Large-Scale Spectral Graph Neural Networks via Laplacian Sparsification

Haipeng Ding
dinghaipeng@ruc.edu.cn
Renmin University of China
Beijing, China

Zhewei Wei*
zhewei@ruc.edu.cn
Renmin University of China
Beijing, China

Yuhang Ye
yeyuhang@huawei.com
Huawei Poisson Lab
Shenzhen, Guangdong, China

ABSTRACT

Graph Neural Networks (GNNs) play a pivotal role in graph-based tasks for their proficiency in representation learning. Among the various GNN methods, spectral GNNs employing polynomial filters have shown promising performance on tasks involving both homophilous and heterophilous graph structures. However, The scalability of spectral GNNs on large graphs is limited because they learn the polynomial coefficients through multiple forward propagation executions during forward propagation. Existing works have attempted to scale up spectral GNNs by eliminating the linear layers on the input node features, a change that can disrupt end-to-end training, potentially impact performance, and become impractical with high-dimensional input features. To address the above challenges, we propose "Spectral Graph Neural Networks with Laplacian Sparsification (SGNN-LS)", a novel graph spectral sparsification method to approximate the propagation patterns of spectral GNNs. We prove that our proposed method generates Laplacian sparsifiers that can approximate both fixed and learnable polynomial filters with theoretical guarantees. Our method allows the application of linear layers on the input node features, enabling end-to-end training as well as the handling of raw text features. We conduct an extensive experimental analysis on datasets spanning various graph scales and properties to demonstrate the superior efficiency and effectiveness of our method. The results show that our method yields superior results in comparison with the corresponding approximated base models, especially on dataset Ogbn-papers100M(111M nodes, 1.6B edges) and MAG-scholar-C (2.8M features).

CCS CONCEPTS

• **Computing methodologies** → **Spectral methods**; *Supervised learning*; • **Information systems** → *Data mining*.

KEYWORDS

Spectral Graph Neural Networks, Scalability, Laplacian sparsification

*Zhewei Wei is the corresponding author.

KDD '25, August 3–7, 2025, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/XXXXXX.XXXXXX

ACM Reference Format:
Haipeng Ding, Zhewei Wei, and Yuhang Ye. 2025. Large-Scale Spectral Graph Neural Networks via Laplacian Sparsification. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

Graph Neural Networks (GNNs) have gathered increasing research attention because of their versatility in handling graph-structured data. They have demonstrated prominent performance in several kinds of real-world graph learning tasks, including link prediction [38], recommendation systems [49, 53, 55], social analysis [30], drug discovery [25], and traffic forecasting [13]. GNNs can be broadly categorized into spatial GNNs, including GCN [26], GAT [46], GCNII [6], GIN [50], MGNN [12], and spectral GNNs.

Spectral GNNs represent one fundamental branch of GNNs that works by constructing a graph filter in the spectral domain of the graph Laplacian matrix. This filtering mechanism enables the recombination of graph signals at different frequencies, effectively leveraging their spectral properties. Constrained by the impractical overhead of eigendecomposition, various works adopt distinct polynomial bases to approximate the desired filter operation, such as GPR-GNN [11] leverages a monomial basis, BernNet [20] employs a Bernstein polynomial basis, JacobiConv [47] uses the Jacobi basis, and OptBasisGNN[18] learns optimal bases from the input signals.

In general, spectral GNNs employing polynomial filters can be formally expressed as $\mathbf{Y} = g_{\mathbf{w}}(\mathbf{L}, f_\theta(\mathbf{X}))$, where $g_{\mathbf{w}}(\cdot)$ denotes the polynomial graph filter with coefficients $\mathbf{w}$, $f_\theta(\cdot)$ represents the linear layer with learnable parameters θ, $\mathbf{L}$ is the Laplacian matrix, $\mathbf{X}$ and $\mathbf{Y}$ refer to the original node representation matrix and the output, respectively. Unlike GNNs that design a uniform aggregation function, spectral GNNs use polynomial filters to combine representations from K-hop neighbors, where K is the polynomial degree. This property enables spectral GNNs to capture a broader scope of graphs and alleviates the dependence on "homophily assumption".

Motivation. Recent advancements in scalable GNN architectures, particularly those employing subsampling-based methods, have been primarily designed to promote the scalability of spatial GNNs, especially the vanilla GCN [26]. These scalable methods are not compatible with spectral GNNs due to the requirement of computing $\mathbf{L}^k f_\theta(\mathbf{X})$, which demands k iterations of full graph propagation during training's forward propagation phase. Moreover, existing acceleration devices, such as GPUs, suffer bottlenecks in storing the computational trees and node representation for extensive graphs. Drawing inspirations from SGC [48], several spectral works such as ChebNetII [21] and OptBasis [18] apply a trick to detach the graph

propagation phase from the linear layers. This trick enables the precomputation of $\mathbf{L}^k\mathbf{X}$, and converts the model training into the linear combination of $\mathbf{L}^k\mathbf{X}$ using learnable coefficient w_k and linear layers. However, this trick has lots of potential defects, including disrupting end-to-end training, negatively affecting performance, and becoming impractical when facing high-dimensional features.

Given these considerations, it prompts the question: *Is there an approach to enhance the **scalability** of spectral GNNs **without decoupling** the graph propagation phase?*

Contribution. We propose a novel methodology termed *Spectral Graph Neural Networks with Laplacian Sparsification* (SGNN-LS), inspired by the classic technique of Laplacian sparsification. This technique offers a strategy for deriving ε-sparsifiers of the equivalent propagation matrices associated with spectral GNNs with a high probability. Specifically, our method approximates $\widetilde{\mathbf{L}}_K \approx \sum_{k=0}^{K} w_k\mathbf{L}^k$, while ensuring the number of non-zeros in $\widetilde{\mathbf{L}}_K$ remains within $O\left(\frac{n\log n}{\varepsilon^2}\right)$. Such sparsification not only facilitates the compression of multi-step graph propagation but also effectively connects multi-hop neighbors. This, in turn, enables the application of various scalable GNN algorithms [17, 19, 59]. Importantly, our approach retains the integration of graph propagation within the model. Our contributions are summarized as follows:

- **Plug-and-play strategy for scaling up spectral GNNs.** We propose a method that is the first work tackling the scalability issue of **spectral GNNs** with either static or learnable polynomial coefficients to the best of our knowledge. This strategy offers adaptions of Laplacian sparsification suitable for different scenarios, including models with static polynomial coefficients, those with learnable polynomial coefficients, and a node-wise sampling way for semi-supervised tasks. Our codes are released at https://anonymous.4open.science/r/SGNN-LS-release-B926/.
- **Theoretical analysis.** We present rigorous mathematical proofs to demonstrate that our methods construct Laplacian sparsifiers of dense matrix $\sum_{k=0}^{K} w_k\mathbf{L}^k$ for both static and learnable w_k within $O\left(\frac{n\log n}{\varepsilon^2}\right)$ edges, a high probability $1 - K/n$, and an approximation error ε. Additionally, we introduce a new loss function and establish that the relative error in the calculated loss between the propagated signals, using accurate and approximated propagation matrices, remains within $O(\varepsilon)$. These properties ensure the quality and reliability of generated sparsifiers, guaranteeing the robust performance and efficiency of our model.
- **Extensive experiments.** We conduct comprehensive experiments to validate the effectiveness and scalability of our methods when applied to various spectral GNNs. The results consistently highlight the practical performance of our method, showcasing stable improvements compared to the corresponding baselines. Notably, our method enables the efficient training of GPR-GNN and APPNP on dataset Ogbn-papers100M (0.11B nodes, 1.62B edges) and MAG-Scholar-C (2.78M features), achieving commendable performance metrics.

2 PRELIMINARIES

Notations. In this study, we consider the undirected graph $G = (V, E)$, where V represents the node set and E is the edge set. Let $n = |V|$ and $m = |E|$ denote the number of nodes and edges, respectively. $\mathbf{A} \in \{0,1\}^{n\times n}$ represents the adjacency matrix of the graph G. The diagonal degree matrix is denoted by $\mathbf{D}$, and $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The normalized adjacency matrix and normalized graph Laplacian of G are defined as $\mathbf{P} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and $\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, respectively. Note that the normalized graph Laplacian $\mathbf{L}$ is symmetric and positive semi-definite. We express the eigenvalue decomposition of $\mathbf{L}$ as $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, where $\mathbf{U}$ is a unitary matrix containing the eigenvectors, and $\mathbf{\Lambda} = \mathrm{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ comprises the eigenvalues of $\mathbf{L}$. We usually modify the Laplacian with $\widehat{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}$ to scale the eigenvalues to $[-1, 1]$. Note that the $\lambda_{\max}$ is set to 2 in practice, then we have $\widehat{\mathbf{L}} \approx -\mathbf{P}$.

Spectral GNNs. Spectral-based GNNs exploit the spectral attributes of G and apply the graph convolution operation on the spectral domain. Many works [21, 27] either approximate the filter with polynomial or exhibit similar properties of polynomial filters. The graph filtering operation with respect to the graph Laplacian matrix $\mathbf{L}$ and signal $\mathbf{x}$ is defined as

$$h(\mathbf{L}) * \mathbf{x} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{x} \approx \mathbf{U}\left(\sum_{k=0}^{K} w_k\mathbf{\Lambda}^k\right)\mathbf{U}^\top\mathbf{x} = \left(\sum_{k=0}^{K} w_k\mathbf{L}^k\right)\mathbf{x}, \quad (1)$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_k]$ represents the polynomial coefficient vector. Given the graph filtering operation, the architecture of spectral GNNs is often expressed as

$$f(\mathbf{L}, \mathbf{x}) = f_{\theta_2}\left(h(\mathbf{L}) * f_{\theta_1}(\mathbf{X})\right),$$

where $f_{\theta_i}(\cdot)$ represents the linear layer with coefficients θ_i, and $\mathbf{X}$ combines multiple channels of signal $\mathbf{x}$.

As mentioned in Section 1, the adoption of the detaching trick modifies the architecture of spectral GNNs to the following form:

$$f'(\mathbf{L}, \mathbf{x}) = f_{\theta_2}\left(f_{\theta_1}\left(h(\mathbf{L}) * \mathbf{X}\right)\right).$$

The revised architecture differed from the original by excluding the passage of input features through a linear layer for dimensionality reduction. Due to the absence of learnable parameters prior to graph propagation, the process of $\mathbf{L}^k\mathbf{X}$ can be precomputed. Consequently, the learnable parameters control the linear combination and transformation of the propagated embeddings. Given that the graph structure is engaged solely during graph propagation, the training process is inherently conductive to mini-batching.

Homophily. Homophily measures the tendency of the connected nodes on the graph to have the same label in node classification tasks. This property heavily influences the classic GNN models which utilize one-hop neighbors, while the spectral GNNs can leverage multi-hop neighbor importance. We usually quantify the homophily of a graph with the following edge homophily [57]:

$$\mathcal{H}(G) = \frac{1}{m}\left|\{(u, v) : y(u) = y(v) \wedge (u, v) \in E\}\right|,$$

where $y(\cdot)$ returns the label of nodes. Intuitively, $\mathcal{H}(\cdot)$ denotes the ratio of homophilous edges on the graph. A heterophilous graph implies $\mathcal{H}(G) \to 0$.

3 PROPOSED METHOD

3.1 Motivation

If we take retrospect on the spectral GNNs, this category of GNNs yields promising results, especially when applied to heterophilous datasets. However, an aspect that has received less attention in existing spectral GNN research is scalability. Numerous works[19, 54] that prioritize scalability resort to random sampling techniques to reduce the neighborhood of central nodes, or employ methods like historical embedding [17] to approximate node embeddings. However, the convolution of spectral GNNs gathers information from up to K-hop neighbors, posing challenges to the effective deployment of scalable techniques.

Nevertheless, it is still possible to enhance the scalability of spectral GNNs using straightforward methods. As is mentioned in Section 2, the detaching trick may somewhat extend the scalability of spectral GNNs. However, this trick is not without its drawbacks. First, the decoupling of the GNNs leads to a non-end-to-end model. Emerging approaches [7, 10] based on language models [3, 15] enhance the performance on Ogbn-papers100M, which requires the raw text as input and end-to-end training. Second, we cannot apply a learnable linear layer to reduce the dimension of raw features. Preprocess is potentially impractical when dealing with large graphs with extremely high-dimensional node features. Third, some researches [4, 32] argue that separating training from the graph structure simplifies the network architecture and will negatively impact performance. Given these considerations, we propose "Spectral Graph Neural Networks with Laplacian Sparsification (SGNN-LS)" to enhance the scalability of spectral GNNs without decoupling the network architecture.

3.2 Simplify the Graph Propagation with Laplacian Sparsification

When we revisit Equation 1, we observe that the filter step can be reformulated as a matrix multiplication involving the combined powers of Laplacian (referred to as $\mathbf{L}_K = \sum_{k=0}^{K} w_k \mathbf{L}^k$, Laplacian polynomial) and the input signal. If we obtain the Laplacian polynomial $\mathbf{L}_K$, the propagation is then squeezed to a single step instead of multi-hop message-passing. Since the computation and storage of matrix $\mathbf{L}_k$ is overwhelming, we introduce the Laplacian sparsification [45] to approximate the matrix within high probability and tolerable error.

