectivity patterns of the original graph, and (ii) functionally, it encodes a subgraph-level feature representation that enforces *feature similarity* among the nodes within the same subgraph. While graph partitioning naturally preserves structural characteristics, our method focuses on learning subgraph-level representations by adaptively combining low-frequency (global) and high-frequency (local) components of node features within each subgraph. To motivate this, we first show that simple mean pooling captures only the direct current (DC) component – i.e., the lowest frequency – of the signal.

Theorem 5.3.1 (Mean pooling as a low-pass filter capturing the DC component)**.** *Let h_v represent the hidden state of node v in subgraph $\mathcal{V}_c$ and let $H_c = [h_1, h_2, \ldots, h_n] \in \mathbb{R}^{n\times d}$ be the matrix of node features for all nodes in $\mathcal{V}_c$ where $n = |\mathcal{V}_c|$ is the number of nodes in the subgraph. The mean pooling operation applied to the node features is equivalent to extracting the DC or the lowest frequency component of the signal in the frequency domain.*

As shown in Theorem 5.3.1, mean pooling corresponds to extracting the lowest frequency component — also known as the DC component — in the Fourier domain. This DC component captures the global characteristic of the subgraph, but it ignores higher-frequency variations that represent local details. A formal proof of Theorem 5.3.1 is provided in Section 5.8.1.

While Theorem 5.3.1 shows that mean pooling only captures the DC component, fractal nodes go beyond this limitation by using LPF and HPF. We adaptively rescale the high-frequency component, and combine LPF and HPF together to form fractal nodes:

$$f_c^{(\ell+1)} = \mathsf{LPF}(h_{v,c}^{(\ell+1)}) + \omega_c^{(\ell)}\mathsf{HPF}(h_{v,c}^{(\ell+1)}), \tag{5.5}$$

where $\omega_c^{(\ell)}$ is a learnable parameter controlling the contribution of high-frequency components. We use a learnable scalar, $\omega_c^{(\ell)} \in \mathbb{R}^1$, or a learnable vector parameter, $\boldsymbol{\omega}_c^{(\ell)} \in \mathbb{R}^d$. The LPF is computed by averaging the node features within the subgraph, so it can capture global information:

$$\mathsf{LPF}(h_{v,c}^{(\ell+1)}) = \frac{1}{|\mathcal{V}_c|}\sum_{v\in\mathcal{V}_c} h_{v,c}^{(\ell+1)}. \tag{5.6}$$

Equation (5.6) is analogous to mean pooling and represents the global, low-frequency component of the subgraph. To capture the local details, the HPF is applied by subtracting

the low-pass filtered output from the original node hidden vector. This allows the model to retain the local variations:

$$\mathsf{HPF}(h_{v,c}^{(\ell+1)}) = h_{v,c}^{(\ell+1)} - \mathsf{LPF}(h_{v,c}^{(\ell+1)}). \tag{5.7}$$

Fractal nodes mixing with MLP-Mixer. We also allow fractal nodes to exchange messages, as the coarsened network in Figure 5.1(a) takes advantage of long-distance interactions. To do this, we can apply the MLP-Mixer layer [228] to the fractal nodes in the last layer. This means that we do not need to create a coarsened network, and the MLP-Mixer flexibly mixes the representations of fractal nodes:

$$\widetilde{F} = \mathsf{MLPMixer}(F^{(L)}),\ F^{(L)} = [f_1^{(L)}, \ldots, f_C^{(L)}], \tag{5.8}$$

where $F^{(L)}$ is the matrix of all fractal node representations at final layer L. The MLP-Mixer layer consists of token-mixing and channel-mixing steps:

$$U = F^{(L)} + (W_2\rho(W_1\mathsf{LayerNorm}(F^{(L)}))) \tag{5.9}$$

$$\widetilde{F}^{(L)} = U + (W_4\rho(W_3\mathsf{LayerNorm}(U^T)^T), \tag{5.10}$$

where ρ is a GELU nonlinearity, $\mathsf{LayerNorm}(\cdot)$ is layer normalization, and matrices $W_1 \in \mathbb{R}^{d_1\times C}, W_2 \in \mathbb{R}^{C\times d_1}, W_3 \in \mathbb{R}^{d_2\times d}, W_4 \in \mathbb{R}^{d\times d_2}$ are learnable weight matrices, where d_1 and d_2 are the tunable hidden widths in the token-mixing and channel-mixing MLPs.

Instance of our framework. To better understand our framework, we show how to integrate fractal nodes into GCN [22], one of the general MPNNs. The update equation for GCN + FN is the following:

$$\begin{aligned}
\widetilde{h}_{v,c}^{(\ell+1)} &= \sigma\Big(h_{v,c}^{(\ell)} + \sum_{u\in N(v)} \frac{1}{\sqrt{\deg_v \deg_u}} h_{u,c}^{(\ell)} W^{(\ell)}\Big), \\
f_c^{(\ell+1)} &= \mathsf{LPF}(\widetilde{h}_{v,c}^{(\ell+1)}) + \omega_c^{(\ell)} \cdot \mathsf{HPF}(\widetilde{h}_{v,c}^{(\ell+1)}), \\
h_{v,c}^{(\ell+1)} &= \widetilde{h}_{v,c}^{(\ell+1)} + f_c^{(\ell+1)},
\end{aligned} \tag{5.11}$$

where σ a ReLU activation function, and $\deg_v$ and $\deg_u$ are their node degrees. Due to space constraints, the update equations of other MPNNs and implementation details are in Section 5.8.3. The method of applying the fractal nodes as in Equation (5.11) is called FN, and the method of using the fractal nodes of the last layer by mixing (see Equation (5.8)) is called FN_M from now on.

The output layer. Once the final representation h_G is derived, we use a multi-layer perceptron (MLP) as an output layer to predict graph-level outputs:

$$y_G = \mathsf{MLP}(h_G), \quad h_G = \mathsf{MeanPool}(H^{(L)} \text{ for } \mathsf{FN},\ \widetilde{F}^{(L)} \text{ for } \mathsf{FN}_M) \in \mathbb{R}^d,$$

where y_G is either a scalar for regression tasks or a vector for classification tasks, and $H^{(L)} = [h_1^{(L)}, ..., h_{|V|}^{(L)}]$ is the matrix of node representations at the final layer L for all nodes in the graph. For details on our method for node classification tasks, please refer to Section 5.8.7.1.

5.4 Properties and Effectiveness of Fractal Nodes

In this section, we analyze why fractal nodes are effective and what properties they have, discuss the model complexity, and compare them with previous work.

5.4.1 Why Fractal Nodes Improve Message Passing

Theoretical analyses. We provide theoretical analyses showing that fractal nodes help mitigate over-squashing by reducing the effective resistance between nodes.

Theorem 5.4.1 (Resistance reduction)**.** *Let $\mathcal{G}$ be the original graph and $\mathcal{G}_f$ be the augmented graph with fractal nodes. For any nodes $u, v \in \mathcal{G}$, the effective resistance in $\mathcal{G}_f$ satisfies:*

$$R_f(u, v) \leq R(u, v),$$

where $R_f(u, v)$ is the effective resistance in $\mathcal{G}_f$ and $R(u, v)$ is the original effective resistance in $\mathcal{G}$.

This reduction in effective resistance improves signal propagation between distant nodes as follows Theorem 5.4.2.

Theorem 5.4.2 (Signal propagation with fractal nodes)**.** *For an MPNN with fractal nodes, the signal propagation between nodes u, v after ℓ layers satisfies:*

$$\|h_u^{(\ell)} - h_v^{(\ell)}\| \leq \exp(-\ell / R_f(u, v)) \|h_u^{(0)} - h_v^{(0)}\|,$$

where $R_f(u, v)$ is the effective resistance in $\mathcal{G}_f$.

Since $R_f(u, v) \leq R(u, v)$, fractal nodes improve the worst-case signal propagation bound compared to the original graph. *The proofs and detailed analyses can be found in* Sections 5.8.2.2 and 5.8.2.3.

Frequency response analysis. We analyze fractal nodes from a frequency perspective. Figure 5-2 shows normalized frequency responses. Mean pooling shows a minimal response in a high-frequency domain, which suggests an oversimplification of node representations by losing local details. GCN and self-attention have higher responses than mean pooling in the high-frequency domain, but are still lower than fractal nodes. The improved high-frequency response of fractal nodes preserves fine local details.

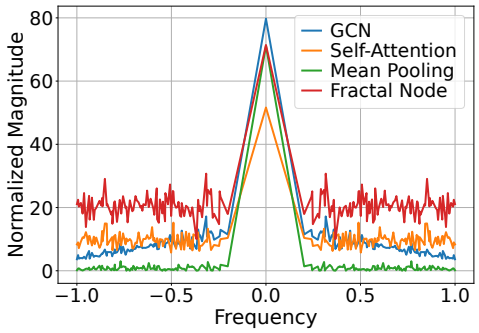
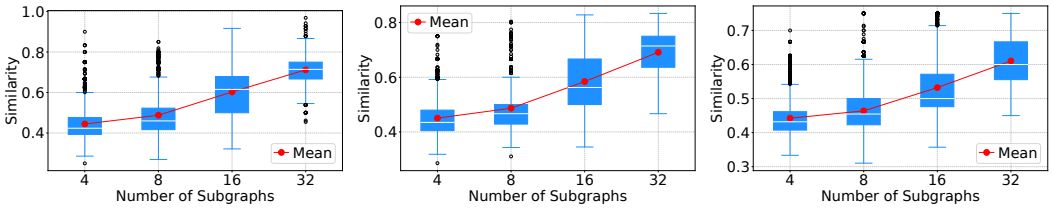


Figure. 5-2: Normalized frequency response on PEPTIDES-STRUCT.

Fractal structure and feature similarity. Fractal structure emerges naturally from the graph partitioning process. Since each subgraph is a subset of the original graph, it often reflects the overall connectivity patterns of the full graph. This structural similarity aligns with the concept of fractality, where similar patterns recur across different scales. To quantify this, we compare the betweenness centrality distributions between the original graph and its subgraphs. Betweenness centrality [200] captures both local and global structural importance, particularly in graphs where even low-degree nodes can serve as critical bridges [237]. As shown in Figure 5-3, the structural similarity increases with the number of subgraphs (see Section 5.8.4 for details). On top of this property, our fractal nodes are designed to enforce *feature similarity* within each subgraph. Each fractal node summarizes the features of its corresponding subgraph by adaptively combining low-frequency (global) and high-frequency (local) components using a learnable parameter $\omega_c^{(\ell)}$. While mean pooling only retains global information through DC components, our approach preserves global patterns and local variations in the feature space.



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

Figure. 5-3: Similarity of node centrality distribution in PEPTIDE-STRUCT.

Expressive power of *fractal nodes*. The expressive power of fractal nodes can be understood through the lens of existing theoretical results on subgraph-based approaches. The methods have been shown to increase expressive power beyond MPNNs. Encoding local

subgraphs is stronger than 1-WL and 2-WL tests [238, Theorem 4.3]. In the context of the subgraph WL (SWL) test [239], fractal nodes achieve expressive power comparable to SWL with additional single-point aggregation and potentially approach SWL with additional global aggregation [239, Theorem 4.4], as the fractal nodes implicitly perform a form of global aggregation within each subgraph. We will empirically verify expressive power in Section 5.5.2.

5.4.2 Model Complexity

Our fractal nodes show improvements in computational efficiency compared to graph Transformers [218, 222]. The time complexity of our FN is $\mathcal{O}(L(|\mathcal{V}| + |\mathcal{E}|))$, where L is the number of layers, $|\mathcal{V}|$ is the number of nodes, and $|\mathcal{E}|$ is the number of edges. The FN_M introduces an additional mixing step through the MLP-Mixer, leading to a time complexity of $\mathcal{O}(L(|\mathcal{V}| + |\mathcal{E}|) + Cd^2)$. Given that C is smaller than $|\mathcal{V}|$, this term does not dominate the overall complexity, preserving the efficiency of the model. In contrast, graph Transformers incur a time complexity of $\mathcal{O}(L(|\mathcal{V}|^2))$, due to the quadratic cost of computing self-attention over all node pairs. Similarly, GraphGPS combines MPNNs with self-attention, resulting in comparable quadratic complexity $\mathcal{O}(L(|\mathcal{V}|^2))$. Thus, fractal nodes offer a computational advantage over graph Transformer-based methods.

5.4.3 Comparison with Prior Work

Subgraphs in graph learning. Several works introduce hierarchical clustering and coarsening for learning on graphs [240]. Chiang et al. [241] use graph clustering to identify well-connected subgraphs on large graphs. HC-GNN [242] shows competitive performance in node classification on large-scale graphs, using hierarchical community structures for message passing. In graph Transformers, several hierarchical models [238, 243, 244, 144] attempt to manage computational complexity, though they still face challenges with scalability as all nodes remain within the computational burden of the Transformer architecture. However, our approach, the incorporation of fractal nodes to MPNN, can reduce this computational cost while preserving structural information.

Comparison to graph coarsening methods. Coarformer [245] tries to use coarsened and original graphs as separate views, where the coarsened graph is input to the Transformer, while ANS-GT [246] feeds a sequence of node representations to the graph Transformer by combining original, global, and coarsened node representations formed via adaptive sampling. Our method, on the other hand, incorporates fractal nodes representing subgraph information into the MPNN and enables fractal nodes to exchange messages with the original nodes and exchange information between fractal nodes via MLP-Mixer.

Comparison to virtual node. If we do not split into subgraphs, there will be only one fractal node. This can be compared to a virtual node [17, 232, 233], which is known to have the information of a global node. While both approaches facilitate global or subgraph-level information exchange, the key difference lies in how they process information. Virtual nodes aggregate global information from the entire graph, whereas fractal nodes operate at a subgraph level. A virtual node has its own update and aggregation functions that process messages from all graph nodes, while regular nodes incorporate both their local neighborhood messages and the virtual node's message. In contrast, our fractal nodes adaptively decompose and process both low and high-frequency components of subgraph features. This allows fractal nodes to capture richer information at the subgraph level compared to virtual node implementations that typically aggregate global information.

5.5 Experiments

To evaluate the effectiveness of our proposed fractal nodes, we conduct extensive experiments on various tasks. We aim to answer the following key questions:

- **(Q1)** Can fractal nodes mitigate over-squashing in MPNNs?
- **(Q2)** Do fractal nodes improve expressiveness of MPNNs?
- **(Q3)** How do fractal nodes compare to MPNNs and other graph Transformers in terms of performance on benchmark datasets?
- **(Q4)** Does our method lead to a faster run time than graph Transformers?

In this experiment, we aim to verify if fractal nodes provide meaningful benefits. Afterwards, we conduct a series of additional studies, including ablation studies and sensitivity analyses.

5.5.1 Analysis on Over-squashing (Q1)

Signal propagation and effective resistance. The signal propagation of MPNNs is inversely proportional to the total effective resistance R_{tot} [14]. Theorems 5.4.1 and 5.4.2 motivates us to check if adding fractal nodes helps maintain signal flow across a graph with high R_{tot}[2]. The theoretical details are provided in Section 5.8.10. The results in Section 5.5.1 validate our theoretical predictions – GCN+FN mitigates the decay of signal propagation with higher R_{tot} compared to GCN. GCN fails to maintain the magnitude of signal flow under a severe bottleneck structure, indicated as higher total effective resistance. In contrast, GCN+FN_M shows resilience to over-squashing and maintains higher levels of signal propagation even under the highest R_{tot}. This improvement can be attributed to fractal nodes, which

[2] R_{tot} is the total effective resistance between all pairs of nodes.

serve as single-hop shortcuts to connect all nodes and enable efficient long-range interactions by exchanging features between them through MLP-Mixer.

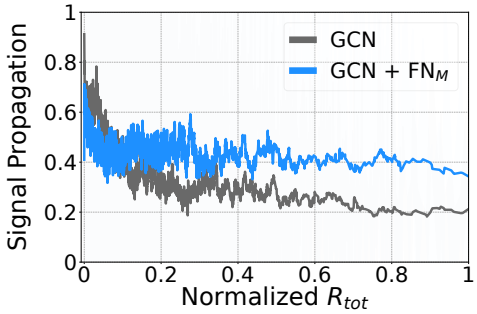


Figure. 5-4: The amount of signal propagated across the graphs w.r.t. the normalized R_{tot} in PEPTIDES-FUNC.

Fractal nodes alleviate over-squashing. We evaluate our fractal nodes on the TREENEIGHBOURSMATCH proposed by Alon and Yahav [10], which has tree structures that show fractal-like properties. The dataset helps evaluate over-squashing. In this dataset, each example consists of a binary tree of depth r, with the task of predicting the label for the target node by matching its degree of neighbors with a leaf node. As shown in Figure 5-5, standard MPNNs (i.e., GCN, GINE, GatedGCN) fail to generalize for $r > 4$, while our fractal nodes mitigate over-squashing and generalize up to $r = 7$. We empirically show that MPNNs augmented with fractal nodes can directly propagate long-distance information, avoiding the over-squashing problem.

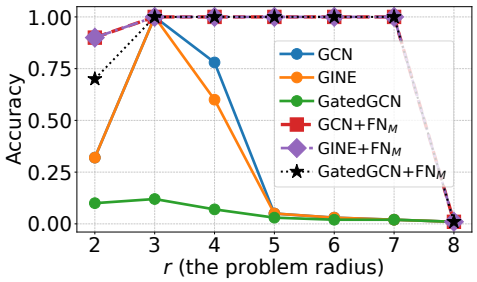


Figure. 5-5: Test accuracy across problem radius (tree depth) in the TREENEIGHBOURMATCH problem.

5.5.2 Expressive Power of Fractal Nodes (Q2)

We evaluate the expressive power of fractal nodes on 3 simulated datasets: CSL [247], EXP [248], and SR25 [249]. Each dataset contains graphs that are indistinguishable by the 1

to 3-WL test, and detailed descriptions are provided in Section 5.8.6.1. Figure 5-6 shows that our model achieves superior accuracy on all 3 datasets while MPNNs fail (see detailed result in Section 5.8.5). Our results are empirical, but align with our discussion in Section 5.4.1.

Figure. 5-6: Results on 3 synthetic datasets (Accuracy ↑).

Method	CSL	SR25	EXP
GCN	10.00	6.67	52.17
GCN + FN_M	39.67	100.0	86.40
GINE	10.00	6.67	51.35
GINE + FN_M	47.33	100.0	95.58
GatedGCN	10.00	6.67	51.25
GatedGCN + FN_M	49.67	100.0	96.50

5.5.3 Experiments on Graph Benchmarks (Q3)

Experimental setting and baselines. We evaluate our method on two different types of tasks: graph-level prediction and large-scale node classification. For graph-level tasks, we use 6 benchmark datasets: 2 peptide datasets from LRGB [139], 2 graph-level super-pixel image datasets from Benchmarking GNNs [140], and 2 molecular datasets from OGB [141]. For large-scale node classification, we use 2 large datasets from OGB [142]: ogbn-arxiv and ogbn-products. We compare our fractal nodes to MPNNs, graph Transformer-based models, and other state-of-the-art models. Detailed experimental settings for graph-level tasks are provided in Section 5.8.6, while the setup and baseline comparisons for the large-scale node classification experiments are described separately in Section 5.8.7.

