
GENERALIZED LEARNING OF COEFFICIENTS IN SPECTRAL GRAPH CONVOLUTIONAL NETWORKS

Mustafa Coskun
 Artificial Intelligence Department
 Ankara University
 Ankara, 06033, TURKEY
 coskunmustafa@ankara.edu.tr

Anath Grama
 Computer Science Department
 Purdue University
 West Lafayette, IN 47906, USA
 ayg@cs.purdue.edu

Mehmet Koyuturk
 Department of Data and Computer Science
 Case Western Reserve University
 Cleveland, OH 44106, USA
 mehmet.koyuturk@case.edu

September 10, 2024

ABSTRACT

Spectral Graph Convolutional Networks (GCNs) have gained popularity in graph machine learning applications due, in part, to their flexibility in specification of network propagation rules. These propagation rules are often constructed as polynomial filters whose coefficients are learned using label information during training. In contrast to learned polynomial filters, explicit filter functions are useful in capturing relationships between network topology and distribution of labels across the network. A number of algorithms incorporating either approach have been proposed; however the relationship between filter functions and polynomial approximations is not fully resolved. This is largely due to the ill-conditioned nature of the linear systems that must be solved to derive polynomial approximations of filter functions. To address this challenge, we propose a novel Arnoldi orthonormalization-based algorithm, along with a unifying approach, called G-ARNOLDI-GCN that can efficiently and effectively approximate a given filter function with a polynomial. We evaluate G-ARNOLDI-GCN in the context of multi-class node classification across ten datasets with diverse topological characteristics. Our experiments show that G-ARNOLDI-GCN consistently outperforms state-of-the-art methods when suitable filter functions are employed. Overall, G-ARNOLDI-GCN opens important new directions in graph machine learning by enabling the explicit design and application of diverse filter functions. Code link: <https://anonymous.4open.science/r/GArnoldi-GCN-F7E2/README.md>

1 Introduction

Spectral Graph Convolutional Networks (Spectral-GCNs) have attracted significant attention in various graph representation learning tasks, including node classification [1], link prediction [2], and applications like drug discovery [3, 4]. Spectral-GCNs utilize spectral convolution or spectral graph filters, operating in the spectral domain of the graph Laplacian matrix or normalized adjacency matrix [1, 5] to address the "feature-over-smoothing" problem by isolating the propagation scheme from neural networks. Spectral GCNs enable distribution of label/ feature information beyond neighboring nodes, as in Spatial GCNs [6], to distant nodes [7]. A key element of Spectral GCNs is spectral graph convolutions or filters, which control the distribution of processed label/feature information. These convolutions assign weights to contributions from each hop in the graph, and the challenge in Spectral GCNs revolves around finding optimal weights.

Spectral GCNs are grouped into two major categories: predetermined spectral graph convolution/filters [7–9] and learnable spectral graph convolution/filters, such as ChebNet [10], CayleyNet [11], GPR-GNN [5], BernNet [1], JacobiCon [12]. Applications in the former class rely on domain/ data knowledge to assign polynomial coefficients to design custom filters, whereas those in the latter class learn coefficients using training labels. Despite the effectiveness of current Spectral GCNs, their methodologies often come with inherent challenges. Specifically, these models either depend on predetermined coefficients, embodying an implicit filter (e.g., APPNP [7], GNN-LP/HP [9], etc.), or utilize a specific polynomial and its update rules without establishing a connection with any filter function [1, 10–12]. In such cases, models rely on neural networks to learn an arbitrary filter based on label/feature information and polynomials’ update rules. However, the lack of a principled link between a filter and the selected polynomial introduces uncertainties regarding the assertion of effectively learning an arbitrary filter.

Our focus in this paper is on the design of polynomials as proxies for implementing explicitly specified filter functions. These polynomials specify the propagation rules, enabling interpretation and effective design of filter functions. Our goal is to establish the mathematical foundations to enable their joint exploration for design of effective and computationally efficient filters. We achieve this goal through the following major contributions:

1. We approximate a given filter to a polynomial by formulating and solving the corresponding Vandermonde linear system. We show that the exponential condition number of the Vandermonde matrix leads to the computation of invalid polynomial coefficients, resulting in poor polynomial approximations.
2. We demonstrate that an alternate QR decomposition for the Vandermonde matrix can be computed through an Arnoldi-process. This approach produces precise polynomial approximations for any filter.
3. Using polynomial approximations obtained via Arnoldi, we introduce the ARNOLDI-GCN algorithm, eliminating the need for polynomial update rules in current methods. This offers a powerful new propagation scheme independent of the polynomial.
4. Finally, we extend the ARNOLDI-GCN approach to G-ARNOLDI-GCN by refining polynomial coefficients computed by ARNOLDI-GCN within neural networks using label information, thereby broadening its applicability within the context of neural networks.

We provide comprehensive theoretical underpinnings for our findings and empirically demonstrate the efficiency of our approaches across diverse real-world benchmark datasets and problems. First, we approximate eight representative filter functions, including simple random walks and intricate band-pass rejection filters, using Equispaced, Chebyshev, Legendre, and Jacobi polynomial samples. We then assess node classification performance of Spectral GCNs that are built using these approximations. Our results show that our Arnoldi-based approximation consistently outperforms the direct solution of the Vandermonde system in generating precise polynomial approximations across all filter functions. Rigorous experiments on 15 benchmark datasets, covering semi-supervised and fully supervised learning tasks, reveal that our algorithms significantly advance state-of-the-art methods in node classification performance.

2 Background and Related Work

Spectral Graph Convolutional Networks. We denote an undirected graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set \mathcal{V} and edge set \mathcal{E} , where the number of nodes is $|\mathcal{V}| = n$. In the context of node classification, we are given an $n \times m$ feature matrix \mathbf{X} , where m denotes the number of features and $\mathbf{X}(i, j)$ represents the value of the j th feature for the i th node. We are also given an $n \times c$ -dimensional label matrix \mathbf{Y} , where c denotes the number of classes, and $\mathbf{Y}(i, j)$ indicates whether node i belongs to class j . Our goal is to learn a neural network model $\mathbf{Y} = f(\mathcal{G}; \mathbf{X})$, where f utilizes topological relationships between nodes in \mathcal{G} to enhance the generalizability of the machine learning model.

Spectral GCNs use spectral filter functions to propagate signals across the graph. The signals include input features and latent features that are computed at intermediate layers of the neural network. For the sake of generality, in the following discussion, we denote quantities propagated across the network as graph signal $\mathbf{x} \in \mathbb{R}^n$, where $\mathbf{x}(i)$ denotes the graph signal at node i .

Spectral Filter Functions. Let \mathbf{A} denote the adjacency matrix of graph \mathcal{G} and \mathbf{D} denote the diagonal degree matrix. We represent the degree-normalized adjacency matrix as $\mathbf{P} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ and graph Laplacian matrix as $\mathbf{L} = \mathbf{I} - \mathbf{P}$, where \mathbf{I} denotes the identity matrix. Let $\mathbf{L} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T$ be the eigendecomposition of \mathbf{L} , where $\mathbf{\Lambda} = \text{diag}[\lambda_1, \cdot, \cdot, \cdot, \lambda_n]$.