Laplacian sparsification is designed to create a sparse graph that retains the spectral properties of the original graph. In essence, the constructed graph, with fewer edges, is spectrally similar to the original one. We provide a formal definition of the spectrally similar as follows.

Definition 3.1. Given an weighted, undirected graph $G = (V, E, w)$ and its Laplacian $\mathbf{L}_G$. We say graph G' is spectrally similar to G with approximation error ε if we have

$$(1-\varepsilon)\mathbf{L}_{G'} \preccurlyeq \mathbf{L}_G \preccurlyeq (1+\varepsilon)\mathbf{L}_{G'},$$

where we declare that two matrix $\mathbf{X}$ and $\mathbf{Y}$ satisfy $\mathbf{X} \preccurlyeq \mathbf{Y}$ if $\mathbf{Y} - \mathbf{X}$ is positive semi-definite.

Algorithm 1: Edge Sampling of $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$

Input: Edge set E, power index k.
Output: Sampled edge (u, v)

```
e ← sample an edge from E uniformly at random
i ← sample an integer in [0, k − 1] uniformly at random
u ← the end of random walk on E (i.e. graph G), starting
    from e_u with length i
v ← the end of random walk on E (i.e. graph G), starting
    from e_v with length k − i − 1
return (u, v)
```

A well-established result [22] states that $\mathbf{X} \preccurlyeq \mathbf{Y}$ implies $\lambda_i(\mathbf{X}) \leq \lambda_i(\mathbf{Y})$ for each $1 \leq i \leq n$, where $\lambda_i(\cdot)$ denotes the ith largest eigenvalue of the matrix. This corollary reveals that when two matrices, such as $\mathbf{L}_G$ and $\mathbf{L}_{G'}$, exhibit spectral similarity, their quadratic form and eigenvalues are in close correspondence. Graph sparsification is a fundamental problem of graph theory. Many studies [1] have introduced algorithms that generate ε-sparsifiers for a given graph.

The sole difference between scaled Laplacian and the propagation matrix is a negative sign. We have

$$\mathbf{L}_K = \sum_{k=0}^{K} w_k \mathbf{L}^K \approx \sum_{k=0}^{K} w_k \mathbf{P}^k = \mathbf{D}^{-1/2} \cdot \mathbf{D} \left(\sum_{k=0}^{K} w_k \left(\mathbf{D}^{-1}\mathbf{A} \right)^k \right) \mathbf{D}^{-1/2}, \tag{2}$$

where the negative sign can incorporated into the coefficients w_k. From Equation 2, we observe that we may convert our desiring matrix $\mathbf{L}_K$ to a random walk matrix polynomial $\left(\mathbf{D}^{-1}\mathbf{A}\right)^k$ with coefficients $[w_0, ..., w_K]$. The prior works [8, 43] has presented a promising construction of the sparsifier of a given graph, whose pseudo-code is presented in Algorithm 2. Following them, we have a promising guarantee to approximate random walking matrix polynomial by effective resistance termed Theorem 3.2.

Theorem 3.2. ***(Random walk polynomial sparsification.)*** *For any unweighted, undirected graph G with n vertices and m edges, any $\mathbf{w} = [w_0, w_1, ..., w_K] \in \mathcal{R}_+^{(K+1)}$, $\mathbf{w} \neq \mathbf{0}$ and any approximation parameter ε, we can construct an unbiased ε-sparsifier of the random walk matrix polynomial $\sum_{k=0}^{K} w_k \mathbf{D}\left(\mathbf{D}^{-1}\mathbf{A}\right)^k$ within $O(n \log n/\varepsilon^2)$ edges and probability $1 - K/n$ at least.*

We have extended the original theorem proposed by [8] to accommodate non-normalized polynomial coefficients $\mathbf{w}$. Sampling an edge from $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$ is an atomic operation in constructing our desired graph, and it is frequently employed in our subsequent algorithms. We state this procedure in Algorithm 1. In the upcoming sections, we will present our comprehensive algorithms, which include the weight correction and the adaptation for both static and learnable polynomial coefficients $\mathbf{w}$. For in-depth discussion regarding the approximation estimation, correctness proof, and complexity analysis, please retrieve section 4 and appendix A.

3.2.1 Laplacian Sparsification for Static Polynomial Coefficients. Some of the early classic GNNs, like GCN [26] and APPNP [27], employ static Laplacian polynomials as the graph filter. For example, the propagation layer of APPNP fixes the weight $w_k = \alpha(1 -$

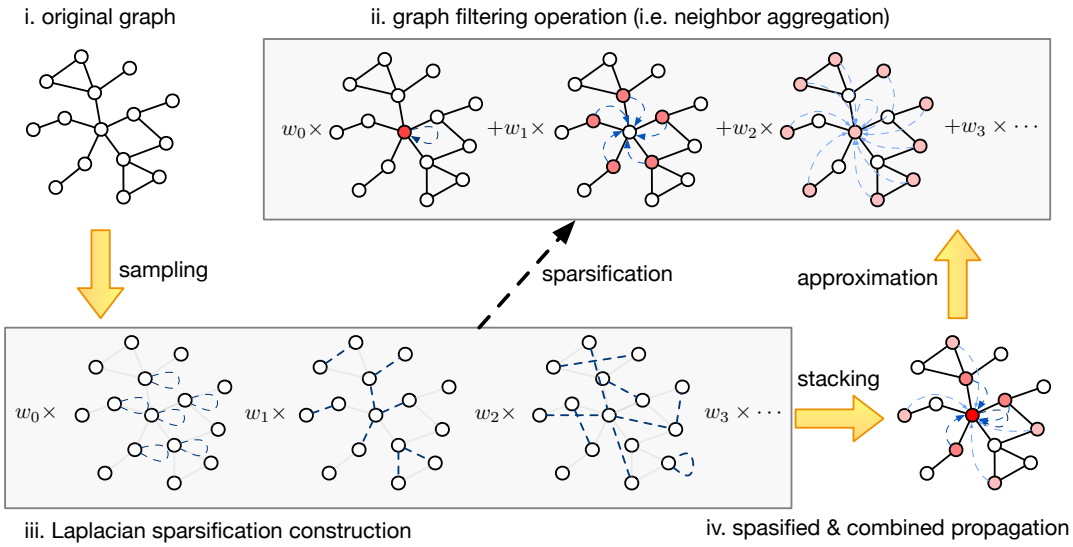

Figure 1: An overview of how Laplacian sparsification works. For clarity, the propagation of one single center node is illustrated. Laplacian sparsification is applied to the entire graph, generating fully sparsified graphs to satisfy the propagation requirements of all the nodes in the graph.

$\alpha)^k, k \neq K$, and $w_K = (1-\alpha)^K$ for different hops, where α is the restart probability of random walk.

Note that if we repeat Algorithm 1 for M times with the same power index k and apply each obtained edge with the weight m/M, we can approximate $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$. This result is still distant from our desired form in Equation 2. First, we need to approximate the random walk matrix polynomial based on our existing method of approximating a term $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$. One intuitive and efficient idea is to distribute the M edges among all $K+1$ subgraphs of $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$. The number of edges assigned to each subgraph follows a multinomial distribution with weights $\mathbf{w}$. Note that w_k is not guaranteed to be positive. The absolute value of w_k is proportional to the probability of being sampled, while $\mathrm{sgn}(w_k)$ decides the sign of the edge weight. We select a random walk length k based on the probability distribution $\Pr\{k=i\} = |w_i|/\|\mathbf{w}\|_1$. Second, we execute Algorithm 1 with edge set E and power index k to generate an edge (u,v). Given that we are now approximating the graph $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$, the edge value is adjusted to

$$\mathrm{sgn}(w_k)\|\mathbf{w}\|_1 \cdot d_u^{-1/2} d_v^{-1/2} \cdot m/M.$$

We provide the pseudo-code for constructing a Laplacian sparsified random walk matrix polynomial with static coefficients in Appendix B.1. Additionally, we offer an example of such a model integrated with our method APPNP-LS in Appendix B.1.

3.2.2 Laplacian Sparsification for Learnable Polynomial Coefficients. Several recent spectral works, like GPR-GNN [11], BernNet [49], and ChebNetII [21], employ learnable polynomial coefficients to dynamically adapt the proper filter. For example, GPR-GNN uses monomial bases to identify the optimal filter, while BernNet employs Bernstein polynomial basis to align with the property that the eigenvalues of normalized Laplacian fall within the range $[0, 2]$.

Upon revisiting the procedure stated above, it becomes apparent that the polynomial coefficients $\mathbf{w}$ primarily affect the sampling of the power index k and the adjustment of edge weights. To facilitate the training of $\mathbf{w}$, we need to calculate the derivative of each w_k for gradient descent. However, we cannot obtain the correct derivatives of $\mathbf{w}$ since all the w_k equally contribute to the part that generates gradients. To address this issue, we directly multiply the polynomial coefficients $\mathbf{w}$ with the weight of the sampled edges instead of the sampling with probability $|w_k|/\|\mathbf{w}\|_1$ in SLSGC. Thus, the edge weight becomes $w_k d_u^{-1/2} d_v^{-1/2} \cdot m/M$. This adjustment connects the gradient of $\mathbf{w}$ with the message passed by the corresponding edges, ensuring the correct derivative chain for training $\mathbf{w}$. However, this modification splits coefficients w_k, leading to independent sampling for each hop k and potentially sacrificing the efficiency. Theoretically, we need to sample more edges to support the training of $\mathbf{w}$ while maintaining the bound of approximation.

We provide the pseudo-code of constructing a Laplacian sparsified random walk matrix polynomial with learnable coefficients (named GLSGC) in Appendix B.2. Besides, we offer an example of such a model integrated with our method GPR-LS in Appendix B.2

3.3 Node-Wise Laplacian Sampling for Semi-supervised Tasks

For representation learning tasks on large-scale graphs with few training nodes, classic S/GLSGC are wasteful as most edges are sampled between nodes in the validation/test set. To address this issue, we propose a node-wise sampling method to approximate the corresponding rows of the result of Equation 2 for the training nodes. This enhancement significantly improves the training efficiency, as nodes in the validation/test set will not aggregate information during training.

Algorithm 2: Edge Sampling by Effective Resistance

Input: Edge set E, upper bound of effective resistance R_{sup}, sampling number M.
Output: Sampled edge set $\widetilde{E}$

$\widetilde{E} \leftarrow \emptyset$
for i from 1 to M **do**
  $\widetilde{E} \leftarrow \widetilde{E} \cup$ sampled edge e with probability $p(e) \propto w(e)R_{\text{sup}}(e)$ and weight $1/(M \cdot R_{\text{sup}}(e))$
return E

Reflecting on Equation 2, it becomes clear that $((\mathbf{D}^{-1}\mathbf{A})^k)_{i,j}$ determines the probability of a random walk of length k starting from node i and ending at node j. This inspires us that we can ensure at least on incident node of the sampled edge belongs to the training set by directly sampling the random walk from the training set. Following Equation 2, for each random walk length k, we first distribute the M sampled edges among all nodes in proportion to their degrees. Then we perform random walk samplings and correct the value of each generated edge (u_0, u_k) to $d_{u_0}^{-1/2} d_{v_0}^{-1/2} \cdot w_k/M$ for each sampled walk $(u_0, u_1, ..., u_k)$.

It can be mathematically proven that the proposed algorithm produces an unbiased approximation of the selected rows of $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$. Moreover, this method is node-wise, allowing for natural mini-batching. As a result, we achieve a strong scalability promotion of the spectral methods. The pesudo-code of this algorithm is proposed in Appendix B.3

4 THEORETICAL ANALYSIS

In this section, we will conduct a comprehensive analysis, including rigorous proofs of correctness, complexity, and other key properties of the methods we have introduced. Due to the space limitation, time and space complexity analysis is proposed in Appendix A.4

4.1 Error Guarantee about Laplacian Sparsification

From a graph theory perspective, a weighted graph is closely related to electric flow. Each edge with weight $w(e), e = (u, v) \in E$ ($w(e) = 1$ for unweighted graphs) can be considered equivalent to a resistor with resistance $1/w(e)$ connecting nodes u and v. When we view the graph as a large complex resistor network, the resistance between u and v is defined as the effective resistance $R(u, v)$.

Lemma 4.1. *[8] **(Upper bound of effective resistance.)** The effective resistance between two nodes u and v on graph G_r is upper bounded by*

$$R_{G_r}(u, v) \leq \sum_{j=0}^{r-1} \frac{2}{\mathbf{A}(i_{j-1}, i_j)} = R_{sup,G_r}(u, v),$$

where $(u = i_0, i_1, \cdots, i_{r-1}, v = i_r)$ is a path on G.

As we are primarily concerned with unweighted graph G, the upper bound can be simplified to a constant $2r$. We can prove a more robust conclusion that Algorithm 1 draws path p on graph G with the probability **strictly** proportional to $w(p)$. This probability is independent of the $R_{\text{sup},G_r}(\cdot)$, which means that running a Monte-Carlo sampling on the graph yields an **unbiased approximation** of G_r. Hence, by replacing the sampling process in Algorithm 2 with Algorithm 1, we obtain an unbiased graph sparsifier generator of G_r with $O(rm \log n/\varepsilon^2)$ edge. This sparsifier can be further reduced to $O(n \log n/\varepsilon^2)$ by the existing works [43, 45]. The proof of Theorem 3.2 is detailed in Appendix A.