Results on graph-level tasks. Our proposed fractal nodes (FN and FN_M) consistently enhance the performance of baseline MPNNs on all benchmark datasets, often surpassing graph Transformer models. In Table 5-1, on Peptides-func, GINE+FN_M achieves an average precision (AP) of 0.7018, outperforming both Exphormer and GraphGPS. Our comparable performance with Graph-ViT and Exphormer on MNIST shows that fractal nodes can effectively capture local and global information without a self-attention layer.

Results on large-scale graphs. The effectiveness of our method is particularly evident in large-scale graph experiments in Table 5-2 of Section 5.8.7. On ogbn-arxiv, GCN+FN improves accuracy from 71.74% to 73.03%, while on ogbn-product, GraphSAGE+FN_M demonstrates a substantial improvement from 78.29% to 83.11%. Several Transformer-based models (marked as OOM — Out of Memory in Table 5-2) fail to scale to ogbn-products due to their quadratic complexity in attention computation. In contrast, our method maintains computational efficiency while achieving superior performance, which we discuss further

Table. 5-1: Test performance on two peptide datasets from LRGB [139] and four other benchmark datasets [141, 140]. ↑ denotes the higher the better and ↓ denotes the lower the better. The top three models are colored by first, second, third.

Method	PEPTIDES-FUNC	PEPTIDES-STRUCT	MNIST	CIFAR10	MOLHIV	MOLTOX21
	AP ↑	MAE ↓	Accuracy ↑	Accuracy ↑	ROCAUC ↑	ROCAUC ↑
GCN [22]	0.6328±0.0023	0.2758±0.0012	0.9269±0.0023	0.5423±0.0056	0.7529±0.0098	0.7525±0.0031
GINE [196]	0.6405±0.0086	0.2780±0.0021	0.9705±0.0023	0.6131±0.0035	0.7885±0.0034	0.7730±0.0064
GatedGCN [250]	0.6300±0.0029	0.2778±0.0017	0.9776±0.0017	0.6628±0.0017	0.7874±0.0119	0.7641±0.0057
GT [218]	-	-	0.9083±0.0016	0.5975±0.0029	0.7350±0.0040	0.7500±0.0060
GraphiT [251]	-	-	-	-	0.7460±0.0100	0.7180±0.0130
Graphormer [143]	-	-	-	-	0.7930±0.0040	0.7730±0.0800
Transformer + LapPE [252]	0.6326±0.0126	0.2529±0.0016	0.9083±0.0016	0.5975±0.0029	-	0.7323±0.0057
SAN + LapPE [252]	0.6384±0.0121	0.2683±0.0043	-	-	0.7775±0.0061	0.7130±0.0080
EGT [253]	-	-	0.9817±0.0009	0.6870±0.0041	-	-
GraphGPS [222]	0.6534±0.0091	0.2509±0.0014	0.9805±0.0013	0.7230±0.0036	0.7880±0.0101	0.7570±0.0040
GRIT [230]	0.6988±0.0082	0.2460±0.0012	0.9811±0.0011	0.7647±0.0089	-	-
Graph-ViT/MLP-Mixer [144]	0.6970±0.0080	0.2449±0.0016	0.9846±0.0009	0.7158±0.0009	0.7997±0.0102	0.7910±0.0040
Exphormer [231]	0.6527±0.0043	0.2481±0.0007	0.9841±0.0035	0.7469±0.0013	-	-
GECO [254]	0.6975±0.0025	0.2464±0.0009	-	-	0.7980±0.0200	-
CRaWl [255]	0.6963±0.0079	0.2506±0.0022	0.9794±0.0050	0.6901±0.0026	0.7707±0.1490	-
PNA [256]	-	-	0.9794±0.0012	0.7035±0.0063	0.7905±0.0132	-
GNN-AK+ [238]	0.6480±0.0075	0.2736±0.0012	-	0.7219±0.0013	0.7961±0.0119	-
SUN [257]	0.6730±0.0115	0.2498±0.0008	-	-	0.8003±0.0055	-
CIN [258]	-	-	-	-	0.8094±0.0057	-
GCN + FN	0.6802±0.0043	0.2530±0.0004	0.9393±0.0084	0.6006±0.0070	0.7564±0.0059	0.7670±0.0073
GINE + FN	0.6815±0.0059	0.2515±0.0020	0.9790±0.0012	0.6584±0.0069	0.7890±0.0104	0.7751±0.0029
GatedGCN + FN	0.6778±0.0056	0.2536±0.0019	0.9826±0.0012	0.7125±0.0035	0.7967±0.0098	0.7759±0.0054
GCN + FN_M	0.6787±0.0048	0.2464±0.0014	0.9455±0.0004	0.6413±0.0068	0.7866±0.0034	0.7882±0.0041
GINE + FN_M	0.7018±0.0074	0.2446±0.0018	0.9786±0.0004	0.6672±0.0068	0.8127±0.0076	0.7926±0.0021
GatedGCN + FN_M	0.6950±0.0047	0.2453±0.0014	0.9848±0.0005	0.7526±0.0033	0.8097±0.0047	0.7922±0.0054

Table. 5-2: Results on large-scale graphs.

Model	ogbn-arxiv	ogbn-product
# nodes	169,343	2,449,029
# edges	1,166,243	61,859,140
MLP	55.50±0.33	71.59±0.71
LINKX [30]	66.18±0.33	71.59±0.71
GraphGPS [222]	70.97±0.41	OOM
NAGphormer [259]	70.13±0.55	73.55±0.21
Exphormer [231]	72.44±0.28	OOM
NodeFormer [260]	69.86±0.25	72.93±0.13
DiffFormer [261]	72.41±0.40	74.16±0.31
PolyNormer [262]	71.82±0.23	82.97±0.28
SGFormer [263]	72.63±0.13	74.16±0.31
HC-GNN [242]	72.79±0.25	-
ANS-GT [246]	72.34±0.50	80.64±0.29
HSGT [244]	72.58±0.31	81.15±0.13
GCN	71.74±0.29	75.64±0.21
GCN + FN	73.03±0.37	81.29±0.21
GCN + FN_M	72.93±0.35	81.33±0.33
GraphSAGE	71.49±0.27	78.29±0.16
GraphSAGE + FN	72.70±0.11	83.07±0.35
GraphSAGE + FN_M	72.54±0.30	83.11±0.07

in Section 5.5.4. This highlights not only the effectiveness of fractal nodes in capturing both local and global graph information but also their practical applicability to large-scale graphs.

5.5.4 Runtime Comparison (Q4)

As we discussed in Section 5.4.2, our fractal nodes provide benefits in capturing long-range dependencies without increasing computational complexity. As shown in Table 5-3, when integrated with base MPNNs, fractal nodes introduce trivial computational overhead – GCN+FN maintains identical training time (1.27s) and memory usage (16.49GB) compared to the vanilla GCN. Our method uses common MPNN operations without introducing complex additional computations. In contrast, graph Transformers (e.g., GraphGPS and Exphormer) require substantially more computational resources (38.91GB and 34.04GB memory, respectively) due to their attention mechanisms. Given these results shown in Sections 5.5.3 and 5.5.4, we believe our method achieves a balance between accuracy and computational efficiency.

In Section 5.8.8, we also conduct experiments on synthetic Erdős-Rényi graphs with nodes ranging from 1,000 to 100,000 to evaluate efficiency and scalability, and we are able to verify linear space complexity while our GPU memory usage scaled linearly with the graph size.

In Table 5-16 of Section 5.8.13, we show that using METIS with $\mathcal{O}(|\mathcal{E}|)$ complexity enables efficient fractal node creation.

Table. 5-3: Training time per epoch and memory usage on ogbn-arxiv

Model	Runtime (s)	Mem. (GB)
GCN	1.27	16.49
GraphSAGE	0.55	7.74
GraphGPS	1.32	38.91
Exphormer	0.74	34.04
NodeFormer	1.20	16.30
DiffFormer	0.77	24.51
PolyNormer	0.31	16.09
GCN + FN	1.27	16.49
GCN + FN_M	1.27	16.49
GraphSAGE + FN	0.57	7.74
GraphSAGE + FN_M	0.58	7.76

5.5.5 Ablation, Sensitivity, and Additional Studies

We report ablation studies for $\omega_c^{(\ell)}$ and HPF in Sections 5.8.9.1 and 5.8.9.2. We report results when $\omega_c^{(\ell)}$ is zero, that is, without HPF, and when we use either a scalar parameter or a learnable vector parameter. We also report sensitivity studies on C, i.e., the number of fractal nodes, and additional analyses on a variant of message passing between fractal nodes and the distribution of subgraph size ratios in Sections 5.8.9.5, 5.8.9.6 and 5.8.11. We also provide the number of subgraphs that affect over-squashing in Corollary 5.8.4, which is related to the sensitivity study in Section 5.8.9.5. An analysis of the use of partitioning algorithms other than METIS, such as random, Louvain [264], Girvan-Newman [265] partitionings, is reported in Section 5.8.13.

MPNNs with fractal nodes vs. other augmented MPNNs. We compare our fractal nodes to 6 augmented MPNNs, including graph rewiring methods (see Section 5.8.9.3 for detailed setup). In Table 5-4, we show that our FN outperforms LASER and PANDA, as well as all 6 baselines on both datasets. If there is only one fractal node and no subgraph is created, our method can be reduced to the virtual node method, so we compare our methods and virtual nodes in Section 5.8.9.4.

Table. 5-4: Results of GCN with rewiring methods.

Method	PEPTIDES-FUNC	PEPTIDES-STRUCT
	AP ↑	MAE ↓
GCN	0.5930±0.0023	0.3496±0.0013
+ FoSR [12]	0.5947±0.0035	0.3473±0.0007
+ GTR [189]	0.5075±0.0029	0.3618±0.0010
+ SDRF [185]	0.5947±0.0126	0.3478±0.0013
+ BORF [13]	0.5994±0.0037	0.3514±0.0009
+ PANDA [180]	0.6028±0.0031	0.3272±0.0001
+ LASER [194]	0.6440±0.0010	0.3043±0.0019
+ FN	0.6445±0.0057	0.2535±0.0012

5.6 Limitations and Future Directions

While fractal nodes are effective, they are currently limited to MPNNs and rely on predefined graph partitioning. Future work could explore learnable partitioning and training strategies that align MPNN representations more closely with those of graph Transformers. Our approach may also benefit scientific domains such as materials science, where both local and global structural properties are essential.

5.7 Summary

We introduced *fractal nodes* that preserve the computational efficiency of MPNNs while enabling long-range interactions typically captured by graph Transformers. Our core idea is that graph partitioning naturally induces *fractal structure*, and fractal nodes enforce *feature similarity* within each subgraph. This design alleviates over-squashing and improves expressive power through subgraph-level representations. Experiments show that fractal nodes consistently improve MPNN performance and achieve results that are competitive with those of graph Transformers.

5.8 Appendix

5.8.1 Proof of Theorem 5.3.1

Proof. The mean pooling operation aggregated the features of all nodes in the subgraph or graph by computing the average:

$$f_c^{mean} = \frac{1}{n} \sum_{v \in \mathcal{C}_c} h_v. \tag{5.12}$$

To understand this operation in the frequency domain, we use the discrete Fourier transform (DFT), which transforms the node feature matrix H_c into its frequency domain. The DFT of a signal h_v is represented as:

$$\mathcal{F}(h_v) = \mathsf{DFT} \cdot h_v, \tag{5.13}$$

where $\mathsf{DFT} \in \mathbb{C}^{n \times n}$ is the Fourier matrix. The rows of the Fourier matrix are given by the Fourier basis vectors, which are complex exponential functions. These basis vectors represent different frequencies, and each row in the DFT corresponds to a specific frequency component. The first row of the Fourier matrix DFT corresponds to the DC component, which is the lowest frequency component of the signal. This row is a vector of ones:

$$\mathsf{DFT}[1, :] = \frac{1}{\sqrt{n}} \cdot [1, 1, \ldots, 1]. \tag{5.14}$$

This row corresponds to the mean or average of the signal. Therefore, when we project the input signal onto this basis vector, we are effectively extracting the global, smooth structure of the signal.

The DC component of the DFT is then expressed as:

$$DC[x] = \mathsf{DFT}^{-1}\mathrm{diag}(1, 0, \ldots, 0)\mathsf{DFT}x = \frac{1}{n}11^{\intercal}x. \tag{5.15}$$

This operation corresponds to projecting the input signal x onto the vector of ones, effectively averaging all elements of x, which is exactly the result of mean pooling:

$$f_c^{DC} = \frac{1}{n}11^{\intercal}H_c = \frac{1}{n}\sum_{v \in \mathcal{C}_c} h_v. \tag{5.16}$$

□

Therefore, mean pooling captures the DC component of the signal, which is the lowest frequency component. This corresponds to extracting the global, smooth node features of the subgraph, but it does not retain higher-frequency variations, which represent the local details.

Thus, mean pooling is equivalent to applying a low-pass filter that only retains the DC component of the signal.

5.8.2 Theoretical Analyses

In this section, we provide theoretical analyses of fractal nodes to show how they mitigate over-squashing. Our analysis builds on effective resistance theory to characterize information flow in networks with fractal nodes.

Preliminaries on effective resistance. Following Black et al. [189] and Section 5.8.10, we recap the effective resistance in graphs. For a connected, non-bipartite graph, the pseudoinverse of the normalized Laplacian can be expressed as:

$$\hat{\mathbf{L}}^{+} = \sum_{j=0}^{\infty} \hat{\mathbf{A}}^{j}, \tag{5.17}$$

Furthermore, the effective resistance between nodes u and v can be written as:

$$R_{u,v} = \sum_{i=0}^{\infty} \left(\frac{1}{d_u} (\hat{\mathbf{A}}^{i})_{uu} + \frac{1}{d_v} (\hat{\mathbf{A}}^{i})_{vv} - \frac{2}{\sqrt{\deg_u \deg_v}} (\hat{\mathbf{A}}^{i})_{uv} \right), \tag{5.18}$$

where $(\hat{\mathbf{A}}^{i})_{u,v}$ represents the number of paths of length i between nodes u and v [189]. This equation intuitively demonstrates that shorter and more disjoint paths connecting two nodes result in lower effective resistance.

5.8.2.1 Effective Resistance with Fractal Nodes

Lemma 5.8.1 (Fractal Node Effective Resistance)**.** *Let $\mathcal{G}$ be a connected graph with $\mathcal{C}$ subgraphs and their associated fractal nodes. The effective resistance between any two nodes u, v with fractal nodes can be expressed as:*

$$R_f(u,v) = (1_u - 1_v)^T \mathbf{L}_f^{+} (1_u - 1_v),$$

where $\mathbf{L}_f$ is the augmented Laplacian incorporating fractal node connections:

$$\mathbf{L}_f = \begin{bmatrix} \mathbf{L} + \sum_{i=1}^{\mathcal{C}} \mathbf{F}_i \mathbf{F}_i^T & -[\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_{\mathcal{C}}] \\ -[\mathbf{F}1, \mathbf{F}2, ..., \mathbf{F}_{\mathcal{C}}]^T & \mathbf{I}_{\mathcal{C}} \end{bmatrix},$$

where $\mathbf{L}$ is the original Laplacian matrix, $\mathbf{F}_i$ is the incidence vector for fractal node i indicating its connections to the original nodes.

Similar to the path-based interpretation in [189], we can express $R_f(u,v)$ in terms of paths:

$$R_f(u,v) = \sum_{i=0}^{\infty} \left(\frac{1}{\deg_u} (\hat{\mathbf{A}}_f^{i})_{uu} + \frac{1}{\deg_v} (\hat{\mathbf{A}}_f^{i})_{vv} - \frac{2}{\sqrt{\deg_u \deg_v}} (\hat{\mathbf{A}}_f^{i})_{uv} \right) \tag{5.19}$$

where $\hat{\mathbf{A}}_f$ is the normalized adjacency matrix including fractal node connections.

5.8.2.2 Proof of Theorem 5.4.1

Theorem 6.4.1 (Resistance reduction)**.** *Let $\mathcal{G}$ be the original graph and $\mathcal{G}_f$ be the augmented graph with fractal nodes. For any nodes $u, v \in \mathcal{G}$, the effective resistance in $\mathcal{G}_f$ satisfies:*

$$R_f(u,v) \leq R(u,v),$$

where $R_f(u,v)$ is the effective resistance in $\mathcal{G}_f$ and $R(u,v)$ is the original effective resistance in $\mathcal{G}$.

Proof. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the original graph and $\mathcal{G}_f = (\mathcal{V} \cup \mathcal{F}, \mathcal{E} \cup \mathcal{E}_f)$ be the augmented graph with fractal nodes, where $\mathcal{F}$ is the set of fractal nodes and $\mathcal{E}_f$ is the set of edges connecting nodes to fractal nodes.

Following [189], we express the effective resistance in terms of path decomposition:

$$R_f(u,v) = \sum_{i=0}^{\infty} \left(\frac{1}{\deg_u} (\hat{\mathbf{A}}_f^i)_{uu} + \frac{1}{\deg_v} (\hat{\mathbf{A}}_f^i)_{vv} - \frac{2}{\sqrt{\deg_u \deg_v}} (\hat{\mathbf{A}}_f^i)_{uv} \right), \tag{5.20}$$

where $\hat{\mathbf{A}}_f$ is the normalized adjacency matrix of $\mathcal{G}_f$.

Let $\mathcal{P}_{uv}$ be the set of all paths connecting u and v in $\mathcal{G}_f$. The effective resistance can be expressed as:

$$R_f(u,v) = \min_{p \in \mathcal{P}_{uv}} \sum_{(x,y) \in p} r_{xy}, \tag{5.21}$$

where r_{xy} is the resistance of edge (x, y).

By Rayleigh's monotonicity principle [189], since $\mathcal{G}_f$ contains all edges of $\mathcal{G}$ plus additional edges through fractal nodes, adding these edges can only decrease the effective resistance between any pair of nodes. Therefore:

$$R_f(u,v) \leq R(u,v). \tag{5.22}$$

□

5.8.2.3 Proof of Theorem 5.4.2

Theorem 6.4.2 (Signal propagation with fractal nodes)**.** *For an MPNN with fractal nodes, the signal propagation between nodes u, v after ℓ layers satisfies:*

$$\|h_u^{(\ell)} - h_v^{(\ell)}\| \leq \exp(-\ell / R_f(u,v)) \|h_u^{(0)} - h_v^{(0)}\|,$$

where $R_f(u,v)$ is the effective resistance in the augmented graph with fractal nodes.