Spectral GCNs create the spectral graph convolutions, or spectral filters, in the domain of the graph Laplacian [5]. Namely:

$$\mathbf{y} = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^T \mathbf{x}, \quad (1)$$

where $g(\mathbf{\Lambda}) = \text{diag}[g(\lambda_1), \cdot, \cdot, \cdot, g(\lambda_n)]$. Here, \mathbf{y} represents the vector obtained by filtering graph signal \mathbf{x} . The function $g(\lambda)$, called a *spectral filter*, transforms the eigenvalues of the Laplacian matrix to suitably shape signal propagation

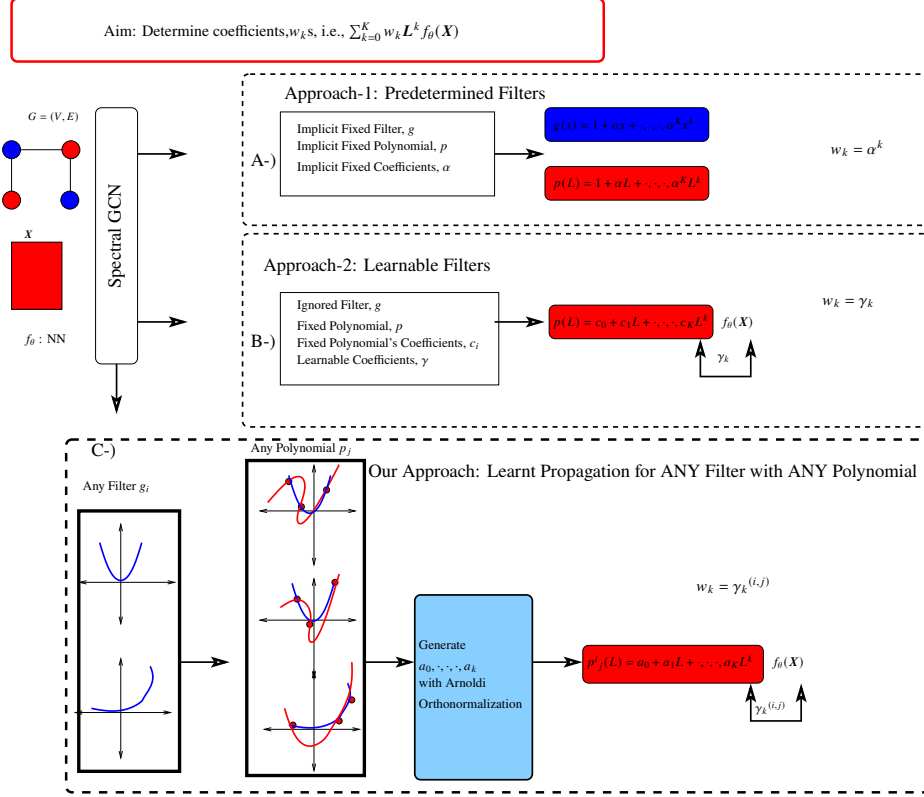


Figure 1: Workflow of proposed Arnoldi-Based Generalized Spectral GCN.

over the graph. Computing the eigendecomposition in Equation 1 is computationally expensive for large matrices, motivating the use of *polynomials* for approximation:

$$\mathbf{y} = \mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^T \mathbf{x} \approx \sum_{k=0}^K w_k \mathbf{L}^k \mathbf{x} \equiv \sum_{k=0}^K c_k \mathbf{P}^k \mathbf{x}, \quad (2)$$

where K is a hyperparameter that specifies the degree of the polynomial used in the approximation. An important design criterion in Spectral GCNs is the choice of coefficients w_k or c_k . In the context of random walks, K specifies the extent of propagation in terms of path length and coefficients w_k or c_k correspond to the relative weight of paths of length k .

2.1 General Formulation of Spectral GCNs

Given a filtering function ($g(\lambda)$) or a polynomial approximation (Equation 2), the general setting for Spectral GCNs is as follows:

$$\mathbf{Y} = \text{softmax}(\mathbf{Z}), \mathbf{Z} = \sum_{k=0}^K w_k \mathbf{H}^{(k)}, \quad (3)$$

$$\mathbf{H}^{(k)} = \mathbf{L} \mathbf{H}^{(k-1)}, \mathbf{H}^{(0)} = f_\theta(\mathbf{X})$$

where $\mathbf{X} \in \mathbb{R}^{n \times m}$ denotes the feature matrix, f_θ denotes neural network with parameters θ , and \mathbf{Y} denotes the label matrix.

Current spectral GCNs can be grouped into two categories based on how they choose the coefficients of the polynomials: Predetermined Graph Convolution and Learnable Graph Convolution.

2.1.1 Predetermined Graph Convolution

This class of spectral GCNs fix the weights a-priori, e.g., $c_k = \alpha^k$ where $\alpha \in (0, 1)$. For example, **APPNP** [7] uses an implicit filter function, $g : [-\alpha, \alpha] \rightarrow \mathbb{R}$ with $g(\omega) = \frac{1-\alpha}{(1-\omega)}$, which is approximated by the polynomial

$p(\omega) = 1 + \alpha\omega + \alpha^2\omega^2 + \dots$. Then, starting with $\mathbf{H}^{(0)} = f_\theta(\mathbf{X})$, APPNP defines the following Spectral-GCN:

$$\mathbf{Z}_{\text{APPNP}} = \sum_{k=0}^K \alpha^k \mathbf{H}^{(k)}, \mathbf{H}^{(k)} = \tilde{\mathbf{P}}\mathbf{H}^{(k-1)}, \quad (4)$$

That is, APPNP sets $c_k = \alpha^k$. Similarly, **GNN-LF/HF** [9] and **SGC** [13] use an implicit filter function and approximate this function using fixed coefficients.

2.1.2 Learnable Graph Convolution

These spectral GCNs simultaneously learn the coefficients of the polynomial alongside $f_\theta(\mathbf{X})$ by leveraging label information in training data. As an example, **GPR-GNN** [5] generalizes **APPNP** and learns γ_k s along with $f_\theta(\mathbf{X})$, instead of fixing them to α^k as in APPNP [7], i.e.:

$$\mathbf{Z}_{\text{GPR-GNN}} = \sum_{k=0}^K \gamma_k \mathbf{H}^{(k)}, \mathbf{H}^{(k)} = \tilde{\mathbf{P}}\mathbf{H}^{(k-1)}, \quad (5)$$

where γ_k s are learnable parameters.

ChebNet [10] replaces the polynomial used in the approximation of Equation 5 with a Chebyshev polynomial. Since Chebyshev polynomials converge more rapidly to the function that is being (implicitly) approximated, this process effectively increases the depth of the propagation. ChebNet [10] can be formulated as:

$$\begin{aligned} \mathbf{Z}_{\text{ChebNet}} &= \sum_{k=0}^K \gamma_k \mathbf{H}^{(k)}, \\ \mathbf{H}^{(k)} &= 2\tilde{\mathbf{P}}\mathbf{H}^{(k-1)} - \mathbf{H}^{(k-2)}, \end{aligned} \quad (6)$$

where $\mathbf{H}^{(0)}, \mathbf{H}^{(1)} = f_\theta(\mathbf{X})$. Similarly, one can replace the polynomial in Equation 6 with any polynomial, provided that the update rules of the chosen polynomial are preserved. In the Spectral GCNs literature, this flexibility allows the creation of various Spectral GCN algorithms. Examples include CayleyNet [11], GPR-GNN [5], BernNet [1], and JacobiCon [12], respectively utilizing Cayley, monomial, Bernstein, and Jacobi Polynomials.

3 Methods

Spectral GCNs can be more versatile and adaptive if they are used with explicit filter functions. For predetermined graph convolution, explicit filter functions enable choosing the filter to suit the topology of graph, the nature of the learning task (e.g., homophilic vs. heterophilic graphs), and domain knowledge relating to distribution of labels. For learnable graph convolution, explicit filter functions enable initialization of the coefficients to values that suit the learning task. The challenge in the application of explicit filter functions is in computing polynomial approximations to these filter functions. With **ARNOLDI-GCN** (for predetermined graph convolution) and **G-ARNOLDI-GCN** (for learnable graph convolution), we address this challenge by using Arnoldi orthonormalization to solve the key challenge associated with ill-conditioned systems that result from the polynomial approximation of filter functions. The workflow of **ARNOLDI-GCN** and **G-ARNOLDI-GCN** is shown in Figure 1.