In the method proposed in Section 3.3, the sampled random walk of length r originating from a distinct source node u also shares the same upper bound of effective resistance $2r$, which can be equivalently considered as the effective resistance based Laplacian sparsification. Moreover, our method is more intuitive and simplified since we directly sample the desired edge proportional to $w(p)$, without relying on effective resistance. We take consideration of the single candidate start u for the random walks and define $c_r(u, v)$ as the final generated edge weight of (u, v) on graph G_r.

Theorem 4.2. *Given a weighted graph G, and the candidate set U of the random walk starts. For any $u \in U$ and $v \in V$, we have $c_r(u, v)$ is the unbiased approximation of $\left(\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^r\right)_{u,v}$.*

The detailed proof is presented in Appendix A. In practice, we do not need many samples to achieve superior performance for all our proposed methods. This node-centered sampling method enables us to adapt our method to semi-supervised tasks, saving the memory for a larger batch size.

4.2 Error Guarantee about Propagated Signals

Having established the similarity between the original propagation matrix and its approximation, a discernible gap between graph theory and machine learning remains. This section delves deeper into the variances between the propagate signals when utilizing either the original or approximated propagation matrices.

We exemplify this investigation with the APPNP model. Given that different signal channels remain independent during propagation, it is feasible to analyze each channel individually w.l.o.g.. Denote $\mathbf{z}^{(t)}$ as the output signal of APPNP following t rounds of propagation. It has been shown that $\mathbf{z}^{(t)}$ converges to the solution of the following linear system when $t \to \infty$:

$$\left(\mathbf{I} - (1-\alpha)\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)\mathbf{z} = \alpha\mathbf{x}, \tag{3}$$

where $\mathbf{x}$ is the input signal vector. As is investigated in [24], the solution to such a linear system is the optima of some convex quadratic optimization problem.

Proposition 4.3. *[24] Let $\mathbf{z}^*$ be the optima of the following optimization problem.*

$$\min_{\mathbf{z}\in\mathbb{R}^n} (1-\alpha)\text{Tr}(\mathbf{z}^\top\mathbf{L}\mathbf{z}) + \alpha\|\mathbf{z}-\mathbf{x}\|_F^2 \tag{4}$$

Then $\mathbf{z}^*$ is the unique solution to the Equation 3.

To qualify the similarity between any determined $\mathbf{z}$ and $\mathbf{z}^{(\infty)}$, we employ the loss function $\mathcal{L}(\mathbf{z}) = (1-\alpha)\text{Tr}(\mathbf{z}^\top\mathbf{L}\mathbf{z}) + \alpha\|\mathbf{z}-\mathbf{x}\|_F^2$. Furthermore, we aim to demonstrate that the difference in loss, calculated for propagated signals under different propagation matrices is theoretically bounded.

Theorem 4.4. *For any graph G, given input signal $\mathbf{x}$, the propagated signal $\mathbf{z}$ after K rounds of APPNP propagation, and the propagated $\tilde{\mathbf{z}}$ after one round of propagation with an ε-sparsifier of corresponding random walk matrix polynomial. The relative error of the*

loss function $\mathcal{L}(\mathbf{z}) = (1-\alpha)\mathrm{Tr}(\mathbf{z}^\top \mathbf{L}\mathbf{z}) + \alpha\|\mathbf{z}-\mathbf{x}\|_F^2$ *is*

$$\left\| \frac{\mathcal{L}(z) - \mathcal{L}(\tilde{z})}{\mathcal{L}(z)} \right\| = O(\varepsilon).$$

The comprehensive formal proof of Theorem 4.4 is detailed in Appendix A. The Theorem 4.4 shows that the approximated propagation matrix conserves the core attributes of the propagated signal, thereby preserving the fundamental aspects of the model.

5 RELATED WORKS

Spectral GNNs. To align with the proposed methods above, we categorize the spectral GNNs based on their used polynomial filter: static (predefined) polynomial filters and learnable polynomial filters. For static polynomial filters, GCN [26] uses a fixed simplified Chebyshev polynomial approximation and operates as a low-pass filter. APPNP [27] combines the GCN with Personalized PageRank, which can approximate more types of filters but still cannot operate as an arbitrary filter. GNN-LF/HF [58] predefines the graph filter from the perspective of graph optimization. For learnable polynomial filters, ChebNet [14] first approximates the desired filter with the Chebyshev polynomial base, which can operate as an arbitrary filter theoretically. Similarly, GPRGNN [11] considers the monomial base to learn the importance of $k-$hop neighbors directly. BernNet [20] uses the Bernstein polynomial base to make the filter semi-positive definite. ChebNetII [21] revisits the ChebNet and proposes a filter design via Chebyshev interpolation. [18] proposes FavardGNN to learn basis from all possible orthonormal bases and OptBasisGNN to compute the best basis for the given graph.

Subsampling-based Scalable GNNs. Many subsampling [37, 59] methods were studied when the training of GNNs faced the memory limit. This strategy can be broadly divided into two sorts: node sampling methods and subgraph sampling methods. GraphSAGE [19] randomly samples the neighborhood of the aggregation center to approximate the graph propagation. Instead of node-centered sampling, FastGCN [5] deploys a layer-wise sampling method to limit the upper bound of graph propagation in each network layer. Besides, METIS, a well-known clustering algorithm, is used in generating mini-batched graph data for the training of Cluster-GCN [9], GAS [17], LazyGNN [51], and LMC-GCN [41]. GraphSAINT [54] invents multiple new subgraph sampling methods and corresponding normalization coefficients for unbiased training.

Node sampling methods are mostly designed for simplifying and approximating the propagation of the vanilla GCN, and the subgraph sampling methods hinder the long-range interaction between the nodes. These methods are not suitable for extending the scalability of the spectral GNNs discussed in our paper.

Graph Sparsification. Graph sparsification is a consistently studied topic in graph theory. [44] introduces the concept of graph sparsification and presents an efficient algorithm to approximate the given graph Laplacian with a smaller subset of edges while maintaining the spectral properties. [43] leverages effective resistance to yield the promising random sampled edges on the graph. [28] proposes the first method to construct linear-sized spectral sparsification within almost linear time. Our work is mainly enlightened by the works [36] and [8], which make an approximation to the series of the multi-hop random walk sampling matrix.

In the context of GNNs, many works like [31, 56] involve graph sparsification to enhance efficiency or performance. As they usually apply sparsification to the original graph, these methods are not promising in the scenario of multi-step graph propagation which is frequently used in spectral GNNs.

6 EXPERIMENTS

In this section, we will first describe our experimental settings. Next, we will conduct a comprehensive analysis of the experimental results. Due to the space limitation, we provide additional experimental results in Appendix D, including the applicability of our methods on multilayer models, the comparison with ClusterGCN and H2GCN, the comparison among detached models, and some analysis on time, space, executability, and sampling numbers.

6.1 Tested Models, Datasets, and Configurations

Tested models. We compare our method with the vanilla MLP, classic GNNs like GCN, GCNII [6], and GAT [46], detaching methods like SGC, spectral GNN with static polynomial coefficients like APPNP, and spectral GNN with learnable coefficients GPR-GNN, JacobiConv, and FavardGNN. Meanwhile, we involve LazyGNN and LMCGCN to show the ability of up-to-date scalable methods, and PPRGo [2] to test the efficacy of our model when dealing with high-dimensional data. For spectral GNNs, we entangle them with our proposed Laplacian sparsification method for the effectiveness test. All the baselines are reimplemented with Pytorch [34] and PyG [16] library modules as competitors. Codes, detailed parameter matrix, and reproduction guidance are stated in Appendix E.

Datasets. All the baselines and our proposed method are tested with various datasets with diverse homophily and scales, including Cora, Citeseer, PubMed [39, 52], Photos, computers [33, 40], Actor [35], Cornell, Texas, Wisconsin [35], Twitch-de, Twitch-gamers [29], Penn94 [29], Ogbn-arxiv, Ogbn-papers100M [23] and MAG-scholar-C [42]. The downstream task of these datasets is node classification. Detailed information on the datasets is stated in Appendix C.

Configurations. All the experiments except those with dataset Ogbn-papers100M are conducted on the server equipping GPU NVIDIA A100 40GB. For Ogbn-papers100M and MAG-scholar-C, we deploy our experiment on the server with GPU NVIDIA A100 80GB and 512G RAM. Detailed information on experimental environments and package versions are stated in Appendix E.2.

6.2 Accuracy of the Approximation

Before analyzing the test results of SGNN models with and without our plug-in unit, we first examine the similarity between the precisely calculated filtered matrix ($\sum_{k=0}^{K} w_k \mathbf{P}^K$) and the approximated matrices. Compared to other spectral methods, the only new hyperparameter introduced is "--ec", which controls the sampling numbers of each propagation step to $\mathrm{ec} \cdot n \log n$. Although it is not possible to directly calculate ε from "--ec" due to the theoretical use of Big-Oh notation, there is a clear inverse relationship: increasing "--ec" decreases the approximation error ε.

In Figure 2, we present a visualization to intuitively display the original matrix $\mathbf{P}$, the polynomial coefficients, and the difference between the polynomial of $\mathbf{P}$ and the approximated matrix with

Table 1: Experimental results of some baselines and our Laplacian sparsification entangled methods on multiple small-scale datasets. Model name with suffix "-LS" represents the spectral methods entangled with our proposed method. The line beginning with "Avg. Δ" reveals the average performance difference between the base model and its LS-variation. Most evaluation metrics are accuracy(%), but ROC AUC (%) for datasets with 2 classes (Twitch-de, and Twitch-gamers / Penn94 in Table 2). The bold font highlights the best results, whereas the underlined numbers indicate the second and third best.

Dataset $\mathcal{H}(G)$	Cora 0.810	Citeseer 0.736	PubMed 0.802	Actor 0.219	Wisconsin 0.196	Cornell 0.305	Texas 0.108	Photo 0.827	Computers 0.777	Twitch-de 0.632
MLP	76.72±0.89	77.29±0.32	86.48±0.20	39.99±0.76	90.75±2.38	92.13±1.80	92.13±1.64	90.11±0.33	85.00±0.35	68.84±0.54
GAT	86.80±0.94	81.16±0.97	86.61±0.35	35.26±0.82	69.13±3.00	78.36±1.80	79.02±2.95	93.31±0.34	88.39±0.35	67.90±0.75
GCNII	88.52±1.03	81.24±0.65	89.17±0.40	41.20±0.82	82.88±2.50	90.49±1.64	84.75±3.44	94.20±0.23	88.55±0.61	68.03±0.33
PPRGo	87.37±0.95	80.76±0.52	88.35±0.34	39.96±0.25	93.13±1.63	90.49±3.28	89.67±1.80	93.73±1.80	87.20±0.34	71.01±0.61
LMCGCN	86.67±1.12	77.58±0.73	89.86±0.18	35.20±1.43	70.25±3.63	78.36±1.97	79.84±1.97	94.12±0.44	90.67±0.30	68.29±0.76
LazyGNN	89.23±0.71	79.37±1.02	89.67±0.56	40.94±0.80	91.13±1.88	86.56±1.64	87.21±3.44	95.10±0.27	90.54±0.31	67.44±0.59
GCN	87.78±1.05	81.50±0.93	87.39±0.42	35.62±0.52	65.75±3.00	71.96±7.86	77.38±1.97	93.62±0.35	88.98±0.37	73.72±0.61
SGC	87.24±0.97	81.53±0.87	87.17±0.15	34.40±0.58	67.38±3.50	70.82±7.70	79.84±1.64	93.41±0.35	88.61±0.30	73.70±0.67
GPR	88.80±1.17	81.57±0.82	**90.98**±0.25	40.55±0.96	91.88±2.00	89.84±1.80	**92.78**±2.30	95.10±0.26	89.69±0.41	**73.91**±0.65
GPR-LS	**89.31**±1.07	81.65±0.53	90.95±0.37	41.82±0.55	93.63±2.88	91.15±1.15	92.62±1.48	95.30±0.22	90.47±0.41	73.49±0.51
Jacobi	88.46±0.93	80.22±0.61	90.21±0.44	41.03±0.94	89.38±3.26	89.18±2.95	89.02±3.44	94.33±3.44	89.77±0.38	69.57±2.15
Jacobi-LS	89.23±0.74	81.43±0.74	89.87±0.44	41.12±0.80	**93.75**±2.63	89.67±2.30	90.66±2.30	**95.37**±2.30	90.76±0.31	73.29±0.82
Favard	86.65±1.00	81.13±0.86	89.87±0.30	41.39±0.53	92.25±2.25	86.72±2.79	90.00±1.97	94.35±0.30	89.43±0.29	72.78±0.47
Favard-LS	88.65±1.07	81.34±0.68	90.13±0.33	41.00±0.91	92.50±2.13	86.56±3.93	89.70±3.77	95.29±0.31	**90.96**±0.29	73.29±0.76
APPNP	88.69±1.00	81.32±0.68	88.49±0.28	40.73±0.67	90.38±2.38	90.98±2.13	90.82±2.79	93.82±0.26	86.97±0.35	68.29±0.72
APPNP-LS	88.44±1.10	**82.28**±0.49	88.70±0.45	**41.98**±0.43	91.00±3.13	**92.30**±0.98	90.98±1.64	93.79±0.36	87.84±0.34	72.82±0.46
Avg. Δ	+0.76	+0.62	+0.03	+0.56	+1.44	+0.74	+0.34	+0.54	+1.04	+2.09

various sampling numbers on the Texas dataset. To mitigate the impact of dominant values, we excluded $\mathbf{P}^0$ (whose coefficient is 3.22) and fixed the range of the value bar to $[-0.5, 0.5]$. As we have proved that our approximation is unbiased, the difference decreases with an increase in "--ec". When "--ec=10", there is almost no difference between the approximated propagation matrix and the precisely calculated one, highlighting the promising results of our sampling method. The relationship between approximation similarity and final GNN performance is complex. More details about the sampling numbers can be found in Appendix D.4.