Proof. First, the message passing process in MPNN (i.e., GCN) with fractal nodes can be expressed as:

$$h_v^{(\ell+1)} = \sigma \left(W h_v^{(\ell)} + \sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{\deg_v \deg_u}} W h_u^{(\ell)} + W_f h_f^{(\ell)} \right), \tag{5.23}$$

where $h_f^{(\ell)}$ is the fractal node representation. To analyze the signal propagation, we consider the continuous-time analog by removing the nonlinearity σ:

$$\frac{d}{dt} h_v(t) = -\mathbf{L}_f h_v(t), \tag{5.24}$$

The solution to this differential equation is:

$$h_v(t) = \exp(-t\mathbf{L}_f)h_v(0), \tag{5.25}$$

The signal difference between two nodes u, v is bounded as follows:

$$\begin{aligned} ||h_u(t) - h_v(t)|| &= \|(\exp(-t\mathbf{L}_f))(h_u(0) - h_v(0))\| && (5.26)\\ &\leq ||exp(-t\mathbf{L}_f)|| \cdot ||h_u(0) - h_v(0)|| && (5.27)\\ &\leq \exp(-t/R_f(u,v))||h_u(0) - h_v(0)|| && (5.28) \end{aligned}$$

The last inequality comes from the spectral bound related to the effective resistance $R_f(u, v)$ in the graph augmented with fractal nodes. Mapping back to the discrete layer steps by setting $t = \ell$, we obtain our desired bound:

$$||h_u^{(\ell)} - h_v^{(\ell)}|| \leq \exp(-\ell/R_f(u,v))||h_u^{(0)} - h_v^{(0)}||, \tag{5.29}$$

This provides the worst-case signal propagation bound in the graph with fractal nodes. By the previously proven Theorem 5.4.1, we know that $R_f(u, v) \leq R(u, v)$; thus, fractal nodes provide better signal propagation guarantees than the original graph. □

Corollary 5.8.2 (Improved signal propagation)**.** *Since $R_f(u, v) \leq R(u, v)$ by the Resistance Reduction theorem, fractal nodes improve the worst-case signal propagation bound compared to the original graph:*

$$\exp(-\ell/R_f(u,v)) \leq \exp(-\ell/R(u,v)).$$

5.8.2.4 Total Resistance Analysis

Theorem 5.8.3 (Total Resistance with Fractal Nodes)**.** *Let $\mathcal{G}_f$ be the graph augmented with C fractal nodes. The total effective resistance satisfies:*

$$R_{tot}^f = n \cdot tr(\mathbf{L}_f^+) = n \cdot \sum_{i=2}^{n+C} \frac{1}{\sigma_i},$$

where $\mathbf{L}_f$ is the augmented Laplacian and σ_i are its eigenvalues.

Proof. The total resistance can be expressed through the trace of the pseudo-inverse of the Laplacian matrix $\mathbf{L}_f$. By construction, $\mathbf{L}_f$ has dimension $(n + C) \times (n + C)$ and its eigendecomposition yields $n+C$ eigenvalues. The pseudo-inverse $\mathbf{L}_f^+$ has the same eigenvectors as $\mathbf{L}_f$ with reciprocal non-zero eigenvalues, giving us the stated formula. The factor n appears because we sum over all pairs of the original nodes n. □

Corollary 5.8.4 (Impact of Fractal Node Count)**.** *For a graph $\mathcal{G}$ augmented with C fractal nodes, the total resistance decreases with C as $R_{tot}^f = n \cdot \sum_{i=2}^{n+C} \frac{1}{\sigma_i}$, where additional eigenvalues from larger C decrease the sum. This leads to improved signal propagation bounds $||h_u^{(\ell)} - h_v^{(\ell)}|| \leq \exp(-\ell/R_f(u, v))$.*

5.8.3 Implementation Details

Metis partitioning for Fractal Node creation. To create fractal nodes, we employ METIS [236], a graph clustering algorithm known for its excellent balance between accuracy and computational efficiency. METIS partitions a graph into a predefined number of clusters, maximizing within-cluster connections while minimizing between-cluster links. This approach effectively captures the community structure of the graph.

However, using non-overlapping partitions could result in the loss of important edge information, particularly at the boundaries between partitions. To address this issue and retain all original edges, we introduce an overlapping subgraph. After the initial METIS partitioning, we expand each partition to include nodes from neighboring partitions.

Formally, we first apply METIS to partition a graph $\mathcal{G}$ into C non-overlapping subgraphs: $\{\mathcal{V}_1, \ldots, \mathcal{V}_C\}$ such that $\mathcal{V} = \{\mathcal{V}_1 \cup \ldots \cup \mathcal{V}_C\}$ and $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing, \forall i \neq j$, where C is the number of fractal nodes or subgraphs. Then, we expand these subgraphs to include k-hop neighborhoods:

$$\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \{\mathcal{N}_k(j) | j \in \mathcal{V}_i\}, \tag{5.30}$$

where $\mathcal{N}_k(j)$ defines the k-hop neighbourhood of node j. This expansion ensures that each subgraph retains information about its immediate surroundings. The choice of k allows us to control the degree of overlap between subgraphs. A larger k value increases the overlap, potentially capturing more global information but at the cost of increased computational complexity. This overlapping subgraph approach allows our fractal nodes to capture both local structural details and broader subgraph-level information, enhancing the model's ability to learn multi-scale representations of the graph structure.

Instances of our framework. We describe update equations for how our fractal node is applied to MPNN. The update equation for GatedGCN + FN is the following:

$$\begin{aligned}
\widetilde{h}_{v,c}^{(\ell+1)} &= \sigma\Big(\Omega^{(\ell)} h_{v,c}^{(\ell)} + \sum_{u \in N(v)} \mathsf{gate}^{(\ell)}(h_{v,c}^{(\ell)}, h_{u,c}^{(\ell)}) \odot h_{u,c}^{(\ell)} W_1^{(\ell)}\Big), \\
f_c^{(\ell+1)} &= \mathsf{LPF}(\widetilde{h}_{v,c}^{(\ell+1)}) + \omega^{(\ell)} \cdot \mathsf{HPF}(\widetilde{h}_{v,c}^{(\ell+1)}), \\
h_{v,c}^{(\ell+1)} &= \widetilde{h}_{v,c}^{(\ell+1)} + f_c^{(\ell+1)}, \\
\mathsf{gate}^{(\ell)}(h_{v,c}^{(\ell)}, h_{u,c}^{(\ell)}) &= \mathsf{sigmoid}(W2^{(\ell)} h_{v,c}^{(\ell)} + W_3^{(\ell)}) h_{u,c}^{(\ell)},
\end{aligned} \tag{5.31}$$

where σ is a ReLU activation function, $W_0^{(\ell)}$, $W_1^{(\ell)}$, $W_2^{(\ell)}$, $W_3^{(\ell)}$ are learnable weight matrices, $\mathsf{gate}^{(\ell)}$ is a gating mechanism that controls the information flow between nodes.

The update equation for GINE + FN is the following:

$$\begin{aligned}
\widetilde{h}_{v,c}^{(\ell+1)} &= \mathsf{MLP}^{(\ell)}\Big((1 + \epsilon^{(\ell)}) \cdot h_{v,c}^{(\ell)} + \sum_{u \in N(v)} \sigma(h_{u,c}^{(\ell)} + e_{uv}^{(\ell)})\Big), \\
f_c^{(\ell+1)} &= \mathsf{LPF}(\widetilde{h}_{v,c}^{(\ell+1)}) + \omega_c^{(\ell)} \cdot \mathsf{HPF}(\widetilde{h}v, c^{(\ell+1)}), \\
h_{v,c}^{(\ell+1)} &= \widetilde{h}_{v,c}^{(\ell+1)} + f_c^{(\ell+1)},
\end{aligned} \tag{5.32}$$

where $\epsilon^{(\ell)}$ is a learnable scalar parameter, and $e_{uv}^{(\ell)}$ is a edge hidden vector between node u and v.

Note that the positional encoding scheme and readout function schemes can also be applied to MPNNs with fractal nodes.

Positional encoding. When integrating our fractal node to MPNN, we incorporate two distinct positional encodings (PE): an absolute PE for individual nodes and a relative PE for fractal nodes.

For node-level encoding, we consider dataset-specific approaches. We utilize random-walk structural encoding (RWSE) for molecular graphs and Laplacian eigenvector encodings for super-pixel image-based tasks. To enhance robustness, we randomly flip the sign of Laplacian eigenvectors during training.

Let $M \in \{0,1\}^{C\times|\mathcal{V}|}$ be a binary matrix where each row corresponds to a fractal node and each column to an original graph node. $M_{ij} = 1$ if node j belongs to fractal node i, and 0 otherwise. Then, the coarsened adjacency matrix is computed as $A^C = MM^\top$. This operation effectively counts the number of connections between fractal nodes, where A^C_{ij} represents the number of edges between fractal nodes i and j in the original graph. We then derive a positional encoding $p_v \in \mathbb{R}^{d_p}$ for each fractal node from this coarsened adjacency matrix. This encoding is incorporated into the fractal node representation through a linear transformation:

$$f_v^{(L)} = Tp_v + Of_v^{(L)} + b \in \mathbb{R}^d, \tag{5.33}$$

where $T \in \mathbb{R}^{d\times d_p}$ and $O \in \mathbb{R}^{d\times d}$ are learnable transformation matrices, and $b \in \mathbb{R}^d$ is a learnable bias vector.

By incorporating relative positional information between fractal nodes, we enable the FN_M variant to better use the hierarchical structure of the graph.

5.8.4 Fractal Structure and Node Centrality

In this section, we describe how we calculate the fractal structure of a network by comparing the node centrality distributions between the original graph and its subgraphs using betweenness centrality. Specifically, we use the Kolmogorov-Smirnov (KS) test to measure the similarity between these distributions.

Fractality definition. We define the fractality of a graph as the degree to which the properties of the subgraphs resemble those of the original graph when the graph is partitioned consistently. In this work, we focus on how the betweenness centrality distribution of the original graph compares to those of its subgraphs.

Let $\Psi(x)$ represent the node centrality distribution function for the original graph, and let $\Psi_0(x), \ldots, \Psi_{32}(x)$ represent the centrality distributions for each of the subgraphs obtained by

partitioning the original graph into 32 subgraphs. We aim to quantify the similarity between $\Psi(x)$ and the subgraph distributions using the KS test.

Kolmogorov-Smirnov test. The KS test is a non-parametric test that compares the empirical cumulative distribution function (CDF) $\Psi_n(x)$ of the sample (subgraph centrality) with the CDF $\Psi(x)$ of the reference distribution (original graph centrality). The KS test statistic D is defined as:

$$D = \sup_x |\Psi_n(x) - \Psi(x)|, \tag{5.34}$$

where D represents the maximum distance between the two CDFs. A smaller D value indicates higher similarity between the two distributions.

Similarity metric. We define the similarity between the original graph and a subgraph as $1 - D$, where D is the KS test statistic. Therefore, a higher $1 - D$ value implies greater similarity. For each graph, we compute the similarity for all C subgraphs, yielding C similarity values.

Fractality calculation. In our structural fractality evaluation, we identify the subgraph whose centrality distribution is most similar to that of the original graph. This is because not all subgraphs need to exhibit structural similarity for the graph to be considered fractal-like; the presence of one or more highly similar subgraphs is indicative of fractality. Thus, we take the maximum of the C similarity values $(1 - D)$ as a structural similarity score for the graph:

$$\text{Similarity Score} = \max_i (1 - D_i), \tag{5.35}$$

where D_i is the KS test statistic for the i-th subgraph. This approach allows us to compute the structural similarity score for a single graph based on betweenness centrality.

5.8.5 Detailed Discussion on Section 5.5.2

To thoroughly analyze the role of positional encodings (PEs) and fractal nodes in model expressivity, we conducted extensive ablation studies analyzing different combinations of structural components. Table 5-5 shows results across three synthetic datasets (CSL, SR25, EXP) designed to test model expressiveness.

Our ablation study reveals several important insights about the interplay between positional encodings and our method. Without PEs, base MPNNs (GCN, GINE, GatedGCN) consistently show limited expressiveness across all datasets, achieving only 10.00% on CSL, 6.67% on SR25, and approximately 51-52% on EXP. Adding PEs substantially improves base model performance, as evidenced by GCN's significant improvements from 10.00% to 76.17% on CSL and from 52.17% to 100% on EXP.

Notably, even without any positional encodings, our fractal node variants demonstrate significantly enhanced expressivity. GINE+FN_M achieves 47.33% on CSL and 95.58% on EXP without any PE, while GatedGCN+FN_M reaches 49.67% on CSL. All FN_M variants achieve 100% on SR25 regardless of PE configuration, and this indicates that our method provides inherent structural awareness independent of positional encodings.

Table. 5-5: Synthetic results (Accuracy ↑). The gray-shaded rows represent the results without using PE and are the fairest to compare against.

Method	Ablation		Dataset		
	PE (Original Graph)	PE (Coarsened Graph)	CSL	SR25	EXP
GCN	✗	N/A	10.00	6.67	52.17
GCN	✓	N/A	76.17	100.0	100.0
GINE	✗	N/A	10.00	6.67	51.35
GINE	✓	N/A	100.0	100.0	100.0
GatedGCN	✗	N/A	10.00	6.67	51.25
GatedGCN	✓	N/A	100.0	100.0	100.0
GCN + FN_M	✗	✗	39.67	100.0	86.40
GCN + FN_M	✗	✓	76.17	100.0	100.0
GCN + FN_M	✓	✗	100.0	100.0	100.0
GCN + FN_M	✓	✓	100.0	100.0	100.0
GINE + FN_M	✗	✗	47.33	100.0	95.58
GINE + FN_M	✗	✓	84.83	100.0	100.0
GINE + FN_M	✓	✗	100.0	100.0	100.0
GINE + FN_M	✓	✓	100.0	100.0	100.0
GatedGCN + FN_M	✗	✗	49.67	100.0	96.50
GatedGCN + FN_M	✗	✓	81.83	100.0	100.0
GatedGCN + FN_M	✓	✗	100.0	100.0	100.0
GatedGCN + FN_M	✓	✓	100.0	100.0	100.0

5.8.6 Experimental Details on Graph-level Tasks

In this section, we provide further details about our experiments.

5.8.6.1 Dataset Description

We provide the descriptions and statistics of all datasets used in our experiments.

Peptides-func & Peptides-struct. (`CC BY-NC 4.0 License`) [139]: These datasets comprise 16K atomic peptide graphs from SAT-Pdb, with residues as nodes. They differ in their graph-level tasks: PETIDES-FUNC is a multi-label classification task with 10 nonexclusive functional classes, while PEPTIDES-STRUCT involves regression on 11 3D structural properties. Dataset splitting utilizes a meta-class holdout approach based on the original peptide labels.

MNIST & CIFAR10. (CC BY-SA 3.0 and MIT License): These datasets adapt popular image classification tasks to graph classification. Images are converted to graphs using super-pixels, representing homogeneous intensity regions. Both are 10-class classification tasks following standard splits: 55K/5K/10K for MNIST and 45K/5K/10K for CIFAR10 (train/validation/test).

Molhiv & Moltox21. (`MIT License`) [141]: These molecular property prediction datasets use common node and edge features representing chemophysical properties, pre-processed with RDKit [266]. Molecules are represented as graphs with atoms as nodes and chemical bonds as edges. Node features are 9-dimensional, including atomic number, chirality, and other properties. Predefined scaffold partitions are used: MOLTOX21 6K/0.78K/0.78K and MOLHIV 32K/4K/4K for training/validation/test.

CSL. CSL [247] is a synthetic dataset testing GNN expressivity, containing 150 4-regular graphs in 10 isomorphism classes. These graphs, indistinguishable by 1-WL tests, form cycles with skip-links. The task is to classify them into their respective isomorphism classes.

EXP. EXP [248] consists of 600 graph pairs that 1&2-WL tests fail to distinguish, aiming to classify these into two categories.

SR25. SR25 [249] consists of 15 strongly regular graphs (3-WL indistinguishable) with 25 nodes each, forming a 15-way classification problem.

TreeNeighbourMatch. Proposed by Alon and Yahav [10], this synthetic dataset highlights over-squashing in MPNNs. It uses binary trees of depth r (problem radius), requiring information propagation from leaves to a target node for label prediction, thus demonstrating over-squashing issues.

5.8.6.2 Hardware Specifications and Libraries

We have implemented our method using PyTorch-Geometric, and built on the source code of Rampášek et al. [222][3] and He et al. [144][4]. All experiments were performed using the following software and hardware environments: Ubuntu 18.04 LTS, Python 3.7.13, PyTorch 1.12.1, PyTorch Geometric 2.5.2, torch-scatter 2.1.0, torch-sparse 0.6.16, Numpy 1.24.3, metis 0.2a5, CUDA 11.3, NVIDIA Driver 465.19, i9 CPU, NVIDIA RTX 3090/A6000.

5.8.6.3 Setup & Hyperparameters

We use the same learning rates and weight decay to GCN, GINE, and GatedGCN, and the hyperparameters we considered are shown in Tables 5-6 to 5-8. The experimental results of MPNN are the same as the results using positional encoding, and we use the setup of He et al. [144].

In Tables 5-6 to 5-8, we report the hyperparameters used in our experiments. L_M denotes the number of layers of MLP-Mixer.

Table. 5-6: Hyperparameter search space of fractal nodes for benchmark datasets

Hyperparameters	Search Space
$\omega_c^{(\ell)}$	{SC, VC}
C	{4, 8, 16, 32}
HPF	{True, False}
k-hop	{0, 1, 2}
L	{2, 3, 4, 5, 6, 7, 8}
L_M	{1, 2, 4}

[3] https://github.com/rampasek/GraphGPS
[4] https://github.com/XiaoxinHe/Graph-ViT-MLPMixer

Table. 5-7: Best hyperparameter of FN for PEPTIDES-FUNC, PEPTIDES-STRUCT, MNIST, CIFAR10, MOLHIV, and MOLTOX21.

Hyperparameter	Method	PEPTIDES-FUNC	PEPTIDES-STRUCT	MNIST	CIFAR10	MOLHIV	MOLTOX21
$\omega_c^{(\ell)}$	GCN	VC	SC	VC	VC	VC	SC
	GINE	SC	VC	VC	VC	SC	SC
	GatedGCN	SC	VC	VC	VC	VC	SC
C	GCN	32	32	32	32	32	32
	GINE	32	32	32	32	32	32
	GatedGCN	32	32	32	32	32	32
HPF	GCN	True	True	True	True	True	True
	GINE	True	True	True	True	True	True
	GatedGCN	True	True	True	True	True	True
k-hop	GCN	1	1	1	1	1	1
	GINE	1	1	1	1	1	1
	GatedGCN	1	1	1	1	1	1
L	GCN	4	4	4	7	2	4
	GINE	4	4	4	7	2	4
	GatedGCN	4	4	4	7	2	4

Table. 5-8: Best hyperparameter of FN_M for PEPTIDES-FUNC, PEPTIDES-STRUCT, MNIST, CIFAR10, MOLHIV, and MOLTOX21.

Hyperparameter	Method	PEPTIDES-FUNC	PEPTIDES-STRUCT	MNIST	CIFAR10	MOLHIV	MOLTOX21
$\omega_c^{(\ell)}$	GCN	VC	SC	VC	VC	VC	VC
	GINE	SC	VC	VC	VC	VC	SC
	GatedGCN	VC	SC	VC	VC	VC	VC
C	GCN	32	32	32	32	32	32
	GINE	32	16	32	32	32	32
	GatedGCN	32	32	4	4	32	32
HPF	GCN	True	True	True	True	True	True
	GINE	True	True	True	True	True	True
	GatedGCN	True	True	False	True	True	True
k-hop	GCN	1	1	1	1	2	1
	GINE	1	1	1	1	2	1
	GatedGCN	1	1	1	1	2	1
L	GCN	4	4	4	7	2	5
	GINE	4	4	4	7	2	4
	GatedGCN	4	4	4	8	2	5
L_{M}	GCN	2	2	4	1	2	4
	GINE	2	2	4	1	2	4
	GatedGCN	2	2	4	1	2	4

5.8.7 Experimental Details on Large-scale Node Classification

5.8.7.1 Implementation for Node Classification

While our main experiment focuses on graph-level tasks, our fractal node method can be naturally extended to node classification tasks. The key distinction lies in how we use the processed fractal node representations from the MLP-Mixer layer to make node-level predictions rather than graph-level ones.