We begin by defining representative filter functions and polynomial samples, emphasizing that the application of our algorithms extend beyond these specific choices. Subsequently, we provide theoretical underpinnings and methodological details of our proposed algorithms.

3.1 Explicit Filter Functions for Spectral GCNs

We focus on eight representative filter functions, with a view to demonstrating and comparing the utility of explicitly specified filter functions in the context of spectral GCNs. These represent four simple and four complex filter functions, establishing propagation rules for homophilic and heterophilic graphs. The simple filters we use correspond to random walks on the graph and operate on the eigen-decomposition of the degree-normalized adjacency matrix (\mathbf{P}). The complex filters operate on the eigen-decomposition of the graph Laplacian (\mathbf{L}). The range of a filter function is therefore determined by the bounds on the eigenvalues of the matrix it operates on. Specifically, we consider the following filter functions, where $0 < \alpha < 1$ is a hyperparameter:

Simple Filters: ($\omega \in (-1, 1)$)

$$\text{Scaled Random Walk: } g_0(\omega) = \frac{1 - \alpha}{1 - \omega} \quad (7)$$

$$\text{Random Walk: } g_1(\omega) = \frac{1}{1 - \omega} \quad (8)$$

$$\text{Self-Depressed RW: } g_2(\omega) = \frac{\omega}{1 - \omega} \quad (9)$$

$$\text{Neighbor-Depressed RW: } g_3(\omega) = \frac{\omega^2}{1 - \omega} \quad (10)$$

$$\text{Low Pass : } g_4(\omega) = e^{-10\omega^2} \quad (11)$$

$$\text{High Pass : } g_5(\omega) = 1 - e^{-10\omega^2} \quad (12)$$

$$\text{Band Pass : } g_6(\omega) = e^{-10(\omega-1)^2} \quad (13)$$

$$\text{Band Rejection : } g_7(\omega) = 1 - e^{-10(\omega-1)^2} \quad (14)$$

Complex Filters: ($\omega \in (0, 2]$)

These functions are used to filter the spectra of the graph Laplacian (Equation 1) to facilitate propagation of the graph signal in accordance with the rules specified by the filter [14]. However, since it is computationally expensive to compute the eigen-decomposition of the Laplacian, we compute and use a polynomial approximation to the given filter (Equation 2). It is important to note that the framework we propose is not limited to explicitly designed filter functions. Since the below polynomial approximation scheme is based on sampling the function, it can be used for any filter function that can be sampled in the range of eigenvalues of the graph Laplacian (or the normalized adjacency matrix).

3.2 Polynomial Approximation of Filter Functions

For a given filter $g(\omega)$ and integer K , our objective is to compute a polynomial of degree K that best approximates $g(\omega)$:

$$P(\omega) = \sum_{k=0}^K a_k \omega^k, \quad (15)$$

That is, we seek to compute the coefficients a_k for $k = 0$ to K such that $P(\omega)$ provides a good approximation to $g(\omega)$. We use these coefficients to implement the spectral GCN associated with filter $g(\omega)$, using the approximation in Equation (2).

Well-established techniques in numerical analysis consider r distinct samples taken from the range of ω , denoted $\omega_1, \omega_2, \dots, \omega_r$, where $r = K$. In this study, we consider four sampling techniques, namely equispaced samples (Eq.), Chebyshev sampling (Ch.), Legendre sampling (Le.), and Jacobi sampling (Ja.). For the range $[l, u]$ of the filter function, the values ω_k for $1 \leq k \leq r$ are sampled from this range by each of these techniques as follows:

$$\text{Eq. : } \omega_k = l + k \frac{u - l}{r + 1} \quad (16)$$

$$\text{Ch. : } \omega_k = \frac{u + l}{2} + \frac{u - l}{2} \cos\left(\frac{2k - 1}{2r} \pi\right) \quad (17)$$

$$\text{Le. : } \int_{-1}^1 p(\omega) d\omega = \sum_{k=1}^r p(\omega_k) \quad (18)$$

$$\text{Ja. : } \int_{-1}^1 (1 + \omega) p(\omega) d\omega = \sum_{k=1}^r (1 + \omega_k) p(\omega_k) \quad (19)$$

Here, $p(\omega)$ denotes the polynomial $1 + \omega + \omega^2 + \dots$. The points for Legendre and Jacobi sampling are computed numerically, using the Gauss Quadrature procedure, since no analytical solution exists for integral-formed polynomials. [15]. Once the sampled points in the interval $[-1, 1]$ are computed, we scale and shift them to the desired interval $[l, u]$.

The next step in computing the approximation involves evaluating the value of the filter functions at these sampled points, yielding $g(\omega_1), g(\omega_2), \dots, g(\omega_r)$ to align the polynomials with the filters. Once these values are computed, expressing the right side of Equation (15) in Vandermonde matrix form, we obtain:

$$\mathbf{V} = \begin{bmatrix} 1 & \omega_1 & \omega_1^2 & \dots & \omega_1^K \\ 1 & \omega_2 & \omega_2^2 & \dots & \omega_2^K \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_r & \omega_r^2 & \dots & \omega_r^K \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_K \end{bmatrix} = \begin{bmatrix} g(\omega_1) \\ g(\omega_2) \\ \dots \\ g(\omega_r) \end{bmatrix} \quad (20)$$

A commonly encountered problem in computing polynomial approximations is that the Vandermonde matrix \mathbf{V} is typically ill-conditioned.

Definition 1 The condition number of matrix A is defined as $\kappa(A) = \|A\| \cdot \|A^{-1}\|$. If $\kappa(A) = O(1)$, the matrix is well-conditioned; otherwise it is ill-conditioned.

Theorem 1 Let $\omega_1, \dots, \omega_r$ be the samples obtained using one of the sampling techniques in Equation 16-19. If $\omega_1, \dots, \omega_r \in [-\alpha, \alpha]$ with $\alpha \in (0, 1)$, then $\kappa(\mathbf{V}) = \|\mathbf{V}\| \cdot \|\mathbf{V}^\dagger\| \geq 2^{r-1} \left(\frac{1}{\alpha}\right)^r$. If $\omega_1, \dots, \omega_r \in (0, 2]$, then $\kappa(\mathbf{V}) = \|\mathbf{V}\| \cdot \|\mathbf{V}^\dagger\| \geq 2^{r-2}$.

The proof of this theorem is provided in the Appendix. A consequence of this theorem is that direct solution of (20) leads to inaccurate coefficients. For this reason, we solve this system using QR decomposition. Let $\mathbf{V} = \mathbf{Q}_V \mathbf{R}_V$ be the QR decomposition of \mathbf{V} and \mathbf{Q}_V^\dagger denote the pseudo-inverse of \mathbf{Q}_V . A solution for Equation (20) can be obtained as:

$$a = (\mathbf{Q}_V^\dagger)g \quad (21)$$

Algorithm 1 Compute coefficients with Arnoldi/Lanczos orthonormalization on $\mathbf{\Omega}$

Input: l, u : lower and upper bounds, function g , sampling p , number of samples r , and degree K .

Sample r samples, i.e., $\omega_1, \dots, \omega_r$.

Obtain $g(\omega_1), \dots, g(\omega_r)$

Create diagonal matrix $\mathbf{\Omega}$ as $\text{diag}(\omega_1, \dots, \omega_r)$.