6.3 Results on Small-scale Real-world Datasets

In this section, we conduct full-supervised transductive node classification tasks on 10 small-scale real-world datasets. The detailed results are presented in the Table 1. Note that we evaluate the relative performance change between the original base model and its variant, with a clear distinction marked by the horizontal line.

The entries in Table 1 offer an insightful comparison between some well-established GNN models, chosen spectral works, and their Laplacian sparsified variation. GPR-GNN (abbreviated as GPR) is one of the strongest spectral GNNs with a learnable polynomial filter, making our evaluation promising. The practical performance of GPR-GNN with Laplacian sparsification is superior to its original version, even under the potential risk of effect loss caused by the approximation. We further investigate our method over non-trivial monomial bases and channel-wise filters, whose typical instances are JacobiConv and FavardGNN. The results share a similar manner with the GPR-GNN ones.

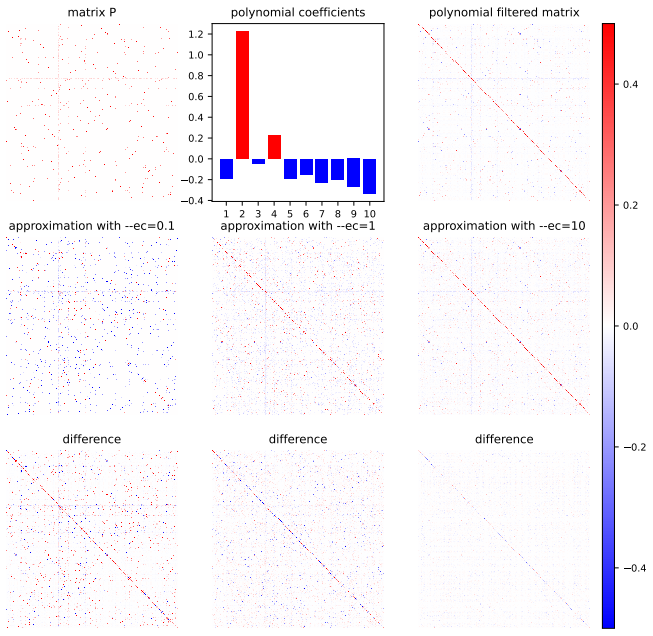

Figure 2: The visualization of the comparison between the polynomial filtered matrix and the results of the Laplacian sparsification with different numbers of samplings, on dataset Texas.

Table 2: Experimental results of some baselines and our Laplacian sparsification entangled methods on medium- and large-scale datasets. Models with the suffix "-de" represent those with detached graph propagation and linear layer. Proposed results share the same annotation formats with Table 1.

Dataset	Arxiv	T-gamers	Penn94	Papers100M
$\mathcal{H}(G)$	0.655	0.554	0.470	-
MLP	54.04	64.90	<u>83.17</u>	41.36
GAT	<u>70.45</u>	62.47	74.14	GPU OOM
GCNII	<u>70.64</u>	63.10	75.19	GPU OOM
PPRGo	63.95	65.20	<u>84.46</u>	-
LMCGCN	67.52	61.61	81.57	-
LazyGNN	70.26	60.58	74.49	-
GCN	70.18	**67.86**	82.55	GPU OOM
SGC	70.35	<u>66.87</u>	GPU OOM	GPU OOM
SGC-de	-	-	-	59.26
Δ	+0.17	-0.99	-	-
GPR	**71.03**	<u>66.69</u>	82.92	GPU OOM
GPR-de	-	-	-	<u>61.68</u>
GPR-LS	70.32	66.45	**85.00**	<u>61.76</u>
Δ	-0.71	-0.24	+2.08	+0.08
APPNP	69.87	65.11	82.89	GPU OOM
APPNP-LS	69.08	65.83	<u>83.17</u>	**62.10**
Δ	-0.79	+0.72	+0.28	-

We choose APPNP as the typical baseline for those spectral works with static polynomial filters. In our implementation, the number of sampled edges for the Laplacian sparsification in spectral works with static polynomial filters is a mere fraction (specifically, k times less) when compared to those with learnable filters. Surprisingly, APPNP-LS exhibits equivalent or even superior performance compared to the original APPNP while achieving a similar performance elevation to the GPR series, despite employing fewer sampled edges.

GCN and SGC, serving as part of our baselines, provided us with some preliminary results about the comparison between common GNNs and their detached versions. While SGC can be considered as the detached version of GCN, the detaching manner does negatively impact the performance of GCN. A more detailed analysis of the detaching manner can be found in Appendix D.2.

6.4 Results on Large-scale Real-world Datasets

To further assess the scalability of both our models and the baselines, we conduct experiments on a diverse range of medium- and large-scale real-world datasets. Note that the dataset Penn94 contains high-dimensional original node representation, and Ogbn-papers100M includes an extensive network with over 111M nodes and 1.6B edges. The experimental results are presented in Table 2.

As is shown in Table 2, our models consistently deliver competitive results even as we scale up to large datasets. Interestingly, on the heterophilous datasets proposed by [29], our models show a slight performance advantage over the corresponding base models, despite the inherent approximation imprecision. This phenomenon shows that the Laplacian sparsification possesses the outstanding ability in 1) denoising, i.e. sampling the unnecessary neighbor connections with low probabilities, and 2) approximating the desired complex filters tailored to the heterophilous graphs.

Table 3: Experimental results of some baselines and our Laplacian sparsification entangled methods on dataset MAG-scholar-C. Proposed results share the same annotation formats with Table 1.

	MAG-scholar-C Accuracy±std(%)	Precomputation Time(s)	Average Epoch Training Time(s)
MLP	83.34±0.08	0	0.12
GCN	GPU OOM	-	-
PPRGo	<u>87.15</u>±0.01	373.52	1.66
SGC	GPU OOM	-	-
SGC-de	OOM	-	-
GPR	GPU OOM	-	-
GPR-de	OOM	-	-
GPR-LS	**87.30**±0.03	0	1.61
APPNP	GPU OOM	-	-
APPNP-LS	<u>86.67</u>±0.07	0	1.91

As data scales up, many existing models suffer scalability challenges. For instance, the standard SGC cannot execute forward propagation on Penn94 without preprocessing since Penn94 contains 1.4 million edges and 4,814 dimensions of original features. The graph propagation leads to GPU out-of-memory errors without the dimensionality reduction. The items marked with "GPU OOM" in Table 2 signify instances where the model cannot be trained on our devices. For SGC and GPRGNN, we preprocess the graph propagation of the required hops on Ogbn-papers100M. Our results clearly demonstrate that GPR-LS attains the performance of the corresponding decoupling method. APPNP-LS allows APPNP to function normally within limited storage space. These outcomes validate the effectiveness of our proposed method in significantly enhancing the scalability of conventional spectral approaches.

6.5 Results on MAG-scholar-C

This section delves into the performance evaluation of the tested models on the MAG-scholar-C dataset, notable for its exceedingly high dimensionality of input node features (2.8M features). The outcomes of these experiments are summarized in Table 3.

It is evident from the experimental data that conventional GNN models are impractical for the MAG-scholar-C dataset due to the significant GPU memory constraints. Moreover, detached models encounter limitations in propagating node features without prior dimensionality reduction, even when operated on CPUs equipped with 512GB of main memory. Conversely, our proposed method enables the entangled models to process the MAG-scholar-C dataset efficiently using mini-batching and dimensionality reduction.

When compared to PPRGo, our GPR-LS model exhibits superior performance, achieving this with reduced training time. Furthermore, all methods employing Laplacian sparsification in our study negate the necessity to compute the approximated PPR matrix, which is a computationally intensive task inherent in PPRGo. Regarding the potential scalability issue possibly encountered when

dealing with large-scale datasets, we provide an analysis about time, space, and excitability in Appendix D.4

7 CONCLUSION

In this paper, we present a novel method centered on Laplacian sparsification to enhance the scalability of spectral GNNs significantly. Our approach demonstrates its capability to approximate the equivalent propagation matrix of Laplacian filters, making it compatible with existing scalable techniques. We provide the theoretical proof affirming that our model produces the correct sparsifier with probability at least $1 - K/n$, approximation parameter ε, and $O\left(\frac{n \log n}{\varepsilon^2}\right)$ non-zeros in the propagation matrix, and thus conserves the core attributes of the propagated signal. The experimental results validate that our methods yield comparable or even superior performance compared to the corresponding base models. This remarkable achievement is particularly noteworthy considering that we sample far fewer edges than the theoretical bound, underscoring the exceptional ability of our method to approximate desired filters.

REFERENCES

[1] Joshua D. Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. 2013. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM* 56, 8 (2013), 87–94. https://doi.org/10.1145/2492007.2492029

[2] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 2464–2473. https://doi.org/10.1145/3394486.3403296

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[4] Julian Busch, Jiaxing Pi, and Thomas Seidl. 2020. PushNet: Efficient and Adaptive Neural Message Passing. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020) (Frontiers in Artificial Intelligence and Applications, Vol. 325)*, Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang (Eds.). IOS Press, 1039–1046. https://doi.org/10.3233/FAIA200199

[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rytstxWAW

[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1725–1735.

[7] Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and Jiliang Tang. 2023. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs. *CoRR* abs/2307.03393 (2023). https://doi.org/10.48550/arXiv.2307.03393 arXiv:2307.03393

[8] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Spectral Sparsification of Random-Walk Matrix Polynomials. *CoRR* abs/1502.03496 (2015). arXiv:1502.03496 http://arxiv.org/abs/1502.03496

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 257–266. https://doi.org/10.1145/3292500.3330925

[10] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. 2022. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

[11] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[12] Guanyu Cui and Zhewei Wei. 2023. MGNN: Graph Neural Networks Inspired by Distance Geometry Problem. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye (Eds.). ACM, 335–347. https://doi.org/10.1145/3580305.3599431

[13] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. 2020. Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting. *IEEE Trans. Intell. Transp. Syst.* 21, 11 (2020), 4883–4894. https://doi.org/10.1109/TITS.2019.2950416

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 3837–3845.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[16] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019). arXiv:1903.02428 http://arxiv.org/abs/1903.02428

[17] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 3294–3304.

[18] Yuhe Guo and Zhewei Wei. 2023. Graph Neural Networks with Learnable and Optimal Polynomial Bases. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 12077–12097.

[19] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1024–1034.

[20] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. 2021. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 14239–14251.

[21] Mingguo He, Zhewei Wei, and Ji-Rong Wen. 2022. Convolutional Neural Networks on Graphs with Chebyshev Approximation, Revisited. In *NeurIPS*.

[22] Roger A. Horn and Charles R. Johnson. 2012. *Matrix Analysis* (2 ed.). Cambridge University Press.

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[24] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 675–684. https://doi.org/10.1145/3447548.3467256

[25] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dong-Sheng Cao, Jian Wu, and Tingjun Hou. 2021. Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *J. Cheminformatics* 13, 1 (2021), 12. https://doi.org/10.1186/s13321-020-00479-8

[26] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

[27] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[28] Yin Tat Lee and He Sun. 2015. Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 250–269. https://doi.org/10.1109/FOCS.2015.24

[29] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 20887–20902.

[30] Junfa Lin, Siyuan Chen, and Jiahai Wang. 2022. Graph Neural Networks with Dynamic and Static Representations for Social Recommendation. In *Database Systems for Advanced Applications - 27th International Conference, DASFAA 2022, Virtual Event, April 11-14, 2022, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13246)*, Arnab Bhattacharya, Janice Lee, Mong Li, Divyakant Agrawal, P. Krishna Reddy, Mukesh K. Mohania, Anirban Mondal, Vikram Goyal, and Rage Uday Kiran (Eds.). Springer, 264–271. https://doi.org/10.1007/978-3-031-00126-0_18

[31] Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. 2023. DSpar: An Embarrassingly Simple Strategy for Efficient GNN training and inference via Degree-based Sparsification. *Trans. Mach. Learn. Res.* 2023 (2023). https://openreview.net/forum?id=SaVEXFuozg

[32] Johannes F. Lutzeyer, Changmin Wu, and Michalis Vazirgiannis. 2022. Sparsifying the Update Step in Graph Neural Networks. In *Topological, Algebraic and Geometric Learning Workshops 2022, 25-22 July 2022, Virtual (Proceedings of Machine Learning Research, Vol. 196)*, Alexander Cloninger, Timothy Doster, Tegan Emerson, Manohar Kaul, Ira Ktena, Henry Kvinge, Nina Miolane, Bastian Rice, Sarah Tymochko, and Guy Wolf (Eds.). PMLR, 258–268.

[33] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, Ricardo Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto (Eds.). ACM, 43–52. https://doi.org/10.1145/2766462.2767755

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035.