For graph-level tasks, as shown in Equation (5.8), the fractal nodes are mixed through the MLP-Mixer to produce

$$\tilde{F} = \mathsf{MLPMixer}(F^{(L)}), \quad F^{(L)} = [f_1^{(L)}, f_2^{(L)}, ..., f_C^{(L)}]. \tag{5.36}$$

These mixed representations are then used directly for graph-level prediction via global pooling.

For node classification, however, we need to propagate this mixed global information back to individual nodes. After the MLP-Mixer processes the C fractal nodes according to Equations (5.9) and (5.10), we obtain $\tilde{F}^{(L)} \in \mathbb{R}^{C \times d}$. These processed fractal node representations need to be aligned with all nodes in their respective subgraphs.

Let $\mathcal{V}_c$ be the set of nodes in subgraph c. For each node $v \in \mathcal{V}_c$, we update its final representation by combining its current features with the processed fractal node information from its corresponding subgraph:

$$h_v^{(\text{final})} = h_v^{(L)} + \tilde{f}_c^{(L)}, \quad \forall v \in \mathcal{V}_c, \tag{5.37}$$

where $\tilde{f}_c^{(L)}$ is the c-th row of $\tilde{F}^{(L)}$ corresponding to the fractal node of subgraph c. This operation ensures that each node receives the processed global context from its subgraph's fractal node and maintains consistency with our method while adapting it for node-level predictions.

In implementation, this process can be efficiently vectorized using a batch membership index that maps each node to its corresponding fractal node representation. This adaptation allows our fractal node framework to effectively handle both graph-level and node-level tasks while maintaining its computational efficiency and theoretical properties.

5.8.7.2 Dataset Description

We consider a collection of large-scale graphs released by the Open Graph Benchmark (OGB) [142]: ogbn-arxiv and ogbn-products with node numbers 0.16M and 2.4M, respectively. We maintain all the OGB standard evaluation settings.

ogbn-arxiv. (`ODC-BY License`) [142]: ogbn-arxiv is a citation network among all Computer Science (CS) papers on Arxiv, as noted by Hu et al. [142]. In this network, nodes represent individual Arxiv papers, with edges denoting citation relationships between them. Each paper

is linked to a 128-dimensional feature vector, derived by averaging the word embeddings from its title and abstract. These embeddings are created using WORD2VEC. The aim is to predict the subject areas of CS papers on Arxiv, focusing on 40 distinct subject areas. We employ the split method from Hu et al. [142], training on papers up to 2017, validating with 2018 publications, and testing on papers published from 2019.

ogbn-products. (`Amazon License`) [142]: ogbn-products dataset is a large-scale undirected graph representing Amazon's product co-purchasing network, where nodes correspond to products and edges indicate co-purchase relationships. Each node is associated with a feature vector derived from bag-of-words representations of product descriptions, and the task is to classify products into 47 top-level categories. To better reflect real-world scenarios, the dataset uses a sales-based split: nodes are sorted by popularity, with the top 8% used for training, the next 2% for validation, and the rest for testing – simulating a setting where labels are available only for the most popular products.

5.8.7.3 Setup & Hyperparameters

Baselines. Our main focus lies on classic MPNNs: GCN [22], and GraphSAGE [41]; the state-of-the-art scalable graph Transformers: GraphGPS [222], NAGphormer [259], Exphormer [231], NodeFormer [260], DiffFormer [261], PolyNormer [267], and SGFormer [263]; hierarchical methods: HC-GNN [242], ANS-GT [246], and HSGT [244]; MLP-based method: LINKX [30].

Table. 5-9: Best hyperparameter of FN for ogbn-arxiv and ogbn-product

Hyperparameter	Method	ogbn-arxiv	ogbn-product
$\omega_c^{(\ell)}$	GCN	SC	SC
	GraphSAGE	SC	VC
C	GCN	64	32
	GraphSAGE	64	32
HPF	GCN	True	True
	GraphSAGE	True	True
d	GCN	512	1
	GraphSAGE	256	1
L	GCN	5	4
	GraphSAGE	4	4

Setting. We conduct hyperparameter tuning on classic MPNNs, which is consistent with the hyperparameter search space of Deng et al. [262]. Specifically, we use the Adam optimizer with a learning rate from {0.001, 0.005, 0.01} and an epoch limit of 2500. We tune the hidden dimension from {64, 256, 512}. We consider whether to use batch or layer normalization,

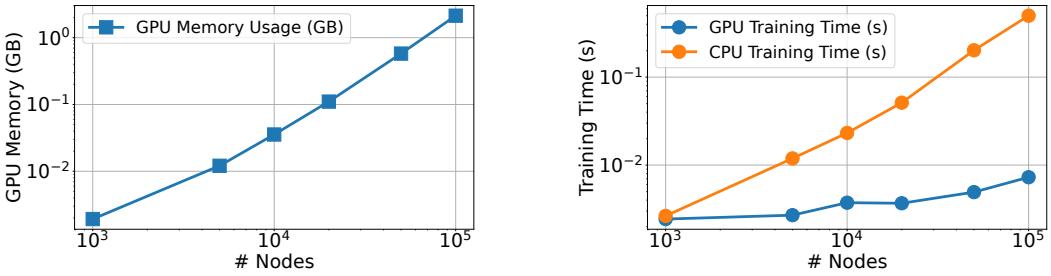
residual connections, and dropout rates from {0.2, 0.3, 0.5, 0.7}, the number of layers from {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, and C from {32, 64, 128}.

5.8.8 Scalability Analysis of of Fractal Node

To evaluate the efficiency and scalability of our FN integrated GCN model, we conducted experiments on synthetic Erdos-Renyi [268] graphs with node counts ranging from 1,000 to 100,000. The edge probability in the Erdos-Renyi network is set to achieve an average node degree of approximately 5, with the node feature dimension fixed at 100.

Figure 5.7(a) represents that the GPU memory usage of GCN+FN increases linearly with the graph size and validates its linear space complexity. Figure 5.7(b) shows the training time for both GPU and CPU implementations. The GPU training time exhibits a sub-linear growth trend as the graph size increases. This means the ability of fractal nodes to effectively use GPU parallelism for large-scale graph computations. In contrast, the CPU training time grows linearly with the graph size and indicates the sequential nature of CPU computations and its limitations in handling large-scale parallel graph operations.

The results demonstrate that the GPU device (RTX A6000 used in our experiments) efficiently handles the computational workload on varying graph sizes. These observations validate the scalability and practicality of our proposed GCN+FN model, particularly for large-scale graph learning tasks where both memory efficiency and computational speed are critical.



(a) Memory Usage vs. Graph Size

(b) Training Time vs. Graph Size

Figure. 5-7: GPU memory usage and training time of GCN+FN on synthetic graphs.

5.8.9 Ablation, Sensitivity and Additional Studies

5.8.9.1 Impact of HPF

We use both LPF and HPF to create fractal nodes, as shown in Equation (5.5). We analyze the cases when $\omega_c^{(\ell)}$ is 0, i.e., with and without HPF. Our results are reported in Table 5-10, and we obtain the best performance when using HPF in almost all cases.

Table. 5-10: Ablation study on HPF

Method	HPF	Peptides-func	Peptides-struct	MNIST	CIFAR10	MolHIV	MolTox21
		AP ↑	MAE ↓	Accuracy ↑	Accuracy ↑	ROCAUC ↑	ROCAUC ↑
GCN + FN	True	**0.6802**±0.0043	**0.2530**±0.0004	**0.9393**±0.0084	**0.6006**±0.0070	**0.7564**±0.0059	**0.7670**±0.0073
	False	0.6768±0.0016	0.2547±0.0023	0.9383±0.0102	0.5993±0.0081	0.7551±0.0084	0.7608±0.0093
GINE + FN	True	**0.6815**±0.0059	**0.2515**±0.0020	**0.9790**±0.0012	**0.6584**±0.0069	**0.7882**±0.0050	**0.7751**±0.0029
	False	0.6749±0.0111	0.2524±0.0021	0.9788±0.0008	0.6584±0.0069	0.7861±0.0054	0.7702±0.0045
GatedGCN + FN	True	**0.6778**±0.0071	**0.2536**±0.0019	**0.9826**±0.0012	**0.7125**±0.0035	**0.7967**±0.0098	**0.7759**±0.0054
	False	0.6661±0.0103	0.2609±0.0016	0.9801±0.0015	0.7010±0.0031	0.7908±0.0084	0.7674±0.0024
GCN + FN_M	True	**0.6787**±0.0048	**0.2464**±0.0014	**0.9455**±0.0004	**0.6413**±0.0070	**0.7866**±0.0034	**0.7882**±0.0041
	False	0.6778±0.0056	0.2461±0.0022	0.9448±0.0007	0.6130±0.0080	0.7689±0.0124	0.7874±0.0080
GINE + FN_M	True	**0.7018**±0.0074	**0.2446**±0.0018	**0.9786**±0.0004	**0.6672**±0.0068	**0.8127**±0.0076	**0.7926**±0.0021
	False	0.6647±0.0052	0.2484±0.0018	0.9744±0.0007	0.6670±0.0056	0.7959±0.0079	0.7895±0.0067
GatedGCN + FN_M	True	**0.6950**±0.0047	**0.2453**±0.0014	0.9836±0.0010	**0.7526**±0.0033	**0.8097**±0.0047	**0.7922**±0.0054
	False	0.6900±0.0055	0.2477±0.0005	**0.9848**±0.0005	0.7501±0.0042	0.7930±0.0057	0.7883±0.0067

5.8.9.2 Impact of type of $\omega_c^{(\ell)}$

When creating a fractal node, we can use a learnable scalar parameter (denoted as 'SC') or a learnable vector parameter (denoted as 'VC') to make the contribution of high-frequency components. We report the results in Table 5-11.

5.8.9.3 Comaprison to Graph Rewiring Methods

We compare our fractal nodes to no graph rewiring and 4 other state-of-the-art rewiring methods: DIGL [11], SDRF [185], FoSR [12], and BORF [12]. We also add the recent method, PANDA [180], to alleviate over-squashing without rewiring and the state-of-the-art method, LASER [194]. We replicate the experimental settings of Dwivedi et al. [139] and use the results from Barbero et al. [194]. We choose the hidden dimension to respect the 500k parameter budget. In our fractal node, we opt out of the positional encodings for a fair comparison.

5.8.9.4 Comaprison to Virtual Node Methods

To provide a comprehensive comparison with the existing virtual node method, we compare with the two virtual node methods by Hu et al. [141] (denoted as 'virtual node') and Rosenbluth et al. [269] (denoted as 'VN'). As shown in Table 5-12, both FN and FN_M outperform the GCN and GIN models augmented with virtual nodes from Hu et al. [141] on MolHIV and MolTox21. On the Peptides datasets, our methods show competitive results with the VN method of Rosenbluth et al. [269].

5.8.9.5 Sensitivity to C

The analysis of sensitivity to the number of fractal nodes (C) reveals distinct performance patterns in various datasets. As shown in Figure 5-8, for Peptides-func and Peptides-struct, there is relatively stable performance across different C values, with GINE+FN_M consistently outperforming the baseline GINE+FN. In MNIST, both GINE variants show an upward trend as C increases, with GINE+FN_M achieving peak accuracy at $C = 32$.

The optimal results are typically achieved at $C = 32$, which indicates that graph tasks benefit from finer-grained subgraph partitioning and additional mixing operations in FN_M. Overall, the results indicate that larger C values (16 or 32) generally yield better performance for most datasets.

5.8.9.6 Additional Results on All-layer Fractal Node Message Passing

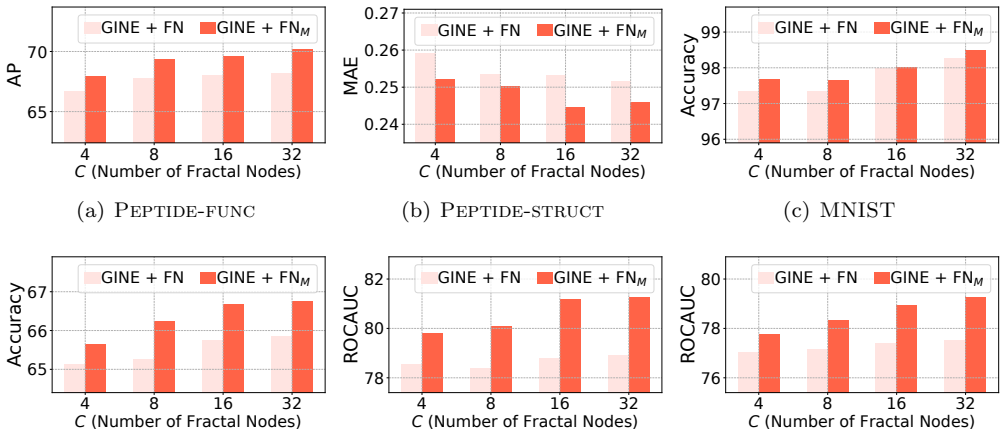
While our main FN_M design uses an MLP-Mixer in the final layer for fractal node interactions, we also explored an alternative approach with message passing between fractal nodes across

Table. 5-11: Sensitivity study on $\omega_c^{(\ell)}$

Method	$\omega_c^{(\ell)}$	PEPTIDES-FUNC	PEPTIDES-STRUCT	MNIST	CIFAR10	MOLHIV	MOLTOX21
		AP ↑	MAE ↓	Accuracy ↑	Accuracy ↑	ROCAUC ↑	ROCAUC ↑
GCN + FN	SC	0.6797±0.0056	**0.2530±0.0004**	0.9377±0.0080	0.6003±0.0075	0.7553±0.0061	**0.7670±0.0073**
	VC	**0.6802±0.0043**	0.2535±0.0033	**0.9393±0.0084**	**0.6006±0.0070**	**0.7564±0.0059**	0.7667±0.0045
GINE + FN	SC	**0.6815±0.0059**	0.2534±0.0016	0.9784±0.0010	0.6548±0.0088	**0.7882±0.0050**	**0.7751±0.0029**
	VC	0.6796±0.0024	**0.2515±0.0020**	**0.9790±0.0012**	**0.6584±0.0069**	0.7849±0.0047	0.7672±0.0009
GatedGCN + FN	SC	**0.6778±0.0071**	0.2546±0.0020	0.9813±0.0018	0.7083±0.0032	0.7910±0.0090	**0.7759±0.0054**
	VC	0.6647±0.0052	**0.2536±0.0019**	**0.9826±0.0012**	**0.7125±0.0035**	**0.7967±0.0098**	0.7662±0.0090
GCN + FN_M	SC	0.6773±0.0039	**0.2464±0.0014**	0.9444±0.0008	0.6405±0.0065	0.7762±0.0089	**0.7882±0.0041**
	VC	**0.6787±0.0048**	0.2485±0.0016	**0.9455±0.0004**	**0.6413±0.0070**	**0.7866±0.0034**	0.7862±0.0037
GINE + FN_M	SC	**0.7018±0.0074**	0.2451±0.0011	0.9735±0.0009	0.6655±0.0066	0.8070±0.0084	0.7924±0.0019
	VC	0.6926±0.0105	**0.2446±0.0018**	**0.9786±0.0004**	**0.6672±0.0068**	**0.8127±0.0076**	**0.7926±0.0021**
GatedGCN + FN_M	SC	0.6932±0.0056	**0.2453±0.0014**	0.9836±0.0010	0.7495±0.0051	**0.8097±0.0047**	**0.7922±0.0054**
	VC	**0.6950±0.0047**	0.2461±0.0009	**0.9836±0.0009**	**0.7526±0.0033**	0.8025±0.0087	0.7885±0.0043

Table. 5-12: Comparison to virtual node methods.

Method	PEPTIDES-FUNC	PEPTIDES-STRUCT	MOLHIV	MOLTOX21
	AP ↑	MAE ↓	ROCAUC ↑	ROCAUC ↑
GCN + virtual node [141]	-	-	0.7599±0.0119	0.7551±0.0100
GIN + virtual node [141]	-	-	0.7707±0.0149	0.7621±0.0062
GCN + VN [269]	0.6732±0.0066	0.2505±0.0022	-	-
GatedGCN + VN [269]	0.6823±0.0069	0.2475±0.0018	-	-
GCN + FN	0.6802±0.0043	0.2530±0.0004	0.7564±0.0059	0.7670±0.0073
GINE + FN	0.6815±0.0059	0.2515±0.0020	0.7890±0.0104	0.7751±0.0029
GatedGCN + FN	0.6778±0.0056	0.2536±0.0019	0.7967±0.0098	0.7759±0.0054
GCN + FN_M	0.6787±0.0048	0.2464±0.0014	0.7866±0.0034	0.7882±0.0041
GINE + FN_M	0.7018±0.0074	0.2446±0.0018	0.8127±0.0076	0.7926±0.0021
GatedGCN + FN_M	0.6950±0.0047	0.2453±0.0014	0.8097±0.0047	0.7922±0.0054



(a) PEPTIDE-FUNC (b) PEPTIDE-STRUCT (c) MNIST

(d) CIFAR10 (e) MOLHIV (f) MOLTOX21

Figure. 5-8: Sensitivity to C with GINE.

all layers (denoted as FN_A). This analysis aims to empirically validate our architectural choice.

Table 5-13 compares 3 variants: i) FN: is a base MPNN with no explicit fractal node interactions; ii) FN_A is an all-layer message passing between fractal nodes; and iii) FN_M is MLP-Mixer in the final layer only (our proposed approach). The results show that while FN_A shows some improvements over the base FN model in certain cases (e.g., MOLHIV accuracy improves from 0.7564 to 0.7783 for GCN), it consistently underperforms compared to our proposed FN_M design. This pattern holds across different base architectures (GCN, GINE, GatedGCN) and datasets.

These empirical results validate our design choice of using MLP-Mixer in the final layer rather than implementing message passing between fractal nodes throughout all layers. This

Table. 5-13: Comparison on FN, FN_A and FN_M

Method	Peptides-func	Peptides-struct	MolHIV	MolTox21
	AP ↑	MAE ↓	ROCAUC ↑	ROCAUC ↑
GCN + FN	0.6802±0.0043	0.2530±0.0004	0.7564±0.0059	0.7670±0.0073
GCN + FN_A	0.6582±0.0032	0.2531±0.0008	0.7783±0.0164	0.7600±0.0037
GCN + FN_M	**0.6787±0.0048**	**0.2464±0.0014**	**0.7866±0.0034**	**0.7882±0.0041**
GINE + FN	0.6815±0.0059	0.2515±0.0020	0.7890±0.0104	0.7751±0.0029
GINE + FN_A	0.6660±0.0067	0.2530±0.0011	0.8025±0.0100	0.7680±0.0056
GINE + FN_M	**0.7018±0.0074**	**0.2446±0.0018**	**0.8127±0.0076**	**0.7926±0.0021**
GatedGCN + FN	0.6778±0.0056	0.2536±0.0019	0.7967±0.0098	0.7759±0.0054
GatedGCN + FN_A	0.6658±0.0048	0.2531±0.0009	0.7898±0.0065	0.7642±0.0050
GatedGCN + FN_M	**0.6950±0.0047**	**0.2453±0.0014**	**0.8097±0.0047**	**0.7922±0.0054**

result indicates that the flexible mixing capabilities of the MLP-Mixer provide more effective fractal node interactions compared to explicit message passing approaches.