$\mathbf{Q} = \mathbf{1}$

for $m = 1$ **to** K **do**

$q = \mathbf{\Omega} \mathbf{Q}(:, m)$

for $L = 1$ **to** m **do**

$H(l, m) = \mathbf{Q}(:, l) \cdot \frac{q}{\sqrt{r}}$

$q = q - H(l, m) * \mathbf{Q}(:, l)$

end for

$H(l+1, l) = \frac{\|q\|}{\sqrt{r}}$

$\mathbf{Q} = [\mathbf{Q} \quad \frac{q}{H(l+1, l)}]$

end for

$a_A = \mathbf{Q}^\dagger g(\omega)$

Note that using QR decomposition does not mitigate the ill-conditioning problem by itself. To address this, we use an alternate QR-decomposition using $\mathbf{\Omega} = \text{diag}(\omega_1, \dots, \omega_r)$. In other words, let \mathbf{Q}_A denote the orthonormal basis that is computed using Arnoldi Orthogonalization on $\mathbf{\Omega}$. We compute the coefficients of the polynomial approximation as:

$$a = (\mathbf{Q}_A^\dagger)g \quad (22)$$

and show that this formulation leads to a correct solution for the linear system (Theorem 2) and produces accurate coefficients for our polynomial approximation (Theorem 3).

The procedure for computing \mathbf{Q}_A is given in Algorithm 1. Note that, since the ω_k s are real, Arnoldi orthonormalization here can be implemented using Lanczos' algorithm. Specifically, we compute orthonormal matrix \mathbf{Q}_A , tridiagonal \mathbf{T} , and an almost-zero matrix $\tilde{\mathbf{O}}$ to satisfy:

$$\mathbf{\Omega} \mathbf{Q}_A = \mathbf{Q}_A \mathbf{T} + \tilde{\mathbf{O}} \quad (23)$$

where $\mathbf{\Omega} \in \mathbb{R}^{r \times r}$, $\mathbf{Q}_A \in \mathbb{R}^{(K+1) \times r}$, $\mathbf{T} \in \mathbb{R}^{(K+1) \times (K+1)}$, and $\tilde{\mathbf{O}} \in \mathbb{R}^{(K+1) \times r}$ is all zero except its last column.

The next theorem guarantees that Arnoldi/Lanczos Orthonormalization can be used to obtain an alternative QR decomposition to original Vandermonde matrix.

Theorem 2 Given Ω , let orthonormal \mathbf{Q} , tridiagonal \mathbf{T} and almost-zero $\tilde{\mathbf{O}}$ matrices be computed using Lanczos algorithm to satisfy Equation (23). Then we can obtain a QR-decomposition for the Vandermonde matrix \mathbf{V} as $\mathbf{V}^{(*)} = \mathbf{Q}\mathbf{R}$, such that $\mathbf{V}^{(*)} = \mathbf{V}/\|\mathbf{e}\|$ and $\mathbf{R} = [\mathbf{e}_1, \mathbf{T}\mathbf{e}_1, \cdot, \cdot, \cdot, \mathbf{T}^K\mathbf{e}_1]$, where \mathbf{e}_1 denotes the $K + 1$ -dimensional vector of ones.

The proof of this theorem is provided in the Appendix. This theorem establishes that it is possible to compute an alternate orthonormal basis for the Vandermonde matrix using the Arnoldi/ Lanczos process, which can be used to solve the linear system of Equation (20) using Equation (22). However, ensuring the accuracy of these coefficients requires addressing whether \mathbf{Q}_A leads to precise computation of coefficients. Theorem 3 provides the basis for accuracy of $a_A = (\mathbf{Q}^\dagger)g$. Specifically, we establish that the condition number of $(\mathbf{Q}^\dagger\mathbf{Q})$ is close to one, ensuring the accuracy of the solution.

Theorem 3 Let $\omega_1, \cdot, \cdot, \cdot, \omega_r$ be points sampled from interval $(-1, 1)$ or $(0, 2]$ using one of the techniques shown in Equation 16-19. Let $\Omega = \text{diag}(\omega_1, \cdot, \cdot, \cdot, \omega_r)$, \mathbf{Q} be the orthonormal basis obtained by applying Arnoldi/ Lanczos orthonormalization on Ω , and \mathbf{Q}^\dagger be the pseudo-inverse of \mathbf{Q} . If the Krylov subspace used for the orthogonalization has $K = r$ dimensions, then $\kappa(\mathbf{Q}^\dagger\mathbf{Q}) \approx 1.01$.

In summary, these three theorems show that computing the QR decomposition of \mathbf{V} with Arnoldi/ Lanczos process on Ω enables us to produce accurate polynomial coefficients, since condition number of \mathbf{Q} is bounded, while performing QR decomposition directly on \mathbf{V} produces inaccurate coefficients due to the ill-conditioned nature of the matrix \mathbf{V} .

Table 1: **Datasets used in our experiments.** For each dataset, basic statistics and the characteristics of graph topology in relation to the distribution of classes are shown.

Datasets	# of Nodes	# of Edges	# of Features	# of Classes	Adj-Homophily	Label Informativeness
<i>Homophilic Graphs</i>						
Cora	2708	5278	1433	7	0.77	0.61
Citeseer	3327	4552	3703	6	0.67	0.45
Pubmed	19717	44324	500	5	0.68	0.40
Photo	7650	119081	745	8	0.78	0.67
Computer	13752	245861	767	10	0.68	0.63
<i>Small Heterophilic Graphs</i>						
Texas	183	279	1703	5	-0.26	0.24
Cornell	183	277	1703	5	-0.21	0.19
Actor	7600	26659	932	5	0.0044	0.0023
Chameleon	2277	31371	2325	5	0.0331	0.0567
Squirrel	5201	198353	2089	5	0.0070	0.0026
<i>Large Heterophilic Graphs</i>						
Roman-Empire	22662	32927	300	18	-0.05	0.11
Amazon-Ratings	24492	93050	300	5	0.13	0.04
Minesweeper	10000	39402	7	2	0.009	0.000
Tolokers	11758	519000	10	2	0.0925	0.0178
Questions	48921	153540	301	2	0.0206	0.0068

3.3 Generalized Spectral GCNs

The framework described above enables computation of accurate polynomial approximations to any explicit filter function. Using this framework, we propose two Spectral GCNs that operate with an explicitly formulated filter function: (i) **Arnoldi-GCN** for predetermined graph convolution; and (ii) **G-Arnoldi-GCN** for learnable graph convolution. These spectral GCNs are *generalized* in the sense that they can work with any filter function. Furthermore, any spectral GCN in the literature can be formulated within this framework.