[35] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[36] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). ACM, 1509–1520. https://doi.org/10.1145/3308558.3313446

[37] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. *CoRR* abs/2004.11198 (2020). arXiv:2004.11198 https://arxiv.org/abs/2004.11198

[38] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. 2021. Graph Neural Networks for Friend Ranking in Large-scale Social Platforms. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 2535–2546. https://doi.org/10.1145/3442381.3450120

[39] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Mag.* 29, 3 (2008), 93–106. https://doi.org/10.1609/aimag.v29i3.2157

[40] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* abs/1811.05868 (2018). arXiv:1811.05868 http://arxiv.org/abs/1811.05868

[41] Zhihao Shi, Xize Liang, and Jie Wang. 2023. LMC: Fast Training of GNNs via Subgraph Sampling with Provable Convergence. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=5VBBA91N6n

[42] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 243–246. https://doi.org/10.1145/2740908.2742839

[43] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, Cynthia Dwork (Ed.). ACM, 563–568. https://doi.org/10.1145/1374376.1374456

[44] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, László Babai (Ed.). ACM, 81–90. https://doi.org/10.1145/1007352.1007372

[45] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025. https://doi.org/10.1137/08074489X

[46] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[47] Xiyuan Wang and Muhan Zhang. 2022. How Powerful are Spectral Graph Neural Networks. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 23341–23362. https://proceedings.mlr.press/v162/wang22am.html

[48] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6861–6871.

[49] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-Based Recommendation with Graph Neural Networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 346–353. https://doi.org/10.1609/aaai.v33i01.3301346

[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ryGs6iA5Km

[51] Rui Xue, Haoyu Han, MohamadAli Torkamani, Jian Pei, and Xiaorui Liu. 2023. LazyGNN: Large-Scale Graph Neural Networks via Lazy Propagation. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 38926–38937. https://proceedings.mlr.press/v202/xue23c.html

[52] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 40–48.

[53] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Yike Guo and Faisal Farooq (Eds.). ACM, 974–983. https://doi.org/10.1145/3219819.3219890

[54] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[55] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. 2023. Dynamic Graph Neural Networks for Sequential Recommendation. *IEEE Trans. Knowl.*

Data Eng. 35, 5 (2023), 4741–4753. https://doi.org/10.1109/TKDE.2022.3151618

[56] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust Graph Representation Learning via Neural Sparsification. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 11458–11468. http://proceedings.mlr.press/v119/zheng20d.html

[57] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Generalizing Graph Neural Networks Beyond Homophily. *CoRR* abs/2006.11468 (2020). arXiv:2006.11468 https://arxiv.org/abs/2006.11468

[58] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. 2021. Interpreting and Unifying Graph Neural Networks with An Optimization Framework. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 1215–1226. https://doi.org/10.1145/3442381.3449953

[59] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. *CoRR* abs/1911.07323 (2019). arXiv:1911.07323 http://arxiv.org/abs/1911.07323

A PROOFS OF THE PROPOSED THEORIES

A.1 Detailed Analysis about SLSGC and GLSGC Algorithm

In this section, we begin by restating the previously established conclusion.

Theorem A.1. *[8] Given a weighted graph G and the upper bound of its effective resistance $R_{sup}(e) \geq R(e)$. For any approximation parameter ε, there exists a sampled graph $\widetilde{G}$ with at most $M = O\left(\log n/\varepsilon^2 \cdot \left(\sum_{e\in E} w(e)R_{sup}(e)\right)\right)$ edges, satisfying $(1-\varepsilon)\mathbf{L}_G \preccurlyeq \mathbf{L}_{\widetilde{G}} \preccurlyeq (1+\varepsilon)\mathbf{L}_G$ with at least $1-1/n$ probability.*

This theorem marks the initial step, whose pseudo-code of the algorithm is presented at Algorithm 2. Even though our focus is on an unweighted graph G, the final destination of our approximation $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})$ is inherently weighted. The weight between node u and v on the graph G_r with its adjacency matrix $\mathbf{A}_r = \mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^r$ can be considered as the union of all the paths between nodes u and v with length r showing up in the unweighted graph G, i.e. $\mathbf{A}_r(u,v) = \sum_{p\in P} w(p)$, where

$$P = \left\{(u = i_0, i_1, \cdots, i_{r-1}, v = i_r) | (i_j, i_{j+1}) \in E, j = 0, 1, \cdots, r-1\right\}.$$

The weight of a path, denoted as $p = (u_0, u_1, ..., u_r)$ can be formally expressed as

$$w(p) = \frac{\prod_{i=0}^{r-1} \mathbf{A}_{u_{i-1},u_i}}{\prod_{i=1}^{r-1} \mathbf{D}_{u_i,u_i}}.$$

This value is symmetrical when viewed from u_0 and u_r, which is a slight deviation from the random walk probability, as it encompasses the probability of the walk starting from the initial node.

Retrieving Lemma 4.1, the upper bound of effective resistance for a path p on the weighted graph can be expressed as

$$R_{\sup,G_r}(u_0,u_r) = \sum_{i=0}^{r-1} \frac{2}{\mathbf{A}_{u_i,u_{i+1}}},$$

where p is a path within graph G, startingfrom u_0, ending at u_r, and possessing a length of r.

We derive the conclusion that

$$\begin{aligned}
&\sum_{p=(u_0,\cdots,u_r)} w(p) R_{\sup,G_r}(u_0,u_r) \qquad (5)\\
&= \sum_p \left(\sum_{i=0}^{r-1} \frac{2}{\mathbf{A}_{u_i,u_{i+1}}}\right)\left(\frac{\prod_{j=0}^{r-1}\mathbf{A}_{u_j,u_{j+1}}}{\prod_{j=1}^{r-1}\mathbf{D}_{u_j,u_j}}\right)\\
&= 2\sum_p \sum_{i=1}^{r}\left(\frac{\prod_{j=1}^{i-1}\mathbf{A}_{u_{j-1},u_j}\prod_{j=i}^{r-1}\mathbf{A}_{u_j,u_{j+1}}}{\prod_{j=1}^{r-1}\mathbf{D}_{u_j,u_j}}\right)\\
&= 2\sum_{e\in E}\sum_{i=1}^{r}\left(\sum_{p|(u_{i-1},u_i)=e} \frac{\prod_{j=1}^{i-1}\mathbf{A}_{u_{j-1},u_j}}{\prod_{i=1}^{j-1}\mathbf{D}_{u_j,u_j}} \cdot \frac{\prod_{j=i}^{r-1}\mathbf{A}_{u_j,u_{j+1}}}{\prod_{j=i}^{r-1}\mathbf{D}_{u_j,u_j}}\right) \qquad (6)\\
&= 2\sum_{e\in E}\sum_{i=1}^{r} 1\\
&= 2mr. \qquad (7)
\end{aligned}$$

This conclusion holds since $\mathbf{D}_{u,u} = \sum_{v\in V}\mathbf{A}_{u,v}$. Recall that the derivation of Equation 6 implies the sampling method process of the Laplacian sparsification.

For the proposed Algorithm 1, the probability of sampling a distinct path $p = (u_0, \cdots, u_r)$ can be derived as

$$\begin{aligned}
&\Pr(p = (u_0,\cdots,u_r|e,k))\\
&= \Pr((u_{k-1},u_k) = e)\cdot \Pr(p = (u_0,\cdots,u_r)|(u_{k-1},u_k) = e)\\
&= \frac{1}{m}\cdot\prod_{i=1}^{k-1}\frac{\mathbf{A}_{u_i,u_{i-1}}}{\mathbf{D}_{u_i,u_i}}\cdot\prod_{i=k}^{r-1}\frac{\mathbf{A}_{\mathbf{u_i},\mathbf{u_{i+1}}}}{\mathbf{D}_{u_i,u_i}}\\
&= \frac{1}{m}\cdot\frac{\prod_{i=1}^{r}\mathbf{A}_{u_{i-1},u_i}}{\left(\prod_{i=1}^{r-1}\mathbf{D}_{u_i,u_i}\right)\cdot\mathbf{A}_{u_{k-1},u_k}}.
\end{aligned}$$

Since e is sampled uniformly at random, as indicated by the term $\Pr((u_{k-1},u_k) = e)$ above, we now consider the randomness introduced by k. k is also sampled uniformly at random. Thus, eliminating k finally yields

$$\begin{aligned}
&\Pr(p = (u_0,\cdots,u_r))\\
&= \left(\frac{1}{m}\cdot\frac{\prod_{i=1}^{r}\mathbf{A}_{u_{i-1},u_i}}{\prod_{i=1}^{r-1}\mathbf{D}_{u_i,u_i}}\right)\left(\frac{1}{r}\cdot\sum_{k=1}^{r}\frac{1}{\mathbf{A}_{u_{k-1},u_k}}\right)\\
&= \frac{1}{2mr}w(p)R_{\sup,G_r}(p).
\end{aligned}$$

Since all the edge weights are considered as 1 on unweighted graphs, the upper bound $R_{\sup,G_r}$ is $2r$ for any p with length r. As is proved that $\Pr(p) \propto w(e)R_{\sup}(p)$, we can execute the Algorithm 2 with M times of Monte-Carlo sampling stated in Algorithm 1 to construct an ε-sparsifier.

In another view, the probability of sampling a distinct path p is proportional to $w(p)$. Follow Equation 7, the summation of $w(p)$ for any length r is m. We can execute the Monte-Carlo sampling M times to generate M paths with length r and weight m/M to generate the unbiased approximation of $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^r$, which further proves the accuracy of the sampling procedure.

Hence, Theorem A.1 have been proven. There exists an algorithm that employs the process outlined above to generate an ε-sparsifier

of $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^r$ within $M = O(n \log n/\varepsilon^2)$ edges with the probability $1 - 1/n$ at least. Our final objective is to approximate the $\mathbf{D}^{-1/2} \cdot \mathbf{D}\left(\sum_{k=0}^{K} w_k \left(\mathbf{D}^{-1}\mathbf{A}\right)^k\right)\mathbf{D}^{-1/2}$ proposed in Equation 1.

In general, all of the sampled edges (u_0, u_r), representing path $p = (u_0, ..., u_r)$, should be multiplied by a correction coefficient of $d_{u_0}^{-1/2} d_{u_r}^{-1/2}$, intuitively.

For random walk polynomials with static coefficients, such as in SLSGC, one effective method to approximate the middle term is to distribute all the edges with the probability proportional to $|w_i|$ for each path length i. Hence, we can first pick the length i with the probability $\Pr\{r = i\} = |w_i|/\|\mathbf{w}\|_1$, then sample the edge and correct the edge weight with $d_{u_0}^{-1/2} d_{v_0}^{-1/2} \operatorname{sgn}(w_i)\|\mathbf{w}\|_1$ to compensate for the rescaling of the probability. This process intuitively maintains the unbiased approximation and the Laplacian sparsification properties. In conclusion, when we are approximating the graph $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^k$, the edge value is adjusted to

$$\operatorname{sgn}(w_k)\|\mathbf{w}\|_1 \cdot d_u^{-1/2} d_v^{-1/2} \cdot m/M.$$

For random walk polynomials with learnable coefficients, the models are required to generate the correct hop-independent derivative of the w_i. This limitation prevents us from directly sampling with w_k. Instead, we may sample the graph hop-by-hop, meaning that for each random walk length i, we independently generate the sparsifier of graph $\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^i$ and stack them with corresponding coefficients w_i.

To maintain the property of being an ε-sparsifier of the given graph, one sufficient condition is that all the generated K sparsifiers are ε-sparsifiers. Thus, the required number of generated edges increases to

$$K \cdot O(n \log n/\varepsilon^2) = O(n \log n/\varepsilon^2),$$

and the probability decreases to

$$(1 - 1/n)^K \geq 1 - K/n.$$

Since K is a predefined constant and $K \ll n$, the complexity does not change significantly. Meanwhile, the generated K components are all unbiased approximations of $\mathbf{D}(\mathbf{D}^{-1}A)^i$, the stacked approximation is also unbiased, evidently. Thus, Theorem 3.2 is proved.

A.2 Proof of Theorem 4.2

We can begin by considering the probability of the sampling method selecting a distinct path on the graph. Assume the desired path is $p = (u_0, \cdots, u_r)$ and the first selected node is u, we can derive the probability as follows:

$$\begin{aligned}
&\Pr(p = (u_0, \cdots, u_r)) \\
&= \Pr(u = u_0) \cdots \Pr(p = (u_0, \cdots, u_r) | u = u_0) \\
&= \frac{\mathbf{D}_{u_0,u_0}}{\sum_{v\in U} \mathbf{D}_{v,v}} \cdot \prod_{i=0}^{r-1} \frac{\mathbf{A}_{u_i,u_{i+1}}}{\mathbf{D}_{u_i,u_i}} \\
&= \frac{1}{\sum_{v\in U} \mathbf{D}_{v,v}} \cdot \frac{\prod_{i=0}^{r-1} \mathbf{A}_{u_i,u_{i+1}}}{\prod_{i=1}^{r-1} \mathbf{D}_{u_i,u_i}} \\
&= \frac{w(p)}{\sum_{v\in U} \mathbf{D}_{v,v}},
\end{aligned}$$

where U is the set of nodes where the start of random walks are selected from.