5.8.10 Effective Resistance and Signal Propagation

Effective resistance and signal propagation. Derived from the field of electrical engineering, the effective resistance between two nodes u and v in an electrical network is defined as the potential difference induced across the edges when a unit current is injected at one of each end [197]. Intuitively, it provides a physical measure of the ease of signal flow from one end to the other. Rayleigh's monotonicity principle, which says that adding paths or shortening existing paths can only decrease the effective resistance between two nodes [198], leads to the following interpretation: more and shorter disjoint paths connecting the nodes u and v lead to a lower resistance between them [189, 199]. Therefore, edges with higher effective resistance have fewer alternative paths or shortcuts for signals passing through that edge and thus struggle to propagate information, causing bottlenecks. The total effective resistance R_{tot}, the sum of the effective resistance among all pairs of nodes (see Equation (5.39)), is a key measure for measuring the overall degree of over-squashing across a graph.

Total effective resistance. The resistance between nodes u and v in the graph is given by

$$R_{u,v} = (1_u - 1_v)^\top \mathbf{L}^+ (1_u - 1_v), \tag{5.38}$$

where $\mathbf{L}$ is a Laplacian matrix, 1_v and 1_u are indicator vectors for node u and v, respectively. Total effective resistance, R_{tot}, is defined as the sum of effective resistance between all pairs

of nodes [197, 189]:

$$R_{tot} = \sum_{u>v} R_{u,v} = n \cdot \mathrm{Tr}(\mathbf{L}^{+}) = n \sum_{i}^{n} \frac{1}{\lambda_i}, \tag{5.39}$$

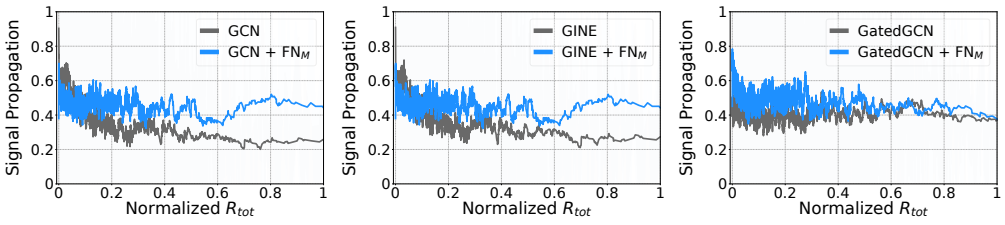
where λ_i is the i-th eigenvalues of $\mathbf{L}$ and $\mathbf{L}^{+}$ is the pseudoinverse of $\mathbf{L}$.

Signal propagation w.r.t. effective resistance. Here, we outline the experimental details for measuring signal propagation with respect to the normalized total effective resistance of the graphs. First, we randomly select a source node $v \in \mathcal{V}$, an entire node set, and assign d-dimensional feature vector to it, while all other nodes are initialized with zero vectors. Then, the amount of signal that has been propagated over the graph by the randomly initialized model with ℓ layers is given by

$$h_{\odot}^{(\ell)} = \frac{1}{d \max_{u \neq v} k_{\mathcal{G}}(u,v)} \sum_{t=1}^{d} \sum_{u \neq v} \frac{h_u^{(\ell),t}}{\|h_u^{(\ell),t}\|} k_{\mathcal{G}}(u,v), \tag{5.40}$$

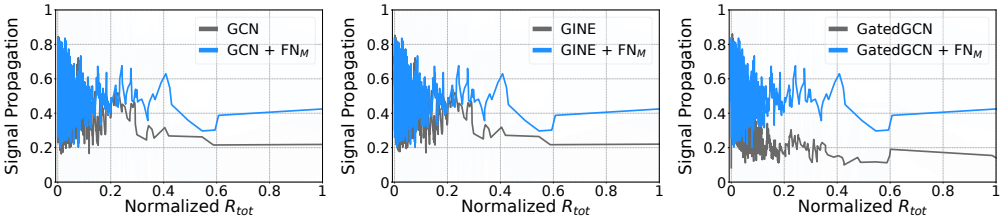
where $h_u^{(\ell),t}$ is the t-th feature of d-dimensional feature vector of node u at layer ℓ and $k_{\mathcal{G}}(u,v)$ is the distance between two nodes u and v, computed as a shortest path. Every unitary signal $h_u^{(\ell),t}/\|h_u^{(\ell),t}\|$ propagated across the graph G from the source node v is weighted by the normalized propagation distance $k_{\mathcal{G}}(u,v)/\max_{u \neq v} d_G(u,v)$ for all nodes $u \neq v$ and then averaged over entire d output channels. To estimate the total effective resistance of the graph, 10 nodes are randomly sampled from each graph, and the total effective resistance of the graph is estimated for each source node. The final $h_{\odot}^{(\ell)}$ and total resistance of the graph are obtained by averaging across the 10 sampled nodes. The experiment is repeated for every graph in the dataset, and the signal propagation measured for each graph is plotted against the normalized total effective resistance of the corresponding graph.

In Figures 5-9 to 5-11, we report the results of this analysis.



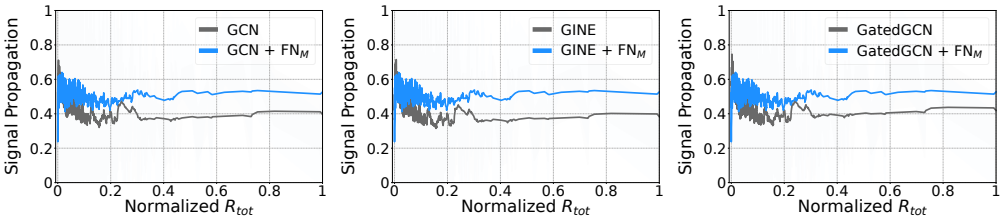
(a) GCN (b) GINE (c) GatedGCN

Figure. 5-9: The amount of signal propagated across the graphs w.r.t. the normalized R_{tot} in PEPTIDES-STRUCT.



(a) GCN (b) GINE (c) GatedGCN

Figure. 5-10: The amount of signal propagated across the graphs w.r.t. the normalized R_{tot} in MOLHIV.



(a) GCN (b) GINE (c) GatedGCN

Figure. 5-11: The amount of signal propagated across the graphs w.r.t. the normalized R_{tot} in MOLTOX21.

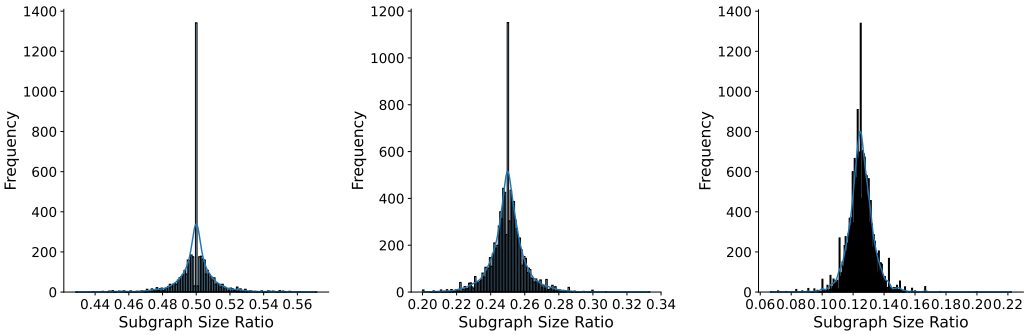
5.8.11 Distribution Analysis of Subgraph Size Ratio

We analyze the distribution of subgraph size ratios produced by METIS partitioning across different numbers of partitions (C) and datasets.

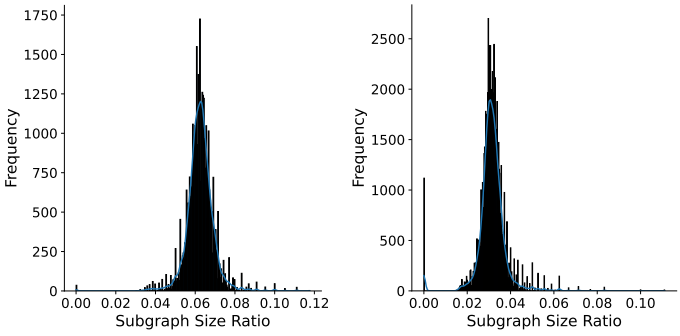
In general, as C increases from 2 to 32, the average subgraph size ratio naturally decreases since each partition contains a smaller portion of the original graph. The width of the distributions generally increases with C, indicating more variance in partition sizes with finer granularity. Most datasets show roughly normal or slightly skewed distributions around the expected mean ratio of $1/C$.

As shown in Figure 5-12, PEPTIDE-FUNC/STRUCT shows relatively tight, symmetric distributions. It indicates that METIS creates balanced partitions for molecular graphs. CIFAR10 and MNIST show distinct bimodal patterns, especially at $C = 16$ and $C = 32$, likely due to the regular grid-like structure of superpixel graphs (See Figure 5-13 and Figure 5-14). As shown in Figure 5-15 and Figure 5-16, MOLHIV and MOLTOX21 show broader distributions, particularly at higher C values, reflecting the more heterogeneous nature of these molecular graphs.

The consistent distributions for molecular datasets indicate METIS partitioning is well-suited for these graph types. The bimodal distributions in image-based graphs indicate the natural clustering of superpixels into regions of different sizes. Higher C values (i.e., 16, 32) generally maintain reasonable balance while allowing for more fine-grained capture of graph structure.

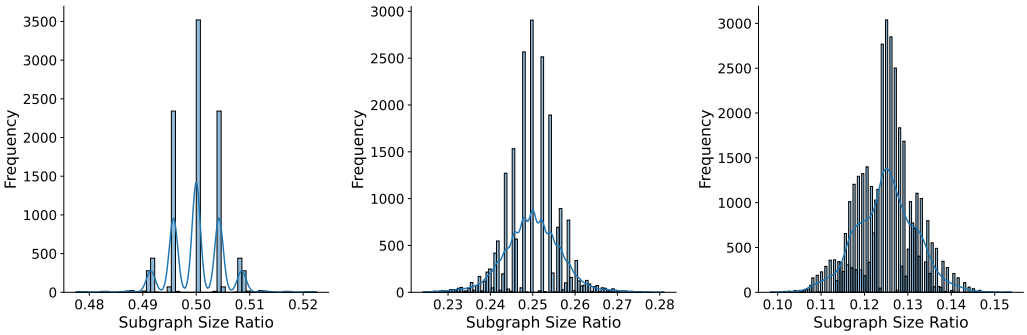


(a) $C = 2$ (b) $C = 4$ (c) $C = 8$

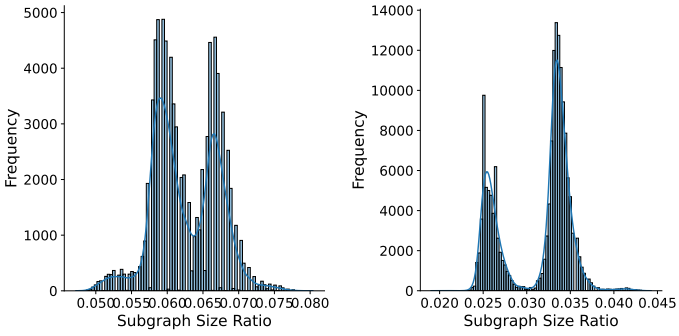


(d) $C = 16$ (e) $C = 32$

Figure. 5-12: Similarity of node centrality distribution in PEPTIDE-FUNC/STRUCT.

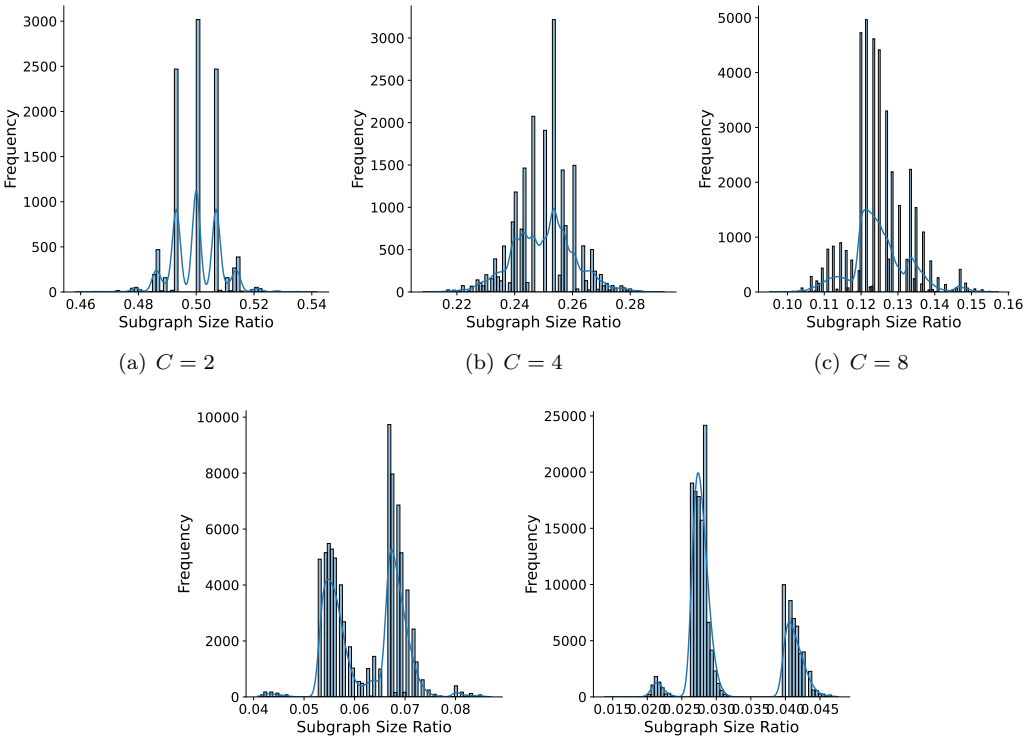


(a) $C = 2$ (b) $C = 4$ (c) $C = 8$



(d) $C = 16$ (e) $C = 32$

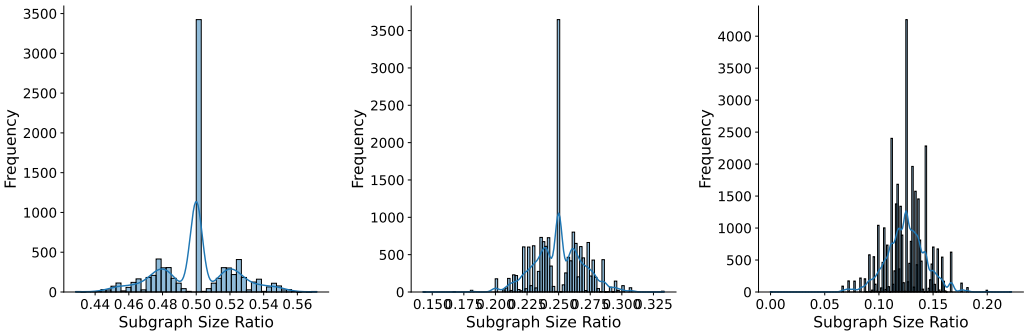
Figure. 5-13: Similarity of node centrality distribution in CIFAR10.



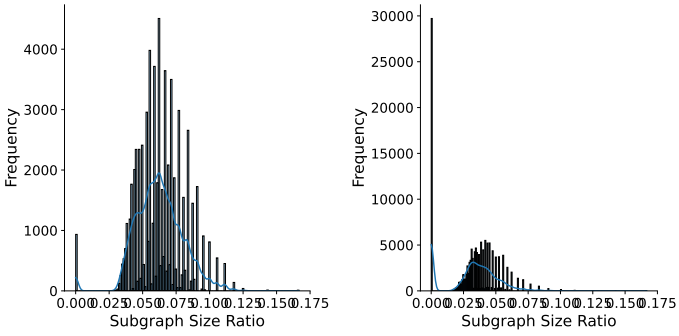
(a) $C = 2$ (b) $C = 4$ (c) $C = 8$

(d) $C = 16$ (e) $C = 32$

Figure. 5-14: Similarity of node centrality distribution in MNIST.

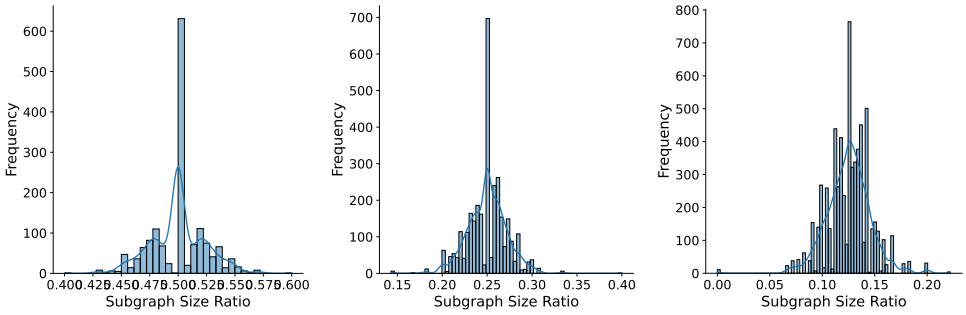


(a) $C = 2$ (b) $C = 4$ (c) $C = 8$

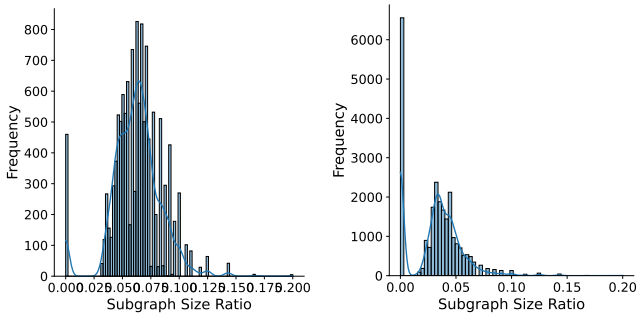


(d) $C = 16$ (e) $C = 32$

Figure. 5-15: Similarity of node centrality distribution in MolHIV.



(a) $C = 2$ (b) $C = 4$ (c) $C = 8$



(d) $C = 16$ (e) $C = 32$

Figure. 5-16: Similarity of node centrality distribution in MolTox21.

5.8.12 Connection to Renormalization Techniques

Our fractal nodes method draws inspiration from renormalization group techniques in physics, where complex systems are analyzed on different scales. While this connection is conceptual, the fundamental idea of scale transformation provides intuition for our approach. The renormalization involves replacing groups of interacting components with effective units. Similarly, our fractal nodes summarize subgraph information, although we maintain these summary units as fractal nodes alongside the original graph structure rather than replacing them.