Given a filter function $g(\omega) : D \rightarrow \mathbb{R}$ and integer K , let a_0, a_1, \dots, a_K denote the coefficients of the polynomial that approximates $g(\omega)$, computed using the procedure described in the previous section. If g operates on the degree-normalized adjacency matrix, then $D = (-1, 1)$. If g operates on the graph Laplacian, then $D = (0, 2]$. Using this filter function, **Arnoldi-GCN** and **G-Arnoldi-GCN** are implemented as follows:

Arnoldi-GCN

$$\begin{aligned}
 \mathbf{Y} &= \text{softmax}(\mathbf{Z}_{\text{ARNOLDI}}), \\
 \mathbf{Z}_{\text{ARNOLDI}} &= \sum_{k=0}^K a_k \mathbf{H}^{(k)}, \mathbf{H}^{(0)} = f_\theta(\mathbf{X}), \\
 \mathbf{H}^{(k)} &= \begin{cases} \tilde{\mathbf{P}}\mathbf{H}^{(k-1)}, & D = (-1, 1) \\ \tilde{\mathbf{L}}\mathbf{H}^{(k-1)}, & D = (0, 2] \end{cases}
 \end{aligned} \tag{24}$$

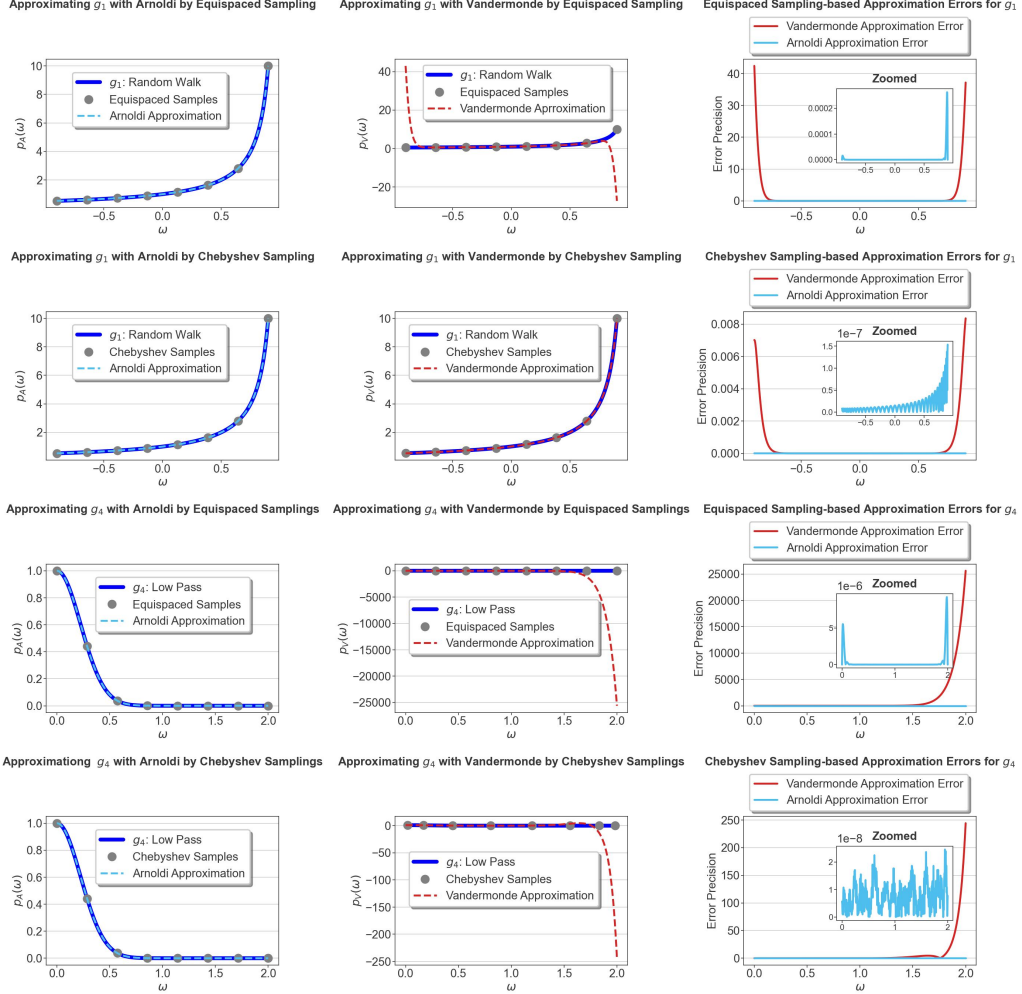
Vandermonde Versus Arnoldi for Approximating g_1 and g_4 by Equispaced and Chebyshev Samplings

Figure 2: **Approximating Random Walk and Low Pass filter functions using Equispace vs. Chebyshev sampling with direct solution to the Vandermonde system vs. Arnoldi orthonormalization.** The first/ second columns show the approximation provided by Arnoldi decomposition/ direct solution, respectively, and the third column shows the error between approximations and actual filter functions.

Here, $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{L}}$ are self-loop added versions of original normalized adjacency and Laplacian matrices, respectively. f_θ denotes the neural network with learnable parameters θ .

G-Arnoldi-GCN

$$\mathbf{Y} = \text{softmax}(\mathbf{Z}_{\text{G-ARNOLDI}}),$$

$$\mathbf{Z}_{\text{G-ARNOLDI}} = \sum_{k=0}^K \gamma_k \mathbf{H}^{(k)}, \mathbf{H}^{(0)} = f_\theta(\mathbf{X}), \quad (25)$$

where γ_k are learnable parameters that are optimized alongside θ in an end-to-end fashion by initializing $\gamma_k = a_k$.

4 Results

In this section, we systematically evaluate the performance of our proposed algorithms, ARNOLDI-GCN and G-ARNOLDI-GCN in utilizing explicit filters along with accurate polynomial approximation to improve node classification using Spectral GCNs. We first describe the datasets that contain five homophilic, five small-heterophilic, and five large heterophilic networks, as well as our experimental setup. Next, we compare the performance of ARNOLDI-GCN and G-ARNOLDI-GCN against state-of-the-art algorithms GCN [6], GAT [16], APPNP [7], ChebNet [17], JKNet [18],

Table 2: Accuracy of node classification for ARNOLDI-GCN, G-ARNOLDI-GCN, and state-of-the-art algorithms. Mean and standard deviation of five runs are reported for all experiments.