From the derivation of Equation 7, we can conclude that the summation of weights of all the paths starting from node u can be divided into $\mathbf{D}_{u,u}$ series, where each series starts with one of the $\mathbf{D}_{u,u}$ edges incident with node u. Since the summation of all the paths with one distinct k satisfying $(u_k, u_{k+1}) = e$ is 1, we have the summation of weights of all the paths starting from u is $\mathbf{D}_{u,u}$. This means the probability of sampling a distinct path is proportional to its weight.

The entry $\left(\mathbf{D}(\mathbf{D}^{-1}\mathbf{A})^r\right)_{i,j}$ represents the combination of the weights of all the path $p = (u_0 = i, u_1, \cdots, u_{r-1}, u_r = j)$. For each sampled path, we can correct it by multiplying it with $\frac{1}{M}\sum_{v\in U} \mathbf{D}_{v,v}$ to obtain an unbiased approximation of the weight of each path. The union of the paths will inevitably generate an unbiased approximation of the corresponding rows of U in $\mathbf{D}(\mathbf{D}^{-1}A)^r$. Thus, the final weight generated for the path $p = (u_0, \cdots, u_r)$ is

$$\frac{d_{u_0}^{-1/2} d_{u_r}^{-1/2}}{M}\left(\sum_{v\in U} \mathbf{D}_{v,v}\right),$$

which indicates that the Theorem 4.2 has been proven.

A.3 Proof of Theorem 4.4

First, the loss function can be derived as

$$\mathcal{L}(\mathbf{z}) = (1-\alpha)\mathbf{z}^\top \mathbf{L}\mathbf{z} + \alpha(\mathbf{z} - \mathbf{x})^\top(\mathbf{z} - \mathbf{x}).$$

Let $f(\mathbf{P}, K) = \sum_{k=0}^{K-1} \alpha(1-\alpha)^k \mathbf{P}^k + (1-\alpha)^K \mathbf{P}^K$ be the original APPNP propagation matrix, and $\tilde{f}(\mathbf{P}, K)$ be the approximated one. Evidently, the connection between the normalized Laplacian and the propagation matrix can be expressed as:

$$\begin{aligned}
\mathbf{L}_K &= \mathbf{I} - f(\mathbf{P}, K), \\
\widetilde{\mathbf{L}}_K &= \mathbf{I} - \tilde{f}(\mathbf{P}, K).
\end{aligned}$$

For convenience, we abbreviate $f(\mathbf{P}, K)$ and $\tilde{f}(\mathbf{P}, K)$ into f and $\tilde{f}$, respectively. The propagated signals are defined as $\mathbf{z} = f\mathbf{x}$ and $\tilde{\mathbf{z}} = \tilde{f}\mathbf{x}$. Then, loss functions $\mathcal{L}(\mathbf{z})$ and $\mathcal{L}(\tilde{\mathbf{z}})$ can be derived as:

$$\begin{aligned}
\mathcal{L}(\mathbf{z}) &= (1-\alpha)\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x} + \alpha \mathbf{x}^\top (f - \mathbf{I})^\top (f - \mathbf{I})\mathbf{x} \\
&= \mathbf{x}^\top \left((1-\alpha) f^\top \mathbf{L} f + \alpha (f - \mathbf{I})^2\right)\mathbf{x}, \\
\mathcal{L}(\tilde{\mathbf{z}}) &= \mathbf{x}^\top \left((1-\alpha) \tilde{f}^\top \mathbf{L} \tilde{f} + \alpha (\tilde{f} - \mathbf{I})^2\right)\mathbf{x}.
\end{aligned}$$

According to the conditions, we can obtain some partial orderings between f and $\tilde{f}$:

$$\begin{cases}
(1-\varepsilon)\,(\mathbf{I} - f) \preccurlyeq \mathbf{I} - \tilde{f} \preccurlyeq (1+\varepsilon)\,(\mathbf{I} - f) \\
(1-\varepsilon)^2\,(\mathbf{I} - f)^2 \preccurlyeq (\mathbf{I} - \tilde{f})^2 \preccurlyeq (1+\varepsilon)^2\,(\mathbf{I} - f)^2 \\
-\varepsilon(\mathbf{I} - f) \preccurlyeq \Delta \preccurlyeq \varepsilon(\mathbf{I} - f) \\
-\frac{\varepsilon}{1+\varepsilon}(\mathbf{I} - \tilde{f}) \preccurlyeq \Delta \preccurlyeq \frac{\varepsilon}{1-\varepsilon}(\mathbf{I} - \tilde{f})
\end{cases},$$

where $\Delta = \mathbf{L}_K - \widetilde{\mathbf{L}}_K = f - \tilde{f}$. Thus, we may calculate the absolute error between $\mathcal{L}(\mathbf{z})$ and $\mathcal{L}(\tilde{\mathbf{z}})$ as follows:

$$\begin{aligned}
\mathcal{L}(\mathbf{z}) - \mathcal{L}(\tilde{\mathbf{z}}) &= \mathbf{x}^\top \left((1-\alpha)\left(f^\top \mathbf{L} f - \tilde{f}^\top \mathbf{L} \tilde{f}\right) + \alpha\left((f-\mathbf{I})^2 - (\tilde{f}-\mathbf{I})^2\right)\right)\mathbf{x} \\
&= \mathbf{x}^\top \left((1-\alpha)\delta\mathbf{X} + \alpha\delta\mathbf{Y}\right)\mathbf{x}, \\
\delta\mathbf{X} &= f^\top \mathbf{L} f - \tilde{f}^\top \mathbf{L} \tilde{f} \\
&= f\mathbf{L}f - f\mathbf{L}\tilde{f} + f\mathbf{L}\tilde{f} - \tilde{f}\mathbf{L}\tilde{f} \\
&= f\mathbf{L}(f-\tilde{f}) + (f-\tilde{f})\mathbf{L}\tilde{f} \\
&= f\mathbf{L}\Delta + \Delta\mathbf{L}\tilde{f}, \\
\delta\mathbf{Y} &= (\mathbf{I}-f)^2 - (\mathbf{I}-\tilde{f})^2.
\end{aligned}$$

Then we calculate the relative error $\left|\frac{\mathcal{L}(z)-\mathcal{L}(\tilde{z})}{\mathcal{L}(z)}\right|$ by part:

$$\frac{\mathbf{x}^\top \delta\mathbf{X}\mathbf{x}}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}} = \frac{\mathbf{x}^\top (f\mathbf{L}\Delta + \Delta L\tilde{f})\mathbf{x}}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}}.$$

Since the matrices $\Delta, \mathbf{L}, \tilde{f}$ are all symmetric, we have $\mathbf{x}^\top \Delta\mathbf{L}\tilde{f}\mathbf{x} = \mathbf{x}^\top (\Delta\mathbf{L}\tilde{f})^\top \mathbf{x} = \mathbf{x}^\top \tilde{f}\mathbf{L}\Delta\mathbf{x}$. Thus, we can derive that:

$$\begin{aligned}
\frac{\mathbf{x}^\top \delta\mathbf{X}\mathbf{x}}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}} &= \frac{\mathbf{x}^\top (f+\tilde{f})\mathbf{L}\Delta\mathbf{x}}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}} \\
&= \frac{2\mathbf{x}^\top f\mathbf{L}\Delta\mathbf{x}}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}} - \frac{\mathbf{x}\Delta\mathbf{L}\Delta\mathbf{x}^\top}{\mathbf{x}^\top f^\top \mathbf{L} f \mathbf{x}} \\
&= O(\varepsilon) - O(\varepsilon^2) = O(\varepsilon).
\end{aligned}$$

Meanwhile, the other part shows

$$\begin{aligned}
\frac{\mathbf{x}^\top \delta Y \mathbf{x}}{\mathbf{x}^\top (\mathbf{I}-f)^2 \mathbf{x}} &= \frac{\mathbf{x}^\top \left((\mathbf{I}-f)^2 - (\mathbf{I}-\tilde{f})^2\right)\mathbf{x}}{\mathbf{x}^\top (\mathbf{I}-f)^2 \mathbf{x}} \\
&= 1 - \frac{\mathbf{x}^\top (\mathbf{I}-\tilde{f})^2 \mathbf{x}}{\mathbf{x}^\top (\mathbf{I}-f)^2 \mathbf{x}}
\end{aligned}$$

.

Since we have

$$(1-\varepsilon)^2 (\mathbf{I}-f)^2 \preccurlyeq (\mathbf{I}-\tilde{f})^2 \preccurlyeq (1+\varepsilon)^2 (\mathbf{I}-f)^2,$$

we may obtain

$$\begin{aligned}
&\Rightarrow -2\varepsilon - \varepsilon^2 \le 1 - \frac{\mathbf{x}^\top (\mathbf{I}-\tilde{f})^2 \mathbf{x}}{\mathbf{x}^\top (\mathbf{I}-f)^2 \mathbf{x}} \le 2\varepsilon - \varepsilon^2, \\
&\Rightarrow 1 - \frac{\mathbf{x}^\top (\mathbf{I}-\tilde{f})^2 \mathbf{x}}{\mathbf{x}^\top (\mathbf{I}-f)^2 \mathbf{x}} = O(\varepsilon) + O(\varepsilon^2) = O(\varepsilon).
\end{aligned}$$

Hence, the relative error finally yields

$$\begin{aligned}
\left|\frac{\mathcal{L}(z)-\mathcal{L}(\tilde{z})}{\mathcal{L}(z)}\right| &= \left|\frac{\mathbf{x}^\top \left((1-\alpha)\delta\mathbf{X} + \alpha\delta\mathbf{Y}\right)\mathbf{x}}{\mathbf{x}^\top \left((1-\alpha)f^\top \mathbf{L} f + \alpha(f-\mathbf{I})^2\right)\mathbf{x}}\right| \\
&\le \left|\frac{\mathbf{x}^\top (1-\alpha)\delta\mathbf{X}\mathbf{x}}{\mathbf{x}^\top \left((1-\alpha)f^\top \mathbf{L} f + \alpha(f-\mathbf{I})^2\right)\mathbf{x}}\right| \\
&\quad + \left|\frac{\mathbf{x}^\top \alpha\delta\mathbf{Y}\mathbf{x}}{\mathbf{x}^\top \left((1-\alpha)f^\top \mathbf{L} f + \alpha(f-\mathbf{I})^2\right)\mathbf{x}}\right| \\
&\le \left|\frac{\mathbf{x}^\top (1-\alpha)\delta\mathbf{X}\mathbf{x}}{\mathbf{x}^\top (1-\alpha)f^\top \mathbf{L} f \mathbf{x}}\right| + \left|\frac{\mathbf{x}^\top \alpha\delta\mathbf{Y}\mathbf{x}}{\mathbf{x}^\top \alpha(f-\mathbf{I})^2\mathbf{x}}\right| \\
&= O(\varepsilon) + O(\varepsilon) = O(\varepsilon).
\end{aligned}$$

A.4 Time and Space Complexity Analysis

In this section, we provide a thorough time and space complexity analysis of our methods and the used backbones.

Briefly, our method streamlines the graph convolution process in spectral GNNs. For k-order polynomial filters, the conventional SGNNs require $O(Km)$ node message-passing operations, whereas our approach reduces this to $O(Kn\log n/\varepsilon^2)$. GPR-GNN serves as our primary model for detailed analysis, with implications extendable to similar models like FavardGNN.

Time and memory overhead of GPRGNN.

Training time: $O(KmF+LnF^2+nF_iF)$ per epoch, where F_i denotes the dimension of original node embeddings.

Training memory: $O(LnF + LF^2 + F_iF + m)$.

For detached GPRGNN, the precomputation of propagations are required.

Precomputation time: $O(KmF_i)$ for K rounds of graph propagation.

Precomputation memory: $O(KnF_i)$ to store $K+1$ node representation matrices of different hops.

Training time: $O(Ln_tF^2 + n_tF_iF)$ per **epoch**, where n_t is the number of nodes in the training set. The training contains L layers of MLP.

Training memory: $O(Ln_bF + LF^2 + F_iF)$ per **batch**. This phase includes L Layers of MLP, while the initial layer transform F_i dimensional layers to F dimensions. n_b denotes the number of nodes in mini-batch, which implies that the model can be trained with mini-batch.

For GPRGNN-LS, the whole model is entangled.

Training time: $O(KMF + LnF^2 + F_sF)$ per **epoch**, where $M = O(n\log n/\varepsilon^2)$, and F_s determines the number of non-zeros in all of the original node embeddings.

Sampling time: $O(K^2M) = O(K^2 n\log n/\varepsilon^2)$, which is not the bottleneck since the random walk number can be controlled without influencing the performance negatively.

Training time: $O(Kn_tr_sF + LnF^2 + F_{st}F)$ per **epoch** for semi-supervised tasks, where r_s is the number of random walk sampling, and F_{st} is the number of non-zeros in all of the training node original embeddings.

Sampling time: $O(K^2n_tr_s)$.

Training memory: $O(Ln_bF + LF^2 + F_iF + m)$ per **batch**.

Our proposed method addresses the significant GPU memory constraints encountered with very large graphs, such as the Ogbn-papers100M dataset, where storing node embeddings for the entire graph requires $O(LnF)$ memory. By enabling mini-batch training for SGNNs, we effectively mitigate these limitations. As our comprehensive time complexity analysis employs GPR-GNN as the primary example, the conclusions drawn from this model are applicable to others, such as FavardGNN, by similar inference.