From a complex network perspective, fractal nodes facilitate a transition from scale-free fractal networks to small-world networks. Similar to the renormalization techniques described by Wei et al. [270], our FN_M method introduces long-range interactions between the fractal nodes, giving small-world properties to the network [271]. We extend beyond renormalization in 3 aspects: (i) preserving the original structure while adding fractal nodes, (ii) enabling adaptive information flow through learned parameters, and (iii) maintaining exchange between local and global scales.

This architecture enables efficient information propagation through several mechanisms. The fractal nodes act as "shortcuts" in the network, reducing the effective distance information must traverse. Maintaining local and summarized representations enables simultaneous processing at multiple scales while preserving local network characteristics. This multi-scale processing capability addresses the over-squashing problem by facilitating efficient global information flow without sacrificing local structural information.

The key differences between our approach and classical renormalization highlight the factors we introduce specifically for graph learning tasks. While traditional renormalization uses fixed transformation rules in a unidirectional manner (fine to coarse), our method learns adaptive representations through trainable parameters and enables bidirectional information exchange. This creates a more flexible framework that captures complex relationships in graph-structured data while maintaining computational efficiency.

5.8.13 Different Partitioning Algorithms

To verify the effectiveness of partitioning other than METIS partitioning, we conduct experiments applying FN and FN_M to GINE on Peptides-func, Peptides-struct, MolHIV, and MolTox21 datasets using random partitioning and Louvain [264] and Girvan-Newman [265] partitioning.

In Table 5-14, our results provide a comprehensive comparison of different graph partitioning methods for GINE with FN and FN_M architectures on multiple molecular and peptide datasets. METIS consistently shows superior or competitive performance on all datasets. It achieves the best results in most cases, such as 0.6815 AP on Peptides-func with GINE+FN and 0.7018 AP with GINE+FN_M. While random partitioning shows surprisingly competitive performance, particularly on MolHiv where it achieves 0.8039 ROCAUC with

GINE+FN, community detection algorithms such as Louvain and Girvan-Newman generally underperform compared to METIS and random partitioning. The performance gap between different partitioning methods becomes more pronounced when using FN_M compared to FN. METIS shows more stable performance with lower standard deviations across all metrics. For molecular property prediction tasks, the choice of partitioning method appears less critical. However, on Peptides-func and Peptides-struct, METIS shows clear advantages with consistently lower MAE scores. These findings validate our choice of METIS as the default partitioning algorithm while suggesting that the optimal partitioning strategy may depend on the specific graph structure and task requirements.

In Table 5-15, the analysis of results on ogbn-arxiv provides additional insights into partitioning methods on large scale graph datasets. The performance differences between partitioning methods are relatively small, with scores ranging between 72.46% and 73.03% accuracy. For GCN+FN, METIS achieves the best performance at 73.03%, while random partitioning performs best for GCN+FN_M at 73.01%. GraphSAGE shows slightly lower performance compared to GCN across all partitioning strategies, with Louvain partitioning achieving the best results at 72.76% for GraphSAGE+FN. Interestingly, the Girvan-Newman algorithm consistently times out on this dataset, indicating scalability issues with larger graphs such as ogbn-arxiv. The standard deviations are generally smaller for GraphSAGE compared to GCN, suggesting more stable performance across different random seeds. These results further support that METIS remains competitive.

Table. 5-14: Comparison of different graph partitioning methods for GINE with FN and FN_M architectures on Peptides-func/struct and molecular property prediction tasks. Best results for each metric are shown in **bold**.

Method	Partitioning	Peptides-func	Peptides-struct	MolHIV	MolTox21
		AP ↑	MAE ↓	ROCAUC ↑	ROCAUC ↑
GINE + FN	METIS	**0.6815±0.0059**	**0.2515±0.0020**	0.7882±0.0050	**0.7751±0.0029**
	Random	0.6533±0.0103	0.2688±0.0014	**0.8039±0.0078**	0.7653±0.0065
	Louvain	0.6044±0.0068	0.2799±0.0015	0.7844±0.0050	0.7701±0.0026
	Girvan-Newman	0.6528±0.0051	0.2628±0.0045	0.7837±0.0078	0.7630±0.0060
GINE + FN_M	METIS	**0.7018±0.0074**	**0.2446±0.0018**	**0.8127±0.0076**	**0.7926±0.0021**
	Random	0.6680±0.0066	0.2538±0.0013	0.8090±0.0061	0.7867±0.0045
	Louvain	0.6164±0.0120	0.2789±0.0022	0.7629±0.0164	0.7510±0.0118
	Girvan-Newman	0.6514±0.0064	0.2655±0.0037	0.7763±0.0174	0.7579±0.0097

Table 5-16 demonstrates the empirical runtime performance of different graph partitioning algorithms across various graph-level tasks, providing evidence for the practicality of our approach. While all algorithms show comparable performance on smaller datasets like Peptides (with runtimes in microseconds), noticeable differences emerge starting with medium-sized datasets like MNIST.

The distinction becomes particularly pronounced on large-scale datasets like ogbn-arxiv. We opt for METIS as our default partitioning algorithm due to its theoretical time complexity

Table. 5-15: Comparison of different graph partitioning methods for GCN/GraphSAGE with FN and FN_M on ogbn-arxiv dataset. Results show accuracy (%) and best results for each metric are shown in **bold**.

ogbn-arxiv	GCN + FN	GCN + FN_M	GraphSAGE + FN	GraphSAGE + FN_M
METIS	**73.03±0.37**	72.93±0.35	72.70±0.11	72.54±0.30
Random	72.79±0.37	**73.01±0.41**	72.46±0.20	72.46±0.27
Louvain	72.73±0.57	72.95±0.26	**72.76±0.15**	**72.56±0.58**
Girvan-Newman	Time-out	Time-out	Time-out	Time-out

of $O(|E|)$ and superior empirical performance. METIS efficiently partitions large graphs such as ogbn-arxiv in under 9 seconds, and even handles massive graphs like ogbn-products around 15 minutes.

In contrast, the Louvain algorithm requires over 50 seconds for ogbn-arxiv, while the Girvan-Newman algorithm encounters runtime limitations, making it impractical for large-scale graphs like ogbn-arxiv and ogbn-products. These results validate our choice of METIS as the primary partitioning algorithm, as it provides an effective balance between computational efficiency and partition quality across different graph scales.

Table. 5-16: Empirical runtime of partitioning algorithms.

Algorithm	Peptides-func/struct	MNIST	MolHIV	ogbn-arxiv	ogbn-product
METIS	0.71 μs	0.36 s	0.71μs	8.57 s	923.27 s
Louvain	1.19 μs	0.36 s	1.19μs	52.12s	119 m
Girvan-Newman	1.19 μs	0.36 s	0.72μs	Time-out	Time-out

Chapter 6

Conclusion

In this thesis, we studied two of the most fundamental challenges in deep learning on graphs — over-smoothing and over-squashing — and proposed a set of principled solutions grounded in theoretical insights and inspired by physics and multi-scale systems. Specifically, we introduced GREAD, a reaction-diffusion-based GNN architecture that maintains discriminative node representations in deep networks, and GFSA, a novel graph filter-based self-attention mechanism that extends these benefits to Transformer models. On the other hand, to address over-squashing, we proposed PANDA, a width-aware message passing strategy, and Fractal Nodes, a mechanism for long-range communication inspired by naturally occurring hierarchical structures.

Through extensive experiments on various homophilic and heterophilic benchmarks, we have demonstrated that our models not only outperform existing baselines but also provide deeper insight into the dynamics of signal propagation in graph networks. Our methods strike a practical balance between scalability and expressiveness — qualities often seen as mutually exclusive in graph learning models.

However, several limitations remain. Although our models scale linearly in complexity, they still require careful tuning of hyperparameters, particularly in determining reaction coefficients and integration steps in continuous GNN models, such as GREAD. Additionally, the behavior of our frameworks on dynamic, time-evolving graphs or under adversarial perturbation warrants further investigation.

Taking a step back, this thesis highlights a broader vision: that graph-structured data is not only a technical representation but a lens through which we can understand, predict, and ultimately shape the world. From protein interaction networks to social dynamics, from road traffic to knowledge graphs, the structures we study reflect real-world complexity. Deep learning on graphs, when properly guided, becomes not just a tool for modeling but a method for reasoning about relational structure.

Looking forward, we identify the following promising directions:

- Foundation Models for Graphs: Inspired by the success of large language models (LLMs),

future work could explore the development of pre-trained foundation models for graphs. Such models would be trained across diverse graph modalities and tasks, and then adapted via fine-tuning or prompting to downstream applications — analogous to GPT-style models in NLP. Incorporating graph-specific inductive biases (e.g., permutation invariance, hierarchy, multi-hop relations) into scalable pretraining frameworks remains a critical challenge and opportunity.

- Graph-Language Integration: Another frontier lies in bridging structured graph data with unstructured language, enabling joint models that understand, reason over, and generate graph-structured knowledge. This could open the door to novel applications in scientific discovery, knowledge base completion, and explainable AI.

For reproducibility and the benefit of the community, we make the algorithms, datasets, and additional resources used throughout this thesis available at `www.jeongwhanchoi.com`.

To conclude, this thesis not only contributes specific algorithmic improvements to the field of graph representation learning but also provides a conceptual foundation for how we might rethink depth, structure, and information flow in graph-based systems.

References

[1] Xiaojun Guo et al. “ContraNorm: A contrastive learning perspective on oversmoothing and beyond”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2023 (cit. on pp. iii, 29, 32, 40, 46, 52, 57).

[2] Jie Tang et al. “Social influence analysis in large-scale networks”. In: *KDD*. 2009, pp. 807–816 (cit. on pp. 1, 14).

[3] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272 (cit. on p. 1).

[4] Xiangnan He et al. “LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation”. In: *SIGIR*. 2020 (cit. on p. 1).

[5] Jeongwhan Choi et al. “LT-OCF: Learnable-Time ODE-based Collaborative Filtering”. In: *CIKM*. 2021 (cit. on pp. 1, 9, 11, 72).

[6] Yaguang Li et al. “Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting”. In: *ICLR*. 2018 (cit. on p. 1).

[7] Jeongwhan Choi et al. “Graph Neural Controlled Differential Equations for Traffic Forecasting”. In: *AAAI*. 2022 (cit. on pp. 1, 5).

[8] Kenta Oono and Taiji Suzuki. “Graph neural networks exponentially lose expressive power for node classification”. In: *ICLR*. 2020 (cit. on pp. 1, 8, 17, 30, 33, 77).

[9] Chen Cai and Yusu Wang. “A note on over-smoothing for graph neural networks”. In: *arXiv preprint arXiv:2006.13318* (2020) (cit. on pp. 1, 30).

[10] Uri Alon and Eran Yahav. “On the Bottleneck of Graph Neural Networks and its Practical Implications”. In: *ICLR*. 2021. URL: https://openreview.net/forum?id=i80OPhOCVH2 (cit. on pp. 2, 72, 73, 76, 85, 90, 98, 100, 101, 109, 123).

[11] Johannes Gasteiger et al. “Diffusion improves graph learning”. In: *Advances in neural information processing systems*. 2019 (cit. on pp. 2, 85, 90, 94, 96, 101, 130).

[12] Kedar Karhadkar et al. “FoSR: First-order spectral rewiring for addressing oversquashing in GNNs”. In: *ICLR*. 2023. URL: https://openreview.net/forum?id=3YjQfCLdrzz (cit. on pp. 2, 73, 77, 85, 86, 91, 94, 96, 101, 114, 130).

[13] Khang Nguyen et al. “Revisiting Over-smoothing and Over-squashing Using Ollivier-Ricci Curvature”. In: *ICML*. 2023. URL: https://proceedings.mlr.press/v202/nguyen23c.html (cit. on pp. 2, 91, 94, 96, 97, 101, 114).

[14] Francesco Di Giovanni et al. “On Over-Squashing in Message Passing Neural Networks: The Impact of Width, Depth, and Topology”. In: *ICML*. 2023. URL: https://proceedings.mlr.press/v202/di-giovanni23a.html (cit. on pp. 2, 74, 76–78, 83, 91, 101, 108).

[15] Jeongwhan Choi et al. “GREAD: Graph neural reaction-diffusion networks”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2023, pp. 5722–5747 (cit. on pp. 5, 37, 72).

[16] Federico Monti et al. “Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs”. In: *CVPR*. 2017, pp. 5425–5434 (cit. on p. 5).

[17] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *ICML*. 2017 (cit. on pp. 5, 98, 101, 102, 108).

[18] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *KDD*. 2018 (cit. on p. 5).

[19] Jeongwhan Choi et al. “Blurring-Sharpening Process Models for Collaborative Filtering”. In: *SIGIR*. 2023 (cit. on p. 5).

[20] Thomas Gaudelet et al. “Utilizing graph machine learning within drug discovery and development”. In: *Briefings in bioinformatics* 22.6 (2021), bbab159 (cit. on p. 5).

[21] Yeon Uk Jeong et al. “Predicting drug-drug interactions: a deep learning approach with GCN-based collaborative filtering”. In: *Artificial Intelligence in Medicine* (2025), p. 103185 (cit. on p. 5).

[22] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *ICLR*. 2017 (cit. on pp. 5, 8, 9, 11, 15, 33, 37, 76, 80, 104, 111, 127).

[23] Petar Veličković et al. “Graph Attention Networks”. In: *ICLR*. 2018 (cit. on pp. 5, 8, 15, 33, 72, 76, 98).

[24] Michaël Defferrard et al. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *NeurIPS*. 2016 (cit. on pp. 5, 15, 37, 72, 98).

[25] Felix Wu et al. “Simplifying Graph Convolutional Networks”. In: *ICML*. 2019 (cit. on pp. 5, 8, 11, 15).

[26] Ming Chen et al. “Simple and Deep Graph Convolutional Networks”. In: *ICML*. 2020 (cit. on pp. 5, 8, 11, 15, 72, 98).

[27] Eli Chien et al. “Adaptive Universal Generalized PageRank Graph Neural Network”. In: *ICLR*. 2021 (cit. on pp. 5, 6, 8, 11, 15, 37, 72).

[28] Jiong Zhu et al. “Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs”. In: *NeurIPS*. 2020 (cit. on pp. 5, 8, 11, 15, 22).

[29] Yujun Yan et al. “Two Sides of the Same Coin: Heterophily and Oversmoothing in Graph Convolutional Neural Networks”. In: *ICDM*. 2022 (cit. on pp. 5, 15).

[30] Derek Lim et al. “Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods”. In: *NeurIPS*. Vol. 34. Curran Associates, Inc., 2021, pp. 20887–20902 (cit. on pp. 5, 8, 15, 112, 127).

[31] Xiang Li et al. “Finding Global Homophily in Graph Neural Networks When Meeting Heterophily”. In: *ICML*. Ed. by Kamalika Chaudhuri et al. Vol. 162. 2022, pp. 13242–13256 (cit. on pp. 5, 8, 15).

[32] Sitao Luan et al. “Revisiting Heterophily For Graph Neural Networks”. In: *NeurIPS*. 2022 (cit. on pp. 5, 6, 8, 15).

[33] T. Konstantin Rusch et al. “Graph-Coupled Oscillator Networks”. In: *ICML*. Vol. 162. 2022, pp. 18888–18909 (cit. on pp. 5, 8, 15, 17).

[34] Benjamin Paul Chamberlain et al. “GRAND: Graph Neural Diffusion”. In: *ICML*. 2021 (cit. on pp. 5, 9, 11, 15, 72, 98).

[35] Benjamin Paul Chamberlain et al. “Beltrami Flow and Neural Diffusion on Graphs”. In: *NeurIPS*. 2021 (cit. on pp. 5, 9, 15).

[36] Cristian Bodnar et al. “Neural Sheaf Diffusion: A Topological Perspective on Heterophily and Oversmoothing in GNNs”. In: *NeurIPS*. 2022 (cit. on pp. 5, 8, 15).

[37] Deyu Bo et al. “Beyond Low-frequency Information in Graph Convolutional Networks”. In: *AAAI*. 2021 (cit. on pp. 6, 11, 15).

[38] Alan Turing. “The chemical basis of morphogenesis”. In: *Phil. Trans. R. Soc. Lond. B* (1952) (cit. on p. 6).

[39] Louis-Pascal A. C. Xhonneux et al. “Continuous Graph Neural Networks”. In: *International Conference on Machine Learning (ICML)*. 2020 (cit. on pp. 6, 11, 15).

[40] Matthew Thorpe et al. “GRAND++: Graph Neural Diffusion with A Source Term”. In: *ICLR*. 2022 (cit. on pp. 6, 9).

[41] Will Hamilton et al. “Inductive representation learning on large graphs”. In: *Advances in Neural Information Processing Systems*. 2017 (cit. on pp. 8, 11, 15, 127).

[42] Hao Zhu and Piotr Koniusz. “Simple spectral graph convolution”. In: *ICLR*. 2020 (cit. on p. 8).

[43] Muhammet Balcilar et al. “Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective”. In: *ICLR*. 2021 (cit. on p. 8).

[44] Matteo Tiezzi et al. “Deep constraint-based propagation in graph neural networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.2 (2021), pp. 727–739 (cit. on p. 8).

[45] Qimai Li et al. “Deeper insights into graph convolutional networks for semi-supervised learning”. In: *AAAI*. 2018 (cit. on pp. 8, 77).

[46] Hongbin Pei et al. “Geom-gcn: Geometric graph convolutional networks”. In: *ICLR*. 2020 (cit. on pp. 8, 14, 15, 94).

[47] Sami Abu-El-Haija et al. “Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing”. In: *ICML*. 2019, pp. 21–29 (cit. on pp. 8, 15).

[48] Mingguo He et al. “BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation”. In: *NeurIPS*. 2021 (cit. on p. 8).

[49] Francesco Di Giovanni et al. “Graph neural networks as gradient flows”. In: *arXiv preprint arXiv:2206.10991* (2022) (cit. on pp. 8, 15).

[50] Keyulu Xu et al. “Representation learning on graphs with jumping knowledge networks”. In: *ICML*. 2018, pp. 5453–5462 (cit. on pp. 8, 15, 37).

[51] Lingxiao Zhao and Leman Akoglu. “PairNorm: Tackling Oversmoothing in GNNs”. In: *ICLR*. 2020 (cit. on pp. 8, 11, 15).

[52] Mark I Freidlin and Alexander D Wentzell. “Diffusion processes on graphs and the averaging principle”. In: *The Annals of probability* (1993), pp. 2215–2245 (cit. on p. 8).

[53] Mark Freidlin and Shuenn-Jyi Sheu. “Diffusion processes on graphs: stochastic differential equations, large deviation principle”. In: *Probability theory and related fields* 116.2 (2000), pp. 181–220 (cit. on p. 8).

[54] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396 (cit. on p. 8).

[55] Ronald R Coifman et al. “Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps”. In: *Proceedings of the national academy of sciences* 102.21 (2005), pp. 7426–7431 (cit. on p. 8).

[56] Xavier Desquesnes et al. “Eikonal equation adaptation on weighted graphs: fast geometric diffusion process for local and non-local image and data processing”. In: *Journal of Mathematical Imaging and Vision* 46.2 (2013), pp. 238–257 (cit. on p. 8).

[57] Abderrahim Elmoataz et al. “Nonlocal discrete regularization on weighted graphs: a framework for image and manifold processing”. In: *IEEE transactions on Image Processing* 17.7 (2008), pp. 1047–1060 (cit. on p. 8).