Semi-supervised on Homophilic datasets					
Method	Cora	Citeseer	Pubmed	Photos	Computers
GCN	75.01 \pm 2.19	67.57 \pm 1.33	84.17 \pm 0.33	81.81 \pm 2.63	68.58 \pm 2.49
GAT	77.22 \pm 1.99	66.42 \pm 1.19	83.32 \pm 0.47	86.66 \pm 1.34	72.38 \pm 2.67
APPNP	79.93 \pm 1.69	68.27 \pm 1.27	84.22 \pm 0.19	83.24 \pm 2.97	67.46 \pm 2.43
ChebNet	69.58 \pm 3.61	65.36 \pm 3.01	83.88 \pm 0.54	88.00 \pm 1.59	79.25 \pm 0.30
JKNet	71.31 \pm 2.65	61.36 \pm 3.94	82.92 \pm 0.56	78.25 \pm 9.29	66.43 \pm 5.69
GPR-GNN	79.65 \pm 1.62	66.92 \pm 1.58	84.21 \pm 0.54	88.55 \pm 1.29	80.73 \pm 1.49
BernNet	73.39 \pm 2.78	65.84 \pm 1.54	84.20 \pm 0.71	86.33 \pm 1.02	79.25 \pm 1.28
JacobiConv	80.02 \pm 1.05	68.23 \pm 1.32	84.32 \pm 0.65	86.41 \pm 1.05	81.54 \pm 1.12
ARNOLDI-GCN	80.25 \pm 0.43	67.81 \pm 0.39	84.02 \pm 0.29	88.25 \pm 0.43	78.81 \pm 1.13
G-ARNOLDI-GCN	82.33\pm0.35	69.88\pm0.48	85.23\pm0.25	92.46\pm0.55	83.81\pm1.07
Semi-supervised on Small Heterophilic datasets					
Method	Texas	Cornell	Actor	Chameleon	Squirrel
GCN	32.13 \pm 15.90	22.08 \pm 13.43	22.45 \pm 1.09	39.89 \pm 2.49	29.66 \pm 3.52
GAT	34.27 \pm 20.06	24.39 \pm 12.58	24.31 \pm 2.17	37.86 \pm 3.24	24.56 \pm 2.79
APPNP	34.67 \pm 14.53	34.98 \pm 19.61	28.41 \pm 3.12	29.38 \pm 1.60	21.11 \pm 1.53
ChebNet	32.13 \pm 13.73	27.57 \pm 9.71	22.00 \pm 3.58	36.41 \pm 2.07	26.43 \pm 1.33
JKNet	30.75 \pm 19.43	25.20 \pm 21.84	21.02 \pm 2.44	32.66 \pm 4.74	24.20 \pm 3.01
GPR-GNN	33.56 \pm 13.49	38.84 \pm 21.45	27.70 \pm 1.80	33.23 \pm 5.80	23.43 \pm 2.30
BernNet	40.69 \pm 19.26	39.32 \pm 13.95	28.85\pm1.48	34.73 \pm 2.45	22.38 \pm 1.43
JacobiConv	41.23 \pm 6.32	39.23 \pm 6.32	26.37 \pm 1.71	41.12 \pm 2.31	32.23\pm1.06
ARNOLDI-GCN	63.20 \pm 2.67	51.24 \pm 1.56	26.63 \pm 1.12	40.25 \pm 2.15	24.12 \pm 1.94
G-ARNOLDI-GCN	66.20\pm3.21	55.87\pm0.73	27.73 \pm 1.81	43.35\pm3.35	26.64 \pm 2.01
Semi-supervised on Large Heterophilic datasets					
Method	roman-empire	amazon-ratings	minesweeper	tolokers	questions
GCN	29.02 \pm 0.70	28.97 \pm 1.42	72.60 \pm 1.08	73.11 \pm 3.61	59.86 \pm 2.42
GAT	37.26 \pm 0.75	29.97 \pm 2.33	73.51 \pm 0.99	71.11 \pm 0.67	64.39 \pm 0.74
APPNP	35.30 \pm 0.95	29.88 \pm 0.53	67.75 \pm 1.12	68.99 \pm 0.48	46.80 \pm 0.61
ChebNet	35.97 \pm 1.33	27.76 \pm 1.21	73.18 \pm 0.32	70.53 \pm 1.25	64.51 \pm 0.34
JKNet	28.81 \pm 0.33	30.50 \pm 1.25	72.48 \pm 1.05	73.42\pm1.44	56.55 \pm 1.25
GPR-GNN	36.13 \pm 1.24	30.03 \pm 1.81	76.87 \pm 0.80	68.64 \pm 0.87	54.13 \pm 3.84
BernNet	39.63 \pm 0.67	29.32 \pm 1.05	75.49 \pm 1.10	69.08 \pm 0.93	56.27 \pm 0.83
JacobiConv	41.02 \pm 0.34	30.24 \pm 1.20	74.13 \pm 1.52	65.15 \pm 0.73	56.21 \pm 1.03
ARNOLDI-GCN	53.04 \pm 0.34	43.71 \pm 0.61	69.73 \pm 1.03	70.25 \pm 2.15	56.12 \pm 0.94
G-ARNOLDI-GCN	69.63\pm0.37	48.68\pm0.70	88.52\pm0.73	73.01 \pm 1.41	66.18\pm1.22

GPR-GNN [5], BernNet [1], and JacobiConv [12]) in both semi and fully supervised settings. Finally, we assess the relationship between the numerical accuracy of the polynomial approximation and node classification performance of the resulting Spectral GCNs.

Table 3: Accuracy of node classification for ARNOLDI-GCN, G-ARNOLDI-GCN, and state-of-the-art algorithms. Mean and standard deviation of five runs are reported for all experiments.

Fully-supervised on Homophilic datasets					
Method	Cora	Citeseer	Pubmed	Photos	Computers
GCN	86.45 \pm 1.02	79.67 \pm 1.06	87.12 \pm 0.49	86.74 \pm 0.62	83.21 \pm 0.29
GAT	87.11 \pm 1.52	79.63 \pm 1.37	86.69 \pm 0.70	90.72 \pm 2.33	83.15 \pm 0.39
APPNP	87.78 \pm 1.78	79.50 \pm 1.33	88.31 \pm 0.39	85.15 \pm 0.92	82.51 \pm 0.31
ChebNet	86.20 \pm 0.72	78.11 \pm 1.19	86.03 \pm 0.56	92.48 \pm 0.39	85.34 \pm 0.60
JKNet	85.87 \pm 1.56	76.56 \pm 1.37	85.52 \pm 0.46	90.04 \pm 1.38	83.18 \pm 1.21
GPR-GNN	87.71 \pm 0.85	79.57 \pm 1.47	87.27 \pm 0.48	93.28 \pm 0.62	84.68 \pm 0.60
BernNet	87.37 \pm 1.09	79.41 \pm 1.73	88.84 \pm 0.63	92.38 \pm 0.58	86.24 \pm 0.31
JacobiConv	88.23 \pm 0.42	79.32 \pm 0.92	88.02 \pm 0.56	93.10 \pm 0.61	86.43 \pm 0.52
ARNOLDI-GCN	89.23 \pm 0.33	79.21 \pm 1.02	87.65 \pm 0.54	92.06 \pm 0.45	86.24 \pm 0.42
G-ARNOLDI-GCN	90.64\pm0.53	81.71\pm0.87	89.25\pm0.43	94.96\pm0.54	87.66\pm0.28
Fully-supervised on Small Heterophilic datasets					
Method	Texas	Cornell	Actor	Chameleon	Squirrel
GCN	75.10 \pm 1.20	66.94 \pm 1.41	33.26 \pm 1.24	60.96 \pm 0.68	45.76 \pm 0.52
GAT	78.64 \pm 0.96	76.27 \pm 1.32	35.90 \pm 0.17	63.40 \pm 0.49	42.98 \pm 0.65
APPNP	81.10 \pm 1.30	91.86 \pm 0.51	38.68 \pm 0.46	52.51 \pm 1.30	35.27 \pm 0.49
ChebNet	86.23 \pm 0.93	85.46 \pm 1.09	37.80 \pm 0.65	59.94 \pm 1.27	40.81 \pm 0.37
JKNet	75.51 \pm 1.43	67.24 \pm 1.65	33.27 \pm 1.04	62.78 \pm 1.34	44.82 \pm 0.54
GPR-GNN	82.22 \pm 0.74	91.45 \pm 1.62	39.48 \pm 0.83	66.91 \pm 1.21	49.80 \pm 0.56
BernNet	91.77 \pm 1.06	92.02 \pm 1.70	41.60 \pm 1.10	67.76 \pm 1.25	50.61 \pm 0.72
JacobiConv	91.23 \pm 1.43	90.17 \pm 1.61	40.23 \pm 1.05	73.12\pm1.16	56.20\pm1.95
ARNOLDI-GCN	92.03 \pm 1.24	91.92 \pm 1.41	39.24 \pm 0.81	58.33 \pm 1.43	41.03 \pm 1.11
G-ARNOLDI-GCN	95.08\pm1.33	94.95\pm1.13	42.33\pm0.46	68.92 \pm 1.29	49.24 \pm 1.21
Fully-supervised on Large Heterophilic datasets					
Method	roman-empire	amazon-ratings	minesweeper	tolokers	questions
GCN	47.51 \pm 0.25	43.65 \pm 0.26	72.44 \pm 1.04	74.96 \pm 0.92	67.34 \pm 0.56
GAT	44.95 \pm 1.34	42.33 \pm 0.31	73.51 \pm 0.99	72.18 \pm 0.90	69.23 \pm 0.90
APPNP	49.34 \pm 0.17	42.59 \pm 0.05	68.74 \pm 1.03	69.69 \pm 0.57	59.00 \pm 0.77
ChebNet	66.23 \pm 1.04	44.80 \pm 0.43	80.12 \pm 0.33	80.26\pm1.56	70.67 \pm 0.19
JKNet	43.55 \pm 0.20	43.95 \pm 0.48	76.32 \pm 0.24	74.51 \pm 0.67	59.55 \pm 1.02
GPR-GNN	63.64 \pm 0.31	45.14 \pm 0.50	81.89 \pm 0.79	70.84 \pm 0.99	68.02 \pm 1.04
BernNet	63.83 \pm 0.38	44.20 \pm 0.75	77.90 \pm 1.13	71.23 \pm 0.76	66.19 \pm 0.95
JacobiConv	62.24 \pm 0.34	41.02 \pm 0.61	79.95 \pm 1.65	69.17 \pm 0.31	67.09 \pm 0.45
ARNOLDI-GCN	61.77 \pm 0.35	43.85 \pm 0.30	77.27 \pm 1.09	75.90 \pm 0.97	65.56 \pm 0.86
G-ARNOLDI-GCN	73.34\pm0.34	49.58\pm0.07	92.14\pm0.46	77.34 \pm 0.49	74.16\pm0.85