For datasets with dense original node embeddings, such as $F_s \approx nF_i$ and $F_{st} \approx n_tF_i$, our method remains efficient. However, in cases like the MAG-Scholar-C dataset, where $F_s \ll nF_i$ and $F_{st} \ll n_tF_i$, traditional approaches like detached GPRGNN encounter bottlenecks due to the high computational demands of graph propagation on CPUs, even with substantial RAM. Our approach overcomes these challenges by eliminating the need for extensive precomputation in the detached GPRGNN and facilitating mini-batch training,

Algorithm 3: Static Laplacian Sparsified Graph Construction (SLSGC)

Input: Hop weights $\mathbf{w}$.
Input: Model Saved: Vertice set V, edge set E, degrees $\mathbf{d}$, maximum neighbor hop K, total sampling number M.
Output: Weighted edge set $\widetilde{E}$ after laplacian sparsification.

$m \leftarrow |E|$
$\widetilde{E} \leftarrow \emptyset$
for i from 1 to M **do**
  $k \leftarrow$ sample an integer k from distribution $\Pr\{k = j\} = |w_j|/\|\mathbf{w}\|_1$
  $(u, v) \leftarrow$ Edge_Sampling(E, k)
  $\widetilde{E} \leftarrow \widetilde{E} \cup \left((u, v), \mathrm{sgn}(w_k)\|\mathbf{w}\|_1 d_u^{-1/2} d_v^{-1/2} \cdot \frac{m}{M}\right)$
return $\widetilde{E}$

Algorithm 4: APPNP with Laplacian Sparsification

Input: Node embeddings $\mathbf{X}$, training status T.
Input: Model Saved: Vertice set V, edge set E, degrees $\mathbf{d}$, maximum neighbor hop K, sampling number M, hop weight matrix $\mathbf{W}$.
Output: Processed node embeddings $\widetilde{\mathbf{X}}$

$\mathbf{X} \leftarrow \mathrm{Linear}(\mathbf{X})$
if T **then**
  $\widetilde{E} \leftarrow \mathrm{SLSGC}(\mathbf{W}_0)$
  $\widetilde{\mathbf{X}} \leftarrow$ A round of message passing of $\mathbf{X}$ on edge set $\widetilde{E}$
else
  $\widetilde{\mathbf{X}} \leftarrow \mathbf{W}_{0,0}\mathbf{X}$
  for i from 1 to K **do**
    $\mathbf{X} \leftarrow$ A round of message passing of $\mathbf{X}$ on edge set $G.E$ with weights $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
    $\widetilde{\mathbf{X}} \leftarrow \widetilde{\mathbf{X}} + \mathbf{W}_{0,i}\mathbf{X}$
Possibly move ahead.
$\widetilde{\mathbf{X}} \leftarrow \mathrm{Linear}(\widetilde{\mathbf{X}})$
return $\widetilde{\mathbf{X}}$

Algorithm 5: General Laplacian Sparsified Graph Construction (GLSGC)

Input: Hop weights $\mathbf{w}$.
Input: Model Saved: Vertice set V, edge set E, degrees $\mathbf{d}$, maximum neighbor hop K, layer-wise sampling number M.
Output: Weighted edge set $\widetilde{E}$ after laplacian sparsification.

$m, \widetilde{E} \leftarrow |E|, \emptyset$
for each $v \in V$ **do**
  $\widetilde{E} \leftarrow \widetilde{E} \cup ((v, v), \mathbf{w}_0)$
for k from 1 to K **do**
  for j from 1 to M **do**
    $(u, v) \leftarrow$ Edge_Sampling(E, k)
    $\widetilde{E} \leftarrow \widetilde{E} \cup \left((u, v), \mathbf{w}_k d_u^{-1/2} d_v^{-1/2} \cdot \frac{m}{M}\right)$
return $\widetilde{E}$

Algorithm 6: GPRGNN with Laplacian Sparsification

Input: Node embeddings $\mathbf{X}$, training status T.
Input: Model Saved: Vertice set V, edge set E, degrees $\mathbf{d}$, maximum neighbor hop K, sampling number M, hop weight matrix $\mathbf{W}$.
Output: Processed node embeddings $\widetilde{\mathbf{X}}$

$\mathbf{X} \leftarrow \mathrm{Linear}(\mathbf{X})$
if T **then**
  $\widetilde{E} \leftarrow \mathrm{GLSGC}(\mathbf{W}_0)$
  $\widetilde{\mathbf{X}} \leftarrow$ A round of message passing of $\mathbf{X}$ on edge set $\widetilde{E}$
else
  $\widetilde{\mathbf{X}} \leftarrow \mathbf{W}_{0,0}\mathbf{X}$
  for i from 1 to K **do**
    $\mathbf{X} \leftarrow$ A round of message passing of $\mathbf{X}$ on edge set $G.E$ with weights $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
    $\widetilde{\mathbf{X}} \leftarrow \widetilde{\mathbf{X}} + \mathbf{W}_{0,i}\mathbf{X}$
Possibly move ahead
$\widetilde{\mathbf{X}} \leftarrow \mathrm{Linear}(\widetilde{\mathbf{X}})$
return $\widetilde{\mathbf{X}}$

thus accommodating large-dimensional sparse node embeddings more effectively.

B EXAMPLE PSEUDO-CODES OF THE ENTANGLED MODELS

B.1 SLSGC and APPNP

This section presents the pseudo-code of constructing a Laplacian spasified random walk polynomial with static coefficients in Algorithm 3 and Laplacian sparsification entangled APPNP in Algorithm 4. Note that the hop weight matrix $\mathbf{W}$ is not learnable. $\mathbf{W}$ is predefined as $\mathbf{W}_{0,i} = \alpha(1-\alpha)^i, i \neq K$ and $\mathbf{W}_{0,K} = (1-\alpha)^K$, where α is a hyper-parameter.

B.2 GLSGC and GPRGNN-LS

This section presents the pseudo-code of constructing a Laplacian sparsified random walk matrix polynomial with learnable coefficients in Algorithm 5 and Laplacian sparsification entangled GPRGNN in Algorithm 6. Note that the hop weight matrix $\mathbf{W}$ is learnable. Please follow GPRGNN [11] for more details of the initialization of $\mathbf{W}$.

B.3 Node-wise Laplacian Sampling

In this section, Algorithm 7 presents the pseudo-code of the node-wise Laplacian sampling algorithm for semi-supervised tasks.

Algorithm 7: Node-Wise Laplacian Sampling

Input: Hop weights $\mathbf{w}$, batch B
Input: Model Saved: Vertice set V, edge set E, degrees $\mathbf{d}$, maximum neighbor hop K, sampling number M.
Output: Weighted edge set $\widetilde{E}$ after random walk sampling.

$s \leftarrow \sum_{u \in B} \mathbf{d}_u$
$\widetilde{E} \leftarrow \emptyset$
for k from 1 to K **do**
  for j from 1 to M **do**
    $u \leftarrow$ sample a node in B from distribution $\Pr\{u = u_0\} = \mathbf{d}_{\mathbf{u}_0}/s$
    $v \leftarrow$ the end of a random walk on E (i.e. graph G), starting from u with length k
    $\widetilde{E} \leftarrow \widetilde{E} \cup \left((u, v), \mathbf{w}_k d_u^{-1/2} d_v^{-1/2} \cdot \frac{s}{M}\right)$
return $\widetilde{E}$

C DATASET DETAILS

All the baselines and our proposed method are tested with various datasets with diverse homophily and scales. Cora, Citeseer, and PubMed [39, 52] have been the most widely tested citation networks since the emergence of the GCN. Photos and computers [33, 40] are the segment of the co-purchase graphs from Amazon. Actor [35] is the co-occurrence graph of film actors on Wikipedia. Cornell, Texas, and Wisconsin [35] are the webpage hyperlink graphs from WebKB. Twitch-de and Twitch-gamers [29] are the social network of Twitch users. Penn94 [29] is a friendship network from the Facebook 100 networks. Ogbn-arxiv and Ogbn-papers100M [23] are two public datasets proposed by the OGB team, where Ogbn-papers100M contains over 100 million nodes and 1 billion edges. MAG-scholar-C is another benchmark from PPRGo based on Microsoft Academic Graph [42] with over 12.4 million nodes and 2.8 million node features.

Here we list the detailed information of used datasets in the experiment. All the information we proposed here refers to the original version of data collected by PyG. All the graphs are converted to the undirected graph and added self-loops, which are precomputed and saved.

D ADDITIONAL EXPERIMENTAL RESULTS

D.1 Applicability and Performance on Multi-layer Models

Our theoretical analysis is primarily concerned with bounding the error in approximating the polynomial of the Laplacian matrix and the subsequent node representations following the linear transformation stages. Importantly, these results are robust and maintain their validity across various model structures and irrespective of the number of layers involved. This independence ensures that our theoretical conclusions remain applicable when implementing Laplacian sparsification in any polynomial-based spectral methods. Furthermore, although stacking spectral convolution layers is generally not recommended (for extra computational overhead and unstably increasing performance), our findings are persuasive and retain their relevance in such configurations.

In our study, we present the comparative results of multiple configurations of GPR-GNN, including standard GPR-GNN, GPR-GNN enhanced with two layers, GPR-GNN incorporating Laplacian sparsification, and GPR-GNN that combines both multilayer enhancements and sparsification in Table 5. As our results show, the 2-layer GPR-GNN entangling with Laplacian sparsification also consistently yields improved performance compared to the simply stacked 2-layer GPR-GNN. Moreover, some result entries of 2-layer GPR-GNN with Laplacian sparsification surpass all the other compared models, demonstrating the superiority of our methods.

D.2 Comparison among Detached Models

In this section, we restate some of the results in Table 1 and provide more experimental tests among GPR-GNN and its variants to argue the potential performance reductions of the detaching trick. The results are proposed in Table 6 and Table 7

We first examine the comparison of GCN and SGC. SGC can be considered as the detached version of GCN, which first executes a K-hop graph propagation ($K = 2$ in practice), followed by the linear layers. Our empirical findings verify that the detaching manner does exert a negative influence on the performance of GCN. Despite SGC exhibiting occasional performance improvements on specific small-scale datasets, the overall results are still distant from the powerful baseline GPR.

The variant known as Detached GPR-GNN modifies the original by omitting the linear transformation during graph propagation. Although GPR-GNN-detach retains many of GPR-GNN's essential characteristics—thereby significantly outperforming traditional GNNs—it also simplifies the original architecture. This simplification could lead to potential performance reductions compared to GPR-GNN, as our proposed results show.

D.3 Additional Comparisons with More Models.

In this section, we include two more baselines named ClusterGCN [9] and H2GCN [57], which are known for their superiority in scalability and unique design for heterophilous graphs. The results are shown in Table 8. Since we aim to demonstrate that our approach either matches or exceeds the performance of standard spectral GNNS, these models are not aligned with the track of our proposed ones. We can still show outstanding performance over those scalable methods and spatial ones.

D.4 Analysis on Time, Space, Executability, and Sampling Numbers

Time, space, and executability. First, we present some results of our time and memory efficiency comparisons on relatively smaller datasets in Table 9. These findings will illustrate the enhancements our model offers, making SGNN training viable in wide scenarios.

For runtime and memory consumption, we provide some metrics from tests conducted on the datasets MAG-scholar-C and Ogbn-papers100M. Compared to traditional SGNNs, our model significantly reduces memory overhead to $O(Ln_bF + LF^2 + F_iF + m)$ per batch, where n_b is the number of nodes per batch, and F_i denotes the dimensions of the original node embeddings. This reduction

Table 4: Detailed information about used datasets.

Dataset	Nodes	Edges	Features	Classes	$\mathcal{H}(\mathcal{G})$	scale
Cora	2,708	5,278	1,433	7	0.810	small
Citeseer	3,327	4,552	3,703	6	0.736	small
PubMed	19,717	44,324	500	3	0.802	small
Actor	7,600	30,019	932	5	0.219	small
Wisconsin	251	515	1,703	5	0.196	small
Cornell	183	298	1,703	5	0.305	small
Texas	183	325	1,703	5	0.108	small
Photo	7,650	119,081	745	8	0.827	small
Computers	13,752	245,861	767	10	0.777	small
Twitch-de	9,498	153,138	2,514	2	0.632	small
Ogbn-arxiv	169,343	1,166,243	128	40	0.655	medium
Twitch-gamers	168,114	6,797,557	7	2	0.554	medium
Penn94	41,554	1,362,229	4,814	2	0.489	medium
Ogbn-papers100M	111,059,956	1,615,685,872	128	172	-	large
MAG-Scholar-C	10,541,560	132,609,997	2,784,240	8	-	large

Table 5: The results of GPR-GNN, and its multi-layer / Laplacian sparsification variants. The models with suffix "-2L" represent the models are stacked for 2 layers, and those with suffix "-LS" represent the models with Laplacian sparsification.