[58] Guy Gilboa and Stanley Osher. "Nonlocal operators with applications to image processing". In: *Multiscale Modeling & Simulation* 7.3 (2009), pp. 1005–1028 (cit. on p. 8).

[59] Yifei Wang et al. "Dissecting the Diffusion Process in Linear Graph Convolutional Networks". In: *NeurIPS*. 2021 (cit. on pp. 9, 11).

[60] Jeehyun Hwang et al. "Climate Modeling with Neural Diffusion Equations". In: *ICDM*. 2021, pp. 230–239 (cit. on pp. 9, 72).

[61] Hwangyong Choi et al. "Climate modeling with neural advection–diffusion equation". In: *Knowledge and Information Systems* (2023), pp. 1–25 (cit. on p. 9).

[62] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *NeurIPS*. 2018 (cit. on p. 9).

[63] J.R. Dormand and P.J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26 (cit. on pp. 9, 11).

[64] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems (NeurIPS)*. Vol. 30. 2017 (cit. on pp. 10, 29, 32, 71, 98).

[65] Jie Chen et al. "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling". In: *ICLR*. 2018 (cit. on p. 11).

[66] Guohao Li et al. "DeepGCNs: Can GCNs Go As Deep As CNNs?" In: *ICCV*. 2019 (cit. on p. 11).

[67] Meng Liu et al. "Towards Deeper Graph Neural Networks". In: *KDD*. 2020, pp. 338–348 (cit. on p. 11).

[68] Wen-bing Huang et al. "Adaptive Sampling Towards Fast Graph Representation Learning". In: *NeurIPS*. 2018 (cit. on p. 11).

[69] Jianfei Chen et al. "Stochastic Training of Graph Convolutional Networks with Variance Reduction". In: *ICML*. 2018 (cit. on p. 11).

[70] Ronald Aylmer Fisher. "The wave of advance of advantageous genes". In: *Annals of eugenics* 7.4 (1937), pp. 355–369 (cit. on p. 11).

[71] Samuel M Allen and John W Cahn. "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening". In: *Acta metallurgica* 27.6 (1979), pp. 1085–1095 (cit. on p. 11).

[72] Brian H Gilding and Robert Kersner. *Travelling waves in nonlinear diffusion-convection reaction*. Vol. 60. Springer Science & Business Media, 2004 (cit. on p. 11).

[73] Benedek Rozemberczki et al. "Multi-Scale Attributed Node Embedding". In: *Journal of Complex Networks* 9.2 (2021) (cit. on pp. 14, 94).

[74] Andrew Kachites McCallum et al. “Automating the construction of internet portals with machine learning”. In: *Information Retrieval* 3.2 (2000), pp. 127–163 (cit. on p. 14).

[75] Prithviraj Sen et al. “Collective Classification in Network Data”. In: *AI Magazine* 29.3 (Sept. 2008), p. 93 (cit. on p. 14).

[76] Zhilin Yang et al. “Revisiting Semi-Supervised Learning with Graph Embeddings”. In: *ICML*. 2016 (cit. on pp. 14, 94).

[77] Susheel Suresh et al. “Breaking the Limit of Graph Neural Networks by Improving the Assortativity of Graphs with Local Mixing Patterns”. In: *KDD*. 2021, pp. 1541–1551 (cit. on p. 15).

[78] Michael Poli et al. “Graph neural ordinary differential equations”. In: *arXiv preprint arXiv:1911.07532* (2019) (cit. on p. 15).

[79] Yuelin Wang et al. “ACMP: Allen-Cahn Message Passing for Graph Neural Networks with Particle Phase Transition”. In: *ICLR*. 2023 (cit. on p. 15).

[80] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: `https://www.wandb.com/` (cit. on p. 15).

[81] Hoang Nt and Takanori Maehara. “Revisiting graph neural networks: All we have is low-pass filters”. In: *arXiv preprint arXiv:1905.09550* (2019) (cit. on pp. 17, 77, 98).

[82] T. Konstantin Rusch et al. “A Survey on Oversmoothing in Graph Neural Networks”. In: *arXiv preprint arXiv: Arxiv-2303.10993* (2023) (cit. on pp. 17, 30, 33).

[83] Yash Deshpande et al. “Contextual Stochastic Block Models”. In: *NeurIPS*. 2018 (cit. on pp. 20, 21).

[84] Yaxin Li et al. “Deeprobust: a platform for adversarial attacks and defenses”. In: *AAAI*. 2021, pp. 16078–16080 (cit. on p. 22).

[85] Jeongwhan Choi et al. “Graph Convolutions Enrich the Self-Attention in Transformers!” In: *NeurIPS*. 2024 (cit. on p. 29).

[86] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. URL: `https://aclanthology.org/N19-1423` (cit. on pp. 29, 38, 52, 53, 66, 68, 70).

[87] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018) (cit. on p. 29).

[88] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on pp. 29, 39, 54, 55, 66, 68, 70).

[89] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021 (cit. on pp. 29, 40, 101).

[90] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2021, pp. 10347–10357 (cit. on pp. 29, 34, 39, 40, 55, 57).

[91] Daquan Zhou et al. “DeepViT: Towards deeper vision transformer”. In: *arXiv preprint arXiv:2103.11886* (2021) (cit. on pp. 29, 30, 34, 40).

[92] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022 (cit. on pp. 29, 39, 40, 55).

[93] Anmol Gulati et al. “Conformer: Convolution-augmented transformer for speech recognition”. In: *arXiv preprint arXiv:2005.08100* (2020) (cit. on p. 29).

[94] Siddique Latif et al. “Transformers in speech processing: A survey”. In: *arXiv preprint arXiv:2303.11607* (2023) (cit. on p. 29).

[95] Ladislav Rampášek et al. “Recipe for a general, powerful, scalable graph transformer”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 14501–14515 (cit. on pp. 29, 41, 62).

[96] Luis Müller et al. “Attending to Graph Transformers”. In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: `https://openreview.net/forum?id=HhbqHBBrfZ` (cit. on p. 29).

[97] Yifan Peng et al. “Branchformer: Parallel mlp-attention architectures to capture local and global context for speech recognition and understanding”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2022, pp. 17627–17643 (cit. on pp. 29, 41, 59, 66, 69).

[98] Jayoung Kim et al. “Polynomial-based Self-Attention for Table Representation Learning”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. PMLR, 2024, pp. 24509–24526 (cit. on p. 29).

[99] Youn-Yeol Yu et al. “Learning Flexible Body Collision Dynamics with Hierarchical Contact Mesh Transformer”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=90yw2uM6J5` (cit. on p. 29).

[100] Yehjin Shin et al. “An attentive inductive bias for sequential recommendation beyond the self-attention”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 8. 2024, pp. 8984–8992 (cit. on p. 29).

[101] Chengyue Gong et al. “Vision transformers with patch diversification”. In: *arXiv preprint arXiv:2104.12753* (2021) (cit. on pp. 29–31, 34, 40).

[102] Ameen Ali et al. “Centered Self-Attention Layers”. In: *arXiv preprint arXiv: 2306.01610* (2023) (cit. on pp. 29, 30).

[103] Peihao Wang et al. “Anti-Oversmoothing in Deep Vision Transformers via the Fourier Domain Analysis: From Theory to Practice”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2022 (cit. on pp. 29–31, 33, 34, 37, 40, 46, 55).

[104] Yihe Dong et al. “Attention is not all you need: Pure attention loses rank doubly exponentially with depth”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2021, pp. 2793–2803 (cit. on pp. 29, 34).

[105] Han Shi et al. “Revisiting over-smoothing in BERT from the perspective of graph”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2022 (cit. on pp. 29, 30, 32, 33, 37).

[106] Guangtao Wang et al. “Multi-hop attention graph neural network”. In: *IJCAI*. 2021 (cit. on p. 30).

[107] Nicolas Keriven. “Not too little, not too much: a theoretical analysis of graph (over) smoothing”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 2268–2281 (cit. on p. 30).

[108] Hanqi Yan et al. “Addressing token uniformity in transformers via singular value transformation”. In: *Uncertainty in Artificial Intelligence*. PMLR. 2022, pp. 2181–2191 (cit. on pp. 30, 34).

[109] Lorenzo Noci et al. “Signal propagation in transformers: Theoretical perspectives and the role of rank collapse”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 27198–27211 (cit. on pp. 30, 34).

[110] Xinyi Wu et al. “Demystifying Oversmoothing in Attention-Based Graph Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023 (cit. on p. 30).

[111] Xinyi Wu et al. “A Non-Asymptotic Analysis of Oversmoothing in Graph Neural Networks”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2023 (cit. on p. 30).

[112] Jiawang Bai et al. “Improving vision transformers by revisiting high-frequency components”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 1–18 (cit. on pp. 31, 34, 37, 39, 40).

[113] Sohir Maskey et al. “A Fractional Graph Laplacian Approach to Oversmoothing”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023 (cit. on pp. 32, 33, 48).

[114] Aliaksei Sandryhaila and José MF Moura. "Discrete signal processing on graphs". In: *IEEE transactions on signal processing* 61.7 (2013), pp. 1644–1656 (cit. on pp. 33, 47).

[115] Aliaksei Sandryhaila and Jose MF Moura. "Discrete signal processing on graphs: Frequency analysis". In: *IEEE Transactions on Signal Processing* 62.12 (2014), pp. 3042–3054 (cit. on pp. 33, 47).

[116] Antonio G. Marques et al. "Signal Processing on Directed Graphs: The Role of Edge Directionality When Processing and Learning From Network Data". In: *IEEE Signal Processing Magazine* 37.6 (2020), pp. 99–116. DOI: 10.1109/MSP.2020.3014597 (cit. on p. 33).

[117] Chunya Zou et al. "A simple yet effective SVD-GCN for directed graphs". In: *arXiv preprint arXiv:2205.09335* (2022) (cit. on pp. 33, 48).

[118] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI open* 1 (2020), pp. 57–81 (cit. on p. 33).

[119] Daquan Zhou et al. "Refiner: Refining self-attention for vision transformers". In: *arXiv preprint arXiv:2106.03714* (2021) (cit. on pp. 34, 40).

[120] Aston Zhang et al. "On Orthogonality Constraints for Transformers". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 2021, pp. 375–382 (cit. on p. 34).

[121] Shuangfei Zhai et al. "Stabilizing Transformer Training by Preventing Attention Entropy Collapse". In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. Vol. 202. PMLR, 2023, pp. 40770–40803 (cit. on p. 34).

[122] Nuo Chen et al. "Alleviating Over-smoothing for Unsupervised Sentence Representation". In: *arXiv preprint arXiv:2305.06154* (2023) (cit. on p. 34).

[123] Gbètondji JS Dovonon et al. "Setting the Record Straight on Transformer Oversmoothing". In: *arXiv preprint arXiv:2401.04301* (2024) (cit. on p. 34).

[124] Nicholas J Higham. *Functions of matrices: theory and computation*. SIAM, 2008 (cit. on p. 35).

[125] Edward De Brouwer et al. "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019 (cit. on p. 35).

[126] Haoyi Zhou et al. "Jump Self-attention: Capturing High-order Statistics in Transformers". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 17899–17910 (cit. on p. 37).

[127] Johannes Gasteiger et al. "Diffusion improves graph learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. 2019 (cit. on p. 37).

[128] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019) (cit. on pp. 38, 53, 66, 68).

[129] Yinhan Liu et al. “RoBERTa: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019) (cit. on pp. 38, 43, 53, 62, 66–69).

[130] Mitchell Marcus et al. “Building a large annotated corpus of English: The Penn Treebank”. In: (1993) (cit. on pp. 39, 54).

[131] Stephen Merity et al. “Pointer sentinel mixture models”. In: *arXiv preprint arXiv:1609.07843* (2016) (cit. on pp. 39, 54).

[132] Zhewei Yao et al. “Random-ltd: Random and layerwise token dropping brings efficient training for large-scale transformers”. In: *arXiv preprint arXiv:2211.11586* (2022) (cit. on pp. 39, 54).

[133] Hugo Touvron et al. “Going deeper with image transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 32–42 (cit. on pp. 39, 40, 55).

[134] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 40).

[135] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708 (cit. on p. 40).

[136] Haiping Wu et al. “CvT: Introducing convolutions to vision transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 22–31 (cit. on p. 40).

[137] Tianlong Chen et al. “The principle of diversity: Training stronger vision transformers calls for reducing all levels of redundancy”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12020–12030 (cit. on p. 40).

[138] Li Yuan et al. “Tokens-to-token vit: Training vision transformers from scratch on imagenet”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 558–567 (cit. on p. 40).

[139] Vijay Prakash Dwivedi et al. “Long Range Graph Benchmark”. In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022. URL: https://openreview.net/forum?id=in7XC5RcjEn (cit. on pp. 40, 62, 84, 96, 97, 110, 111, 123, 130).

[140] Vijay Prakash Dwivedi et al. “Benchmarking graph neural networks”. In: *Journal of Machine Learning Research* 24.43 (2023), pp. 1–48 (cit. on pp. 41, 62, 110, 111).

[141] Weihua Hu et al. “Open graph benchmark: Datasets for machine learning on graphs”. In: *Advances in neural information processing systems (NeurIPS)* 33 (2020), pp. 22118–22133 (cit. on pp. 41, 62, 101, 110, 111, 123, 130, 132).

[142] Weihua Hu et al. “OGB-LSC: A large-scale challenge for machine learning on graphs”. In: *arXiv preprint arXiv:2103.09430* (2021) (cit. on pp. 41, 61, 110, 126, 127).

[143] Chengxuan Ying et al. “Do transformers really perform badly for graph representation?” In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021, pp. 28877–28888 (cit. on pp. 41, 61, 66, 68, 101, 111).

[144] Xiaoxin He et al. “A generalization of vit/mlp-mixer to graphs”. In: *International conference on machine learning (ICML)*. PMLR. 2023, pp. 12724–12745 (cit. on pp. 41, 42, 62, 101, 107, 111, 124).

[145] Vassil Panayotov et al. “Librispeech: an asr corpus based on public domain audio books”. In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 5206–5210 (cit. on pp. 41, 58).

[146] Mirco Ravanelli et al. *SpeechBrain: A General-Purpose Speech Toolkit*. arXiv:2106.04624. 2021. arXiv: `2106 . 04624 [eess.AS]` (cit. on pp. 41, 58, 59).

[147] Yaqin Zhou et al. “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks”. In: *Advances in neural information processing systems (NeurIPS)*. Vol. 32. 2019 (cit. on pp. 43, 62).

[148] Zhangyin Feng et al. “CodeBERT: A Pre-Trained Model for Programming and Natural Languages”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2020, pp. 1536–1547 (cit. on pp. 43, 63, 67, 69).

[149] Wasi Ahmad et al. “Unified Pre-training for Program Understanding and Generation”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2021, pp. 2655–2668 (cit. on pp. 43, 62, 67, 69).

[150] Yue Wang et al. “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021, pp. 8696–8708 (cit. on pp. 43, 62–65, 67, 69).

[151] Angelos Katharopoulos et al. “Transformers are rnns: Fast autoregressive transformers with linear attention”. In: *International conference on machine learning*. PMLR. 2020, pp. 5156–5165 (cit. on p. 44).

[152] Zhuoran Shen et al. “Efficient attention: Attention with linear complexities”. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 3531–3539 (cit. on pp. 44, 45, 71).

[153] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023) (cit. on p. 45).

[154] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023) (cit. on p. 45).

[155] George Dasoulas et al. “Lipschitz normalization for self-attention layers with application to graph neural networks”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2021, pp. 2456–2466 (cit. on p. 50).

[156] Alex Warstadt et al. “Neural network acceptability judgments”. In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 625–641 (cit. on p. 52).

[157] Richard Socher et al. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642 (cit. on p. 52).

[158] Bill Dolan and Chris Brockett. “Automatically constructing a corpus of sentential paraphrases”. In: *Third International Workshop on Paraphrasing (IWP2005)*. 2005 (cit. on p. 52).

[159] Zihan Chen et al. *Quora question pairs*. 2018 (cit. on p. 52).

[160] Daniel Cer et al. “Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation”. In: *arXiv preprint arXiv:1708.00055* (2017) (cit. on p. 53).

[161] Adina Williams et al. “A broad-coverage challenge corpus for sentence understanding through inference”. In: *arXiv preprint arXiv:1704.05426* (2017) (cit. on p. 53).

[162] Wei Wang et al. “Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering”. In: *arXiv preprint arXiv:1811.11934* (2018) (cit. on p. 53).

[163] Luisa Bentivogli et al. “The Fifth PASCAL Recognizing Textual Entailment Challenge.” In: *TAC* 7 (2009), p. 8 (cit. on p. 53).

[164] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017) (cit. on pp. 53, 54, 59, 61).

[165] Ross Wightman. *PyTorch Image Models*. `https://github.com/rwightman/pytorch-image-models`. 2019. DOI: `10.5281/zenodo.4414861` (cit. on p. 55).

[166] Badri N Patro et al. “SpectFormer: Frequency and Attention is what you need in a Vision Transformer”. In: *arXiv preprint arXiv:2304.06446* (2023) (cit. on p. 57).

[167] Badri Patro and Vijay Agneeswaran. “Scattering Vision Transformer: Spectral Mixing Matters”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023 (cit. on p. 57).

[168] Tam Nguyen et al. “Mitigating Over-smoothing in Transformers via Regularized Non-local Functionals”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. 2023 (cit. on p. 57).

[169] James Lee-Thorp et al. “Fnet: Mixing tokens with fourier transforms”. In: *arXiv preprint arXiv:2105.03824* (2021) (cit. on p. 57).

[170] Yongming Rao et al. “Global filter networks for image classification”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021, pp. 980–993 (cit. on p. 57).

[171] Daniel S Park et al. “SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition”. In: *Interspeech 2019* (2019) (cit. on p. 59).

[172] Alex Graves and Navdeep Jaitly. “Towards End-To-End Speech Recognition with Recurrent Neural Networks”. In: *International Conference on Machine Learning (ICML)*. PMLR, 2014, pp. 1764–1772 (cit. on p. 59).

[173] Ying Zhang et al. “Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks”. In: *Interspeech 2016* (2016) (cit. on p. 59).

[174] John J Irwin et al. “ZINC: a free tool to discover chemistry for biology”. In: *Journal of chemical information and modeling* 52.7 (2012), pp. 1757–1768 (cit. on p. 61).

[175] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7871–7880 (cit. on p. 62).

[176] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5485–5551 (cit. on p. 63).

[177] Wenhan Wang et al. “Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree”. In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020, pp. 261–271. DOI: 10.1109/SANER48275.2020.9054857 (cit. on p. 64).

[178] Sinong Wang et al. “Linformer: Self-attention with linear complexity”. In: *arXiv preprint arXiv:2006.04768* (2020) (cit. on p. 71).

[179] Zhanpeng Zeng et al. “You only sample (almost) once: Linear cost self-attention via bernoulli sampling”. In: *International conference on machine learning*. PMLR. 2021, pp. 12321–12332 (cit. on p. 71).

[180] Jeongwhan Choi et al. “PANDA: Expanded Width-Aware Message Passing Beyond Rewiring”. In: *ICML*. 2024 (cit. on pp. 72, 101, 114, 130).