4.1 Datasets and Experimental Setup

We use fifteen real-world network datasets for our experiments. Five of these datasets are homophilic networks, which include three citation graph datasets, Cora, CiteSeer and PubMed [19], and two Amazon co-purchase graphs, Computers and Photo [20]. The remaining five networks are heterophilic, which include Wikipedia graphs Chameleon and Squirrel [21], the Actor co-occurrence graph, and the webpage graph Texas and Cornell from WebKB3 [22] as well as recently curated large heterophilic datasets, Roman-Empire, Amazon-Rating, Widesweeper, Tolokers, and

Questions [23]. The statistics and homophily measures of these fifteen datasets are given in Table 1. *Adjusted Homophily* measures the association between being connected by an edge and having the same class label, corrected for class imbalance. *Label Informativeness* measures the extent to which a neighbor’s label provides information on the label of the node [24].

In the following subsections, we report the results of three sets of experiments. For all settings, we perform cross-validation by hiding a fraction of labels (this fraction depends on the specific experiment as specified below) and assess performance using Accuracy (fraction of correctly classified nodes). *Widesweeper*, *Tolokers*, and *Questions* datasets have binary classes, thus we report area under ROC curve (AUROC) for these datasets. We repeat all experiments five times and report the mean and standard deviation of accuracy. Hyper-parameter settings are presented in the Appendix.

4.2 Comparison of Node Classification Performance of ARNOLDI-GCN and G-ARNOLDI-GCN to State-of-the-Art

We compare our proposed algorithms, ARNOLDI-GCN and G-ARNOLDI-GCN, against state-of-the-art Spectral GCNs on five homophilic and ten heterophilic real-world datasets for both semi and fully supervised node classification tasks. To assess semi-supervised node classification performance, we use cross-validation by randomly splitting the data into training/validation/test samples as (%2.5 / %2.5 / %95). To assess fully supervised node classification performance, we use cross-validation by randomly splitting the data into training/validation/test samples as (%60 / %20 / %20). The results of these experiments are presented in Table 2 with the best-performing method(s) highlighted in bold. Our methods, particularly G-ARNOLDI-GCN, demonstrate significant improvements in the performance of Spectral-GCNs, as seen in these tables. We observe that the performance improvement is particularly pronounced on heterophilic networks, highlighting the importance of using explicit filters for complex machine learning tasks. To provide insights into which filter functions deliver best performance on which classification tasks, we provide the details of the best-performing filter in Table 4 in the Appendix.

4.3 Value Added by Numerical Accuracy

The results shown in the previous set of experiments demonstrate that the use of explicit filter functions and their polynomial approximation significantly enhances node classification performance of Spectral GCNs. To gain insights into the value added by the numerical accuracy in this process, we first investigate the effect of the linear system solver and the sampling technique used for the polynomial approximation on the accuracy of the approximation. Figure 2 compares Arnoldi orthonormalization to direct solution of the Vandermonde system in approximating *Random Walk* (g_1) and *Low Pass* (g_4) filters using *Equispaced* and *Chebyshev* sampling. As seen in the figure, Arnoldi significantly reduces errors, particularly for complex filters. Furthermore, *Chebyshev* sampling consistently outperforms *Equispaced* sampling.

4.4 From Approximation Quality to Classification Accuracy

In the previous section, we observe that Arnoldi orthonormalization drastically improves the accuracy of polynomial approximations to filter functions. We now assess how this numerical accuracy translates to the observed superior classification performance of the Spectral GCN. For this purpose, we compare the node classification performance of the Spectral GCNs that utilize polynomials generated by direct solution of the Vandermonde system (VANDERMONDE-GCN) vs. Arnoldi orthonormalization (ARNOLDI-GCN), using three homophilic and three heterophilic datasets. In these experiments, we use *Random Walk* and *Low Pass* filters and polynomial approximations computed using *Equispaced* and *Chebyshev* sampling. As seen in Figure 3, ARNOLDI-GCN consistently and significantly outperforms VANDERMONDE-GCN, demonstrating the importance of numerically stable polynomial approximations in training reliable GCN models. In addition, *Chebyshev* sampling consistently outperforms *Equispace* in semi-supervised node classification. Integration of complex filters, e.g., *Low-Pass*, delivers substantial improvements, especially for heterophilic datasets such as Texas and Cornell.

4.5 Discussion

Based on the comprehensive results presented in this section, we derive the following key insights:

- G-ARNOLDI-GCN outperforms state-of-the-art algorithms on 12 out of 15 datasets in both semi-supervised and supervised node classification tasks.
- On 6 out of 10 heterophilic datasets, G-ARNOLDI-GCN’s performance improvement over other algorithms is highly significant (more than 10 standard deviations).

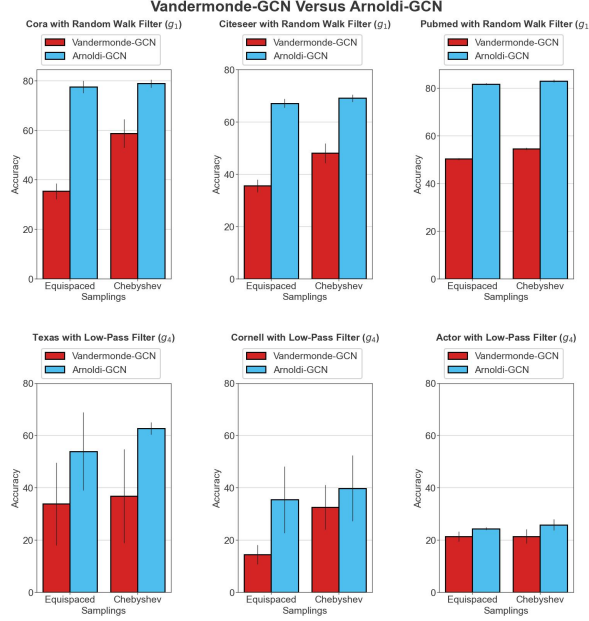


Figure 3: **Semi-supervised learning performance of Vandermonde and Arnoldi-generated approximations.** The plots show the mean and standard deviation of node classification accuracy across five runs on six datasets.

- On homophilic graphs, filters that assign more weight to local neighborhood of nodes deliver best performance. These include simple filters, which can be considered variations of random walks, as well as low-pass filters.
- On heterophilic graphs, self-depressed and neighbor-depressed filters compete with high-pass filters. This is likely because they initialize coefficients to assign less weight to a node’s self and immediate neighbors, allowing supervised learning to converge to desirable local optima.
- On the Questions dataset, band-rejection filters outperform all other filters and competing algorithms by more than 10 standard deviations. This indicates that the graph has specific topological properties that cannot be captured solely by homophily and label informativeness. Additionally, this observation demonstrates that the ability to design and apply complex filters enhances the effectiveness of spectral GCNs on graphs with intricate topological characteristics.