Dataset	Cora	Citeseer	PubMed	Actor	Wisconsin	Cornell	Texas	Photo	Computers	Twitch-de
$\mathcal{H}(G)$	0.810	0.736	0.802	0.219	0.196	0.305	0.108	0.827	0.777	0.632
GPR-GNN	88.80	81.57	90.98	40.55	91.88	89.84	**92.78**	95.10	89.69	**73.90**
GPR-GNN-LS	**89.31**	**81.65**	90.95	41.82	93.63	**91.14**	92.62	**95.30**	90.47	73.71
GPR-GNN-2L	88.09	80.23	**91.31**	42.08	92.38	85.90	87.87	94.24	90.24	73.49
GPR-GNN-2L-LS	88.82	80.60	90.98	**42.82**	**96.75**	89.51	91.80	95.10	**90.57**	73.88

Table 6: Part of the experimental results in Table 1, including the comparison between GCN and SGC.

Dataset	Cora	Citeseer	PubMed	Actor	Wisconsin	Cornell	Texas	Photo	Computers	Twitch-de
$\mathcal{H}(G)$	0.810	0.736	0.802	0.219	0.196	0.305	0.108	0.827	0.777	0.632
GCN	87.78	81.50	87.39	35.62	65.75	71.96	77.38	93.62	88.98	73.72
SGC	87.24	81.53	87.17	34.40	67.38	70.82	79.84	93.41	88.61	73.70
Δ	-0.54	+0.03	-0.22	-1.22	+1.63	-1.14	+2.46	-0.21	-0.37	-0.02

Table 7: The results of GPR-GNN, GPR-GNN with Laplacian sparsification and detached GPR-GNN.

Dataset	Cora	Citeseer	PubMed	Actor	Wisconsin	Cornell	Texas	Photo	Computers	twitch-de
$\mathcal{H}(G)$	0.810	0.736	0.802	0.219	0.196	0.305	0.108	0.827	0.777	0.632
GPR-GNN-LS	**89.31**	**81.65**	90.95	**41.82**	**93.63**	**91.14**	92.62	**95.30**	**90.47**	73.71
GPR-GNN	88.80	81.57	**90.98**	40.55	91.88	89.84	**92.78**	95.10	89.69	**73.90**
GPR-GNN-detached	88.13	80.96	90.96	40.10	88.88	89.83	85.08	95.12	89.36	73.17
Δ	-0.67	-0.61	-0.02	-0.45	-3.00	-0.01	-7.70	+0.02	-0.33	-0.73

enables the practical training of SGNNs on the Ogbn-papers100M dataset using Laplacian sparsification.

Additionally, unlike detached SGNNs, which face infeasibility issues due to the high memory demands of propagating node embeddings on CPU ($O(KnF_i)$ on the MAG-scholar-C dataset), our method reduces this overhead and makes training feasible on such large-scale datasets. This transition from impractical to applicable training scenarios underlines our model's improvements in memory management and computational efficiency. For more information about time and memory, we recommend the readers refer

Table 8: The results of GPR-GNN, GPR-GNN with Laplacian sparsification, ClusterGCN, and H2GCN.

Dataset	Cora	Citeseer	PubMed	Actor	Wisconsin	Cornell	Texas	Photo	Computers	Penn94
$\mathcal{H}(G)$	0.810	0.736	0.802	0.219	0.196	0.305	0.108	0.827	0.777	0.470
GPR-GNN	88.80	81.57	**90.98**	40.55	91.88	89.84	**92.78**	89.69	95.10	82.92
GPR-GNN-LS	**89.31**	**81.65**	90.95	**41.82**	**93.63**	**91.14**	92.62	**90.47**	**95.30**	**85.00**
ClusterGCN	87.45	79.66	86.52	29.66	61.88	56.72	65.08	87.11	93.17	81.75
H2GCN	-	-	87.78	34.49	87.50	86.23	85.90	-	-	81.31

Table 9: The time and GPU memory consumption results of APPNP and APPNP-LS on several datasets.

Time per epoch (ms)	Cora	Computers	Twitch-gamer	Penn94
APPNP	6.50	7.59	19.99	13.75
APPNP-LS	5.58	5.83	16.73	12.42
GPU memory (GB)	Cora	Computers	Twitch-gamer	Penn94
APPNP	0.034	0.144	0.976	1.725
APPNP-LS	0.034	0.127	0.845	1.821

to the time and space complexity analysis in Appendix A.4 for a comprehensive understanding.

To demonstrate the universality of our methods in large-scale tasks, we present Table 10 to show the excitability of different models on the Ogbn-papers100M and MAG-scholar-C datasets, along with the reasons for any model's inability to execute.

Sampling Numbers. The relationship between approximation similarity—controlled by sampling numbers—and final GNN performance is complex. While insufficient sampling numbers can disrupt the graph filter significantly, leading to unacceptable inaccuracies, increasing the number of samples does not invariably enhance performance. In some cases, an optimal level of randomness can actually mitigate the effects of noise inherent in real-world datasets. This interaction is evidenced by the variability in optimal settings for the "−ec" parameter. Notably, the best parameters do not uniformly include "−ec=10", indicating that more sampling is not always beneficial.

We present some results from our ablation study in Table 11, which investigates the relationship between approximation similarity and the ultimate performance of GNN models. We have selected the GPR-GNN-LS model as our primary focus for this study. As the results show, when "−ec" is at a low level (<1), the performance increases significantly as "−ec" increases. The performance then stabilizes without severe fluctuations, no matter how the growth of "−ec" strengthens the similarity of approximation. The findings aim to clarify how different settings of the "−ec" parameter influence model effectiveness, providing insights that could guide the optimization of sampling strategies in practical GNN applications. In reality, we may limit the "−ec" to a reasonable range, such as $[1, 10]$, and perform hyper-parameter tuning, similar to how the "learning rate" is selected.

E EXPERIMENT DETAILS

E.1 Codes

Our codes are released at https://anonymous.4open.science/r/SGNN-LS-release-B926.

E.2 Configurations

Here we list the detailed information on the experimental platform and the environment we deployed.

- Operating System: Red Hat Linux Server release 7.9 (Maipo).
- CPU: Intel(R) Xeon(R) Gold 8358 64C@2.6GHz.
- GPU: NVIDIA A100 40GB PCIe.
  GPU: NVIDIA A100 80GB PCIe for the experiment on Ogbn-papers100M.
- GPU Driver: 525.125.06.
- RAM: 512G.
- Python: 3.10.6.
- CUDA toolkit: 11.3.
- Pytorch: 1.12.1.
- Pytorch-geometric: 2.1.0.

E.3 Details for small-scale experiments.

For all the datasets involved in Table 1, we conduct a full-supervised node classification experiment. Except for Twitch-de, all the datasets are randomly divided into 10 splits with the commonly adapted training/validation/test proportions of 60%/20%/20%. For the Twitch-de, we use the default 5 splits proposed by PyG and propose the average performance of all the datasets.

We limit the hidden size to 64 and the layers of MLP to 2 for a relatively fair comparison among all the tested models. We propose the best hyperparameters of our Laplacian sparsification extended models in Table 12, 13, 14, and 15 for full reproducibility.

E.4 Details for large-scale experiments

For all the datasets involved in Table 2 except MAG-scholar-C, we conduct a supervised node classification experiment. Since the proportion of training nodes in Ogbn-papers100M is small, we consider it a semi-supervised task. All the datasets here have the default splits provided by PyG. For Twitch-gamers and Penn94, we propose the average accuracy of all the default splits. For OGB datasets, we repeat the experiment on the single split 5 times with different random seeds and report the average accuracy.

For MAG-scholar-C, we continue to use the academic settings provided in PPRGo, which sets the size of the training/validation sets to 105415 nodes. We repeat the experiment on the randomly

Table 10: The executability of different models on the Ogbn-papers100M and MAG-scholar-C.

	Ogbn-papers100M	Reasons for inability to execute	MAG-scholar-C	Reasons for inability to execute
GCN, GCNII, GAT (traditional GNNs)	No	GPU OOM during full-batch message-passing	No	GPU OOM during full-batch message-passing
SGC with precomputed graph propagation	Yes	/	No	OOM during full-batch message-passing on CPU
APPNP, GPRGNN, BernNet, ChebNetII, JacobiConv, FavardGNN (polynomial-based spectral GNNs)	No	GPU OOM during full-batch message-passing	No	GPU OOM during full-batch message-passing
Detached spectral GNNs with precomputed graph propagation	Yes	/	No	OOM during full-batch message-passing on CPU
LazyGNN	No	Error while partitioning the graph	Partial	Executable under extreme parameter settings. It takes a few days to yield unreliable results.
LMCGCN	No	Error while partitioning the graph	Partial	Executable under extreme parameter settings. It takes extra time and memory to maintain historical embeddings.
PPRGo	No	Error while calculating the approximation matrix	Yes	/
Spectral GNNs with our proposed Laplacian sparsification method.	Yes	/	Yes	/

Table 11: Experimental results of investigating the impact of hyper-parameter "–ec".

–ec	0.01	0.05	0.1	0.5	1	2	3	4	5	6	7	8	10
PubMed	87.30	88.96	89.50	90.49	90.80	90.95	90.98	90.84	90.88	**91.01**	90.99	90.94	90.95
Twitch-de	70.46	71.20	72.50	72.85	**73.40**	73.19	73.15	73.31	73.17	73.12	73.19	73.22	73.26

Table 12: The hyper-parameters of APPNP-LS on small-scale datasets.

Dataset	learning rate	weight decay	dropout=dprate	α	K	ec
Cora	0.05	0.0005	0.5	0.1	5	10
Citeseer	0.01	0	0.8	0.5	10	10
PubMed	0.05	0.0005	0.0	0.5	10	10
Actor	0.01	0.0005	0.8	0.9	2	10
Wisconsin	0.05	0.0005	0.5	0.9	10	20
Cornell	0.05	0.0005	0.5	0.9	5	20
Texas	0.05	0.0005	0.8	0.9	2	20
Photo	0.05	0	0.5	0.5	5	10
Computers	0.05	0	0.2	0.1	2	10
Twitch-de	0.01	0.0005	0.5	0.1	2	10

generated splits with 5 fixed random seeds and report the average accuracy.

We limit the hidden size to 128 and the layers of MLP to 3 at most for a relatively fair comparison among all the tested models. We propose the best hyperparameters for our Laplacian sparsification extended models in Table 16 and 17 for full reproducibility.

Table 13: The hyper-parameters of GPR-LS on small-scale datasets.

Dataset	lr=prop_lr	wd=prop_wd	dropout=dprate	α	K	ec
Cora	0.05	0.0005	0.8	0.9	10	3
Citeseer	0.01	0	0.2	0.5	10	10
PubMed	0.05	0.0005	0.2	0.5	5	10
Actor	0.05	0.0005	0.5	0.5	10	3
Wisconsin	0.05	0.0005	0.8	0.9	10	10
Cornell	0.05	0.0005	0.5	0.9	2	1
Texas	0.05	0.0005	0.8	0.5	10	10
Photo	0.05	0.0005	0.8	0.1	2	10
Computers	0.05	0	0.5	0.1	10	10
Twitch-de	0.01	0.0005	0.8	0.1	2	20

Table 14: The hyper-parameters of Jacobi-LS on small-scale datasets.

Dataset	lr=prop_lr	wd=prop_wd	dropout=dprate	paraA	paraB	K	ec
Cora	0.05	0.0005	0.8	0.5	0.5	10	3
Citeseer	0.01	0	0	0.5	0.5	2	1
PubMed	0.05	0.0005	0.2	1.0	0.5	5	3
Actor	0.01	0.0005	0.5	0.5	0.5	10	1
Wisconsin	0.05	0.0005	0.8	0.5	0.5	2	10
Cornell	0.05	0.0005	0.8	0.5	0.5	2	1
Texas	0.05	0.0005	0.5	0.5	1.0	10	1
Photo	0.05	0	0.5	1.0	0.5	10	3
Computers	0.05	0.0005	0.2	1.0	1.0	2	3
Twitch-de	0.01	0.0005	0.8	0.5	1.0	5	10

Table 15: The hyper-parameters of Favard-LS on small-scale datasets.

Dataset	lr=prop_lr	wd=prop_wd	dropout=dprate	K	ec
Cora	0.05	0.0005	0.8	2	1
Citeseer	0.01	0.0005	0	2	1
PubMed	0.01	0.0005	0.2	5	3
Actor	0.05	0.0005	0	2	1
Wisconsin	0.05	0.0005	0.8	5	20
Cornell	0.05	0.0005	0.2	5	1
Texas	0.05	0.0005	0.8	10	10
Photo	0.05	0.0005	0.8	2	10
Computers	0.05	0.0005	0.8	5	10
Twitch-de	0.01	0	0.8	10	10

Table 16: The hyper-parameters of APPNP-LS on medium- and large-scale datasets.

Dataset	learning rate	weight decay	dropout=dprate	α	K	ec
Ogbn-arxiv	0.01	0	0	0.1	5	10
Twitch-gamer	0.01	0	0	0.5	2	10
Penn94	0.01	0	0.5	0.9	10	10
Ogbn-papers100M	0.01	0	0.2	0.1	2	-
MAG-scholar-C	0.01	0	0.5	0.5	2	-

Table 17: The hyper-parameters of GPR-LS on medium- and large-scale datasets.

Dataset	lr=prop_lr	wd=prop_wd	dropout=dprate	α	K	ec
Ogbn-arxiv	0.01	0	0.2	0.5	10	10
Twitch-gamer	0.05	0	0.5	0.1	5	10
Penn94	0.01	0	0.5	0.1	2	10
Ogbn-papers100M	0.01	0	0.2	0.9	2	-
MAG-scholar-C	0.01	0	0.5	0.5	2	-