[181] Jeongwhan Choi and Noseong Park. "Graph neural rough differential equations for traffic forecasting". In: *ACM Transactions on Intelligent Systems and Technology* 14.4 (2023), pp. 1–27 (cit. on p. 72).

[182] Qiyu Kang et al. "Node embedding from neural Hamiltonian orbits in graph neural networks". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 15786–15808 (cit. on p. 72).

[183] Anthony Gruber et al. "Reversible and irreversible bracket-based dynamics for deep graph neural networks". In: *Advances in Neural Information Processing Systems* 36 (2023) (cit. on p. 72).

[184] Dai Shi et al. "Exposition on over-squashing problem on GNNs: Current Methods, Benchmarks and Challenges". In: *arXiv preprint arXiv:2311.07073* (2023) (cit. on pp. 72, 76, 101).

[185] Jake Topping et al. "Understanding over-squashing and bottlenecks on graphs via curvature". In: *ICLR*. 2022. URL: https://openreview.net/forum?id=7UmjRGzp-A (cit. on pp. 73, 74, 76, 78, 85, 90, 91, 93, 94, 96, 114, 130).

[186] Pradeep Kr Banerjee et al. "Oversquashing in GNNs through the lens of information contraction and graph expansion". In: *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2022, pp. 1–8 (cit. on pp. 73, 91).

[187] Andreea Deac et al. "Expander graph propagation". In: *Learning on Graphs Conference*. PMLR. 2022, pp. 38–1 (cit. on pp. 73, 91).

[188] Adrián Arnaiz-Rodríguez et al. "DiffWire: Inductive Graph Rewiring via the Lovász Bound". In: *Learning on Graphs Conference*. 2022. URL: https://proceedings.mlr.press/v198/arnaiz-rodri-guez22a.html (cit. on pp. 73, 86, 92).

[189] Mitchell Black et al. "Understanding Oversquashing in GNNs through the Lens of Effective Resistance". In: *ICML*. 2023. URL: https://proceedings.mlr.press/v202/black23a.html (cit. on pp. 73, 77, 86, 91, 101, 114, 116, 117, 133, 134).

[190] Lukas Fesser and Melanie Weber. "Mitigating over-smoothing and over-squashing using augmentations of forman-ricci curvature". In: *arXiv preprint arXiv:2309.09384* (2023) (cit. on p. 73).

[191] Rishi Sonthalia et al. "RelWire: Metric Based Graph Rewiring". In: *Advances in Neural Information Processing Systems 2023 Workshop on Symmetry and Geometry in Neural Representations*. 2023. URL: https://openreview.net/forum?id=c9u8tH1WA0 (cit. on p. 73).

[192] Jhony H Giraldo et al. "On the trade-off between over-smoothing and over-squashing in deep graph neural networks". In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2023, pp. 566–576 (cit. on p. 73).

[193] Dai Shi et al. "How Curvature Enhance the Adaptation Power of Framelet GCNs". In: *arXiv preprint arXiv:2307.09768* (2023) (cit. on pp. 73, 91).

[194] Federico Barbero et al. "Locality-Aware Graph-Rewiring in GNNs". In: *arXiv preprint arXiv:2310.01668* (2023) (cit. on pp. 73, 91, 114, 130).

[195] Haiyuan Yu et al. "The importance of bottlenecks in protein networks: correlation with gene essentiality and expression dynamics". In: *PLoS computational biology* 3.4 (2007), e59 (cit. on p. 74).

[196] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=ryGs6iA5Km` (cit. on pp. 76, 80, 86, 101, 111).

[197] Arpita Ghosh et al. "Minimizing effective resistance of a graph". In: *SIAM review* 50.1 (2008), pp. 37–66 (cit. on pp. 77, 133, 134).

[198] Carsten Thomassen. "Resistances and currents in infinite electrical networks". In: *Journal of Combinatorial Theory, Series B* 49.1 (1990), pp. 87–102 (cit. on pp. 77, 133).

[199] Karel Devriendt and Renaud Lambiotte. "Discrete curvature on graphs from the effective resistance". In: *Journal of Physics: Complexity* 3.2 (2022), p. 025008 (cit. on pp. 77, 133).

[200] Linton C Freeman. "A set of measures of centrality based on betweenness". In: *Sociometry* (1977), pp. 35–41 (cit. on pp. 78, 93, 106).

[201] Stephen P Borgatti and Daniel S Halgin. "Analyzing affiliation networks". In: *The Sage handbook of social network analysis* 1 (2011), pp. 417–433 (cit. on pp. 78, 92).

[202] Linton C Freeman. "Centrality in social networks conceptual clarification". In: *Social networks* 1.3 (1978), pp. 215–239 (cit. on pp. 78, 93).

[203] Lawrence Page et al. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999 (cit. on pp. 78, 93).

[204] K-I Goh et al. "Universal behavior of load distribution in scale-free networks". In: *Physical review letters* 87.27 (2001), p. 278701 (cit. on pp. 78, 93).

[205] Emanuele Rossi et al. "Edge Directionality Improves Learning on Heterophilic Graphs". In: *arXiv preprint arXiv:2305.10498* (2023) (cit. on p. 81).

[206] Michael Schlichtkrull et al. "Modeling relational data with graph convolutional networks". In: *European Semantic Web Conference*. Springer. 2018, pp. 593–607 (cit. on pp. 81, 86).

[207] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learning with graphs". In: *ICML 2020 Workshop on Graph Representation Learning and*

Beyond (GRL+ 2020). 2020. arXiv: 2007.08663. URL: www.graphlearning.io (cit. on pp. 84, 85).

[208] Marc Brockschmidt. "Gnn-film: Graph neural networks with feature-wise linear modulation". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1144–1152 (cit. on p. 86).

[209] Benjamin Gutteridge et al. "Drew: Dynamically rewired message passing with delay". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 12252–12267 (cit. on p. 91).

[210] Ben Finkelshtein et al. "Cooperative graph neural networks". In: *arXiv preprint arXiv:2310.01267* (2023) (cit. on p. 91).

[211] Joshua Southern et al. "Understanding Virtual Nodes: Oversmoothing, Oversquashing, and Node Heterogeneity". In: *arXiv preprint arXiv:2405.13526* (2024) (cit. on p. 91).

[212] Federico Errica et al. "Adaptive Message Passing: A General Framework to Mitigate Oversmoothing, Oversquashing, and Underreaching". In: *arXiv preprint arXiv:2312.16560* (2023) (cit. on p. 91).

[213] Seonghyun Park et al. "Non-backtracking Graph Neural Networks". In: *arXiv preprint arXiv:2310.07430* (2023) (cit. on p. 91).

[214] Chendi Qian et al. "Probabilistically Rewired Message-Passing Neural Networks". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=Tj6Wcx7gVk (cit. on p. 91).

[215] Ali Behrouz and Farnoosh Hashemi. "Graph Mamba: Towards Learning on Graphs with State Space Models". In: *arXiv preprint arXiv:2402.08678* (2024) (cit. on p. 91).

[216] Haiquan Qiu et al. "Graph Unitary Message Passing". In: *arXiv preprint arXiv:2403.11199* (2024) (cit. on p. 91).

[217] F. C. Graham Fan R. K. Chung. "Spectral graph theory". In: *American Mathematical Society*. 92. 1997 (cit. on p. 92).

[218] Vijay Prakash Dwivedi and Xavier Bresson. "A Generalization of Transformer Networks to Graphs". In: *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021) (cit. on pp. 98, 101, 107, 111).

[219] Zhanghao Wu et al. "Representing long-range context for graph neural networks with global attention". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 13266–13279 (cit. on pp. 98, 101).

[220] Devin Kreuzer et al. "Rethinking graph transformers with spectral attention". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 21618–21629 (cit. on p. 98).

[221] Yujie Xing et al. “Less is More: on the Over-Globalizing Problem in Graph Transformers”. In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=uKmcyyrZae (cit. on p. 98).

[222] Ladislav Rampášek et al. “Recipe for a general, powerful, scalable graph transformer”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 14501–14515 (cit. on pp. 98, 101, 107, 111, 112, 124, 127).

[223] Benoit B Mandelbrot. “The fractal geometry of nature/Revised and enlarged edition”. In: *New York* (1983) (cit. on pp. 99, 102).

[224] Stephen Dill et al. “Self-similarity in the web”. In: *ACM Transactions on Internet Technology (TOIT)* 2.3 (2002), pp. 205–223 (cit. on p. 99).

[225] Pureun Kim and Byungnam Kahng. “Fractal Network in Protein Interaction Network Model”. In: *Journal of the Korean Physical Society* 56.3 (2010), pp. 1020–1024 (cit. on p. 99).

[226] Ying Chen et al. “On the capacity of fractal D2D social networks with hierarchical communications”. In: *IEEE Transactions on Mobile Computing* 20.6 (2020), pp. 2254–2268 (cit. on pp. 99, 102).

[227] Chaoming Song et al. “Self-similarity of complex networks”. In: *Nature* 433.7024 (2005), pp. 392–395 (cit. on pp. 99, 102).

[228] Ilya O Tolstikhin et al. “Mlp-mixer: An all-mlp architecture for vision”. In: *Advances in neural information processing systems* 34 (2021), pp. 24261–24272 (cit. on pp. 100, 104).

[229] Luis Müller et al. “Attending to graph transformers”. In: *arXiv preprint arXiv:2302.04181* (2023) (cit. on p. 101).

[230] Liheng Ma et al. “Graph inductive biases in transformers without message passing”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 23321–23337 (cit. on pp. 101, 111).

[231] Hamed Shirzad et al. “Exphormer: Sparse transformers for graphs”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 31613–31632 (cit. on pp. 101, 111, 112, 127).

[232] EunJeong Hwang et al. “An Analysis of Virtual Nodes in Graph Neural Networks for Link Prediction (Extended Abstract)”. In: *The First Learning on Graphs Conference*. 2022. URL: https://openreview.net/forum?id=dI6KBKNRp7 (cit. on pp. 102, 108).

[233] Chen Cai et al. “On the connection between mpnn and graph transformer”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 3408–3430 (cit. on pp. 102, 108).

[234] Jin Seop Kim et al. “Fractality and self-similarity in scale-free networks”. In: *New Journal of Physics* 9.6 (2007), p. 177 (cit. on p. 102).

[235] Agata Fronczak et al. “Scaling theory of fractal complex networks”. In: *Scientific Reports* 14.1 (2024), p. 9079 (cit. on p. 102).

[236] George Karypis and Vipin Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal on scientific Computing* 20.1 (1998), pp. 359–392 (cit. on pp. 103, 119).

[237] Maksim Kitsak et al. “Betweenness centrality of fractal and nonfractal scale-free model networks and tests on real networks”. In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 75.5 (2007), p. 056115 (cit. on p. 106).

[238] Lingxiao Zhao et al. “From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=Mspk_WYKoEH (cit. on pp. 107, 111).

[239] Bohang Zhang et al. “A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 41019–41077 (cit. on p. 107).

[240] Honghua Dong et al. “MeGraph: capturing long-range interactions by alternating local and hierarchical aggregation on multi-scaled graph hierarchy”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 63609–63641 (cit. on p. 107).

[241] Wei-Lin Chiang et al. “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 257–266 (cit. on p. 107).

[242] Zhiqiang Zhong et al. “Hierarchical message-passing graph neural networks”. In: *Data Mining and Knowledge Discovery* 37.1 (2023), pp. 381–408 (cit. on pp. 107, 112, 127).

[243] Han Gao et al. “Patchgt: Transformer over non-trainable clusters for learning graph representations”. In: *Learning on Graphs Conference*. PMLR. 2022, pp. 27–1 (cit. on p. 107).

[244] Wenhao Zhu et al. “Hierarchical transformer for scalable graph learning”. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 2023, pp. 4702–4710 (cit. on pp. 107, 112, 127).

[245] Weirui Kuang et al. *Coarformer: Transformer for large graph via graph coarsening*. 2022. URL: https://openreview.net/forum?id=fkjO_FKVzw (cit. on p. 107).

[246] Chen Cai et al. “Graph Coarsening with Neural Networks”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=uxpzitPEooJ (cit. on pp. 107, 112, 127).

[247] Ryan Murphy et al. “Relational pooling for graph representations”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4663–4673 (cit. on pp. 109, 123).

[248] Ralph Abboud et al. “The Surprising Power of Graph Neural Networks with Random Node Initialization”. In: *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*. 2021 (cit. on pp. 109, 123).

[249] Muhammet Balcilar et al. “Breaking the limits of message passing graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 599–608 (cit. on pp. 109, 123).

[250] Xavier Bresson and Thomas Laurent. “Residual gated graph convnets”. In: *arXiv preprint arXiv:1711.07553* (2017) (cit. on p. 111).

[251] Grégoire Mialon et al. “Graphit: Encoding graph structure in transformers”. In: *arXiv preprint arXiv:2106.05667* (2021) (cit. on p. 111).

[252] Devin Kreuzer et al. “Rethinking Graph Transformers with Spectral Attention”. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 21618–21629 (cit. on p. 111).

[253] Md Shamim Hussain et al. “Global self-attention as a replacement for graph convolution”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 655–665 (cit. on p. 111).

[254] Kaan Sancak et al. “A Scalable and Effective Alternative to Graph Transformers”. In: *arXiv preprint arXiv:2406.12059* (2024) (cit. on p. 111).

[255] Jan Tönshoff et al. “Walking Out of the Weisfeiler Leman Hierarchy: Graph Learning Beyond Message Passing”. In: *Transactions on Machine Learning Research* (2023). ISSN: 2835-8856. URL: https://openreview.net/forum?id=vgXnEyeWVY (cit. on p. 111).

[256] Gabriele Corso et al. “Principal neighbourhood aggregation for graph nets”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13260–13271 (cit. on p. 111).

[257] Fabrizio Frasca et al. “Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 31376–31390 (cit. on p. 111).

[258] Cristian Bodnar et al. “Weisfeiler and lehman go cellular: Cw networks”. In: *Advances in neural information processing systems* 34 (2021), pp. 2625–2640 (cit. on p. 111).

[259] Jinsong Chen et al. “NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=8KYeilT3Ow (cit. on pp. 112, 127).

[260] Qitian Wu et al. “Nodeformer: A scalable graph structure learning transformer for node classification”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27387–27401 (cit. on pp. 112, 127).

[261] Qitian Wu et al. "DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=j6zUzrapY3L (cit. on pp. 112, 127).

[262] Chenhui Deng et al. "Polynormer: Polynomial-Expressive Graph Transformer in Linear Time". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=hmv1LpNfXa (cit. on pp. 112, 127).

[263] Qitian Wu et al. "Simplifying and empowering transformers for large-graph representations". In: *Advances in Neural Information Processing Systems* 36 (2023) (cit. on pp. 112, 127).

[264] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008 (cit. on pp. 113, 142).

[265] Michelle Girvan and Mark EJ Newman. "Community structure in social and biological networks". In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826 (cit. on pp. 113, 142).

[266] *RDKit: Open-source cheminformatics*. http://www.rdkit.org (cit. on p. 123).

[267] Enikő Zakar-Polyák et al. "Towards a better understanding of the characteristics of fractal networks". In: *Applied Network Science* 8.1 (2023), p. 17 (cit. on p. 127).

[268] Paul Erdos, Alfréd Rényi, et al. "On the evolution of random graphs". In: *Publ. math. inst. hung. acad. sci* 5.1 (1960), pp. 17–60 (cit. on p. 128).

[269] Eran Rosenbluth et al. "Distinguished In Uniform: Self-Attention Vs. Virtual Nodes". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=AcSChDWL6V (cit. on pp. 130, 132).

[270] Zong-Wen Wei et al. "Renormalization and small-world model of fractal quantum repeater networks". In: *Scientific reports* 3.1 (2013), p. 1222 (cit. on p. 142).

[271] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Reviews of modern physics* 74.1 (2002), p. 47 (cit. on p. 142).

국문요약

그래프 기반 딥러닝 강화: 오버스무딩과 오버스쿼싱 문제 해결

최정환
인공지능학과
대학원
연세대학교

그래프는 전자상거래부터 분자 구조에 이르기까지 어디에나 존재하며, 개별 노드 간의 상호작용을 추상화합니다. 그래프 구조 데이터를 기반으로 하는 다양한 실제 애플리케이션은 그래프 노드의 효과적인 표현을 필요로 합니다. 최근 그래프에 대한 딥러닝은 다양한 휴리스틱 접근법을 통해 그래프 표현을 학습함으로써 다양한 분야에서 획기적인 발전을 이루었습니다. 그러나 그래프 딥러닝은 몇 가지 근본적인 과제에 직면합니다. 첫째, 그래프 신경망(GNN)은 연결된 이웃(neighborhood)을 통해 정보를 전파합니다. GNN 계층의 수를 늘리면 고차 이웃을 포착할 수 있지만, 이러한 반복적인 전파는 노드 표현이 이웃 간에 구분되지 않는 오버스무딩(over-smoothing) 문제를 야기합니다. 둘째, 실제 그래프는 종종 중요한 장거리 상호작용을 포함합니다. 노드는 오버스쿼싱(over-squashing) 문제로 인해 멀리 떨어진 노드로부터 정보를 수신하는 데 한계에 직면합니다. (over-squashing) 문제는 고정 차원 벡터가 멀리 떨어진 이웃으로부터 기하급수적으로 증가하는 정보의 병목 현상이 되는 문제입니다.

본 논문에서는 1) 오버스무딩(over-smoothing)와 2) 오버스쿼싱(over-squashing) 문제를 해결하기 위한 접근 방식들을 제안합니다. 먼저, 물리 관점에서 그래프 신경망을 열 확산 과정으로 바라보고 오버스무딩을 해결하기 위한 동기를 갖습니다. 반응-확산 방정식에서 영감을 받아 새로운 그래프 신경망 아키텍처를 제안하여 오버스무딩 문제를 완화합니다. 또한, 우리는 트랜스포머의 셀프 어텐션을 그래프로 해석하고 오버스무딩가 트랜스포머에도 있다는 것을 소개하고 이 문제를 완화하기 위한 새로운 자기 주의 메커니즘을 제안합니다. 오버스쿼싱의 경우, 기존의 재배선 방법들이 무분별하게 엣지를 추가하여 기존 그래프 구조를 손상하는 문제를 지적하며 시작합니다. 우리는 노드 너비 확장을 통해 오버스쿼싱 문제를 해결하는 새로운 그래프 신경망 아키텍처를 설계합니다. 또한, 우리는 프랙탈 개념에서 영감을 받아, 장거리 상호작용을 위해 오버스쿼싱을 완화하면서 더 전역적인 정보를 전파시킬 수 있도록하는 방법을 설계합니다.

전세계적으로 수집하는 데이터의 규모와 다양성이 증가함에 따라, 개별 노드 간의 관계는 규모와 복잡성 측면에서 2차적으로 증가합니다. 이 연구들은 그래프에서 더 나은 표현을 개발하고 딥러닝의 확장성을 향상시킴으로써 이러한 관계를 효과적으로 처리하고 광범위한 영역에 걸쳐 긍정적인 영향을 미치는 것을 목표로 합니다.

핵심 되는 말 : 그래프 머신러닝, 그래프 신경망, 오버스무딩, 오버스쿼싱