5 Conclusion

The ill-conditioned nature of polynomial approximation to filter functions poses significant challenges for Spectral GCNs in defining suitable signal propagation. Our algorithm, G-ARNOLDI-GCN, overcomes this challenge by enabling numerically stable polynomial approximation of any filter function. G-ARNOLDI-GCN excels in multi-class node classification across diverse datasets, showcasing its customizability with various filters and sampling methods available to choose from. G-ARNOLDI-GCN marks a new direction in graph machine learning, enabling the explicit design and application of diverse filter functions in Spectral GCNs.

6 Appendix

6.1 Proofs of Theorems

In this subsection, we prove the theorems in the body of the paper. In the following proofs we assume $K = r$ and all the proofs are valid when $K \leq r$ since least square fitting is norm-wise backward stable [25].

Proof 1 (Proof of Theorem 1) Let $\mathcal{K}^K = \text{span}\{\mathbf{q}_1, \Omega\mathbf{q}_1, \Omega^2\mathbf{q}_1, \dots, \Omega^K\mathbf{q}_1\}$ be the Krylov space, where $\mathbf{q}_1 = \frac{\mathbf{e}}{\|\mathbf{e}\|}$, and its column vectors be \mathbf{Q} generated by applying QR factorization on \mathbf{V} . Assume that $\Omega^K\mathbf{q}_1$ is dropped from the Krylov space because QR factorization found an invariant subspace at point K . Let \mathbf{v} be an arbitrary orthonormal vector to $\Omega^K\mathbf{q}_1$ as \mathbf{v} , i.e., \mathbf{v} is perpendicular to $\Omega^K\mathbf{q}_1$. Since this \mathbf{v} is perpendicular to $\Omega^K\mathbf{q}_1$ that is dropped from \mathcal{K}^K , we can express this vector as a linear combination of all previous vectors in \mathcal{K}^K . That is, for some $\beta_i \in (0, 1)$, we have $\mathbf{v} = \beta_1\mathbf{q}_1 + \beta_2\Omega\mathbf{q}_1 + \dots + \beta_K\Omega^K\mathbf{q}_1$. Clearly, we can state the last equation as a monic polynomial, i.e., $\mathbf{v} = p_K(\Omega)\mathbf{q}_1$.

Let \mathcal{P}_j be the family of all monic polynomials, then:

$$\|\mathbf{v}\| = \min_{p_j(\omega) \in \mathcal{P}_j} \|p_K(\boldsymbol{\Omega})\mathbf{q}_1\| \quad (26)$$

$$\leq \min_{p_j(\omega) \in \mathcal{P}_j} \max_{-\alpha \leq \omega \leq \alpha} |p_K(\omega)| \|\mathbf{q}_1\| \quad (27)$$

$$= \min_{p_j(\omega) \in \mathcal{P}_j} \max_{-\alpha \leq \omega \leq \alpha} |p_K(\omega)| \quad (28)$$

The last equation suggests that choose such a j ω s from $[-\alpha, \alpha]$, $-\alpha \leq \omega_1, \omega_2, \dots, \omega_j \leq \alpha$, such that following values are as small as possible:

$$\max_{-\alpha \leq \omega \leq \alpha} |p_K(\omega)| = \max_{-\alpha \leq \omega \leq \alpha} |(\omega - \omega_1)(\omega - \omega_2) \cdots (\omega - \omega_j)|$$

From Approximation Theory, we know that Chebyshev polynomial first kind's roots satisfy the above equation because from cosine definition of Chebyshev polynomial, the maximum value of $\mathcal{T}_K(\frac{\omega}{\alpha}) = 1$. To use this property, let us define re-scaled Chebyshev polynomial as:

$$\mathcal{T}_K(\frac{\omega}{\alpha}) = \frac{1}{2^{K-1}} \alpha^K \omega^K + \dots$$

Since ω s are chosen as the roots of the above polynomial:

$$|(\omega - \omega_1)(\omega - \omega_2) \cdots (\omega - \omega_j)| = \frac{1}{2^{K-1}} \alpha^K |\mathcal{T}_K(\frac{\omega}{\alpha})|$$

Then,

$$\min_{p_j(\omega) \in \mathcal{P}_j} \max_{-\alpha \leq \omega \leq \alpha} |(\omega - \omega_1)(\omega - \omega_2) \cdots (\omega - \omega_j)| \leq \frac{1}{2^{K-1}} \alpha^K$$

Thus, $\|\mathbf{v}\| \leq (1/2^{K-1})\alpha^K$. Using the definition of Frobenius norm of a low rank matrix, and using the fact that limit of matrix goes zero after convergence, we have:

$$\|\mathbf{Q}^\dagger\|_2 = \sigma_r \geq 2^{r-1} \left(\frac{1}{\alpha}\right)^r \text{ and } \|\mathbf{Q}\|_2 \geq 1$$

Thus, we have:

$$\kappa(\mathbf{V}) = \|\mathbf{V}\|_2 \|\mathbf{V}^\dagger\|_2 \geq \sigma_r \geq 2^{r-1} \left(\frac{1}{\alpha}\right)^r$$

Now, we need to show that \mathbf{V} is still ill-conditioned when $\omega_k \geq 1$ for $k = 1, \dots, r$ to establish the result for range $(\varepsilon, 2)$ (since we have shown result holds for $(-1, 1)$). To prove this, let $\hat{\mathbf{q}}_1 = \boldsymbol{\Omega}^{r-1} \mathbf{q}_1$ and $\hat{\boldsymbol{\Omega}} = \frac{1}{\hat{\boldsymbol{\Omega}}}$. Now, let $\hat{\mathbf{V}}$ be reversely ordered version of the original \mathbf{V} . Clearly, these matrices have the same condition number, i.e., $\kappa(\mathbf{V}) = \kappa(\hat{\mathbf{V}})$. Furthermore:

$$\hat{\mathbf{V}} = [\hat{\mathbf{q}}_1, \hat{\boldsymbol{\Omega}} \hat{\mathbf{q}}_1, \dots, \hat{\boldsymbol{\Omega}}^{K-1} \hat{\mathbf{q}}_1]$$

Now, all elements of $\hat{\boldsymbol{\Omega}}$ are in interval $[-1, 1]$, thus (from the previous establishment for $(-1, 1)$) we have:

$$\kappa(\hat{\mathbf{V}}) \geq 2^{r-2}$$

Therefore, condition number of \mathbf{V} or $\hat{\mathbf{V}}$, which is computed by QR factorization on \mathbf{V} is exponentially unbounded and thus coefficients generated by this \mathbf{V} are inaccurate in Equation 21.

Proof 2 (Proof of Theorem 2) By using the definition of Vandermonde matrix, $\mathbf{V} = [\mathbf{e}, \boldsymbol{\Omega} \mathbf{e}, \dots, \boldsymbol{\Omega}^{K-1} \mathbf{e}]$, $\mathbf{V}^{(*)} = [\mathbf{q}_1, \boldsymbol{\Omega} \mathbf{q}_1, \dots, \boldsymbol{\Omega}^{K-1} \mathbf{q}_1]$, where $\mathbf{q}_1 = \frac{\mathbf{e}}{\|\mathbf{e}\|}$. Let $\mathbf{e}_i \in \mathbb{R}^{K-1}$ be i -th unit vector, i.e., only i -th entry is 1 and rest zero. We prove the theorem by induction. Let $i \leq K-1$, then for $i = 1$, since $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_K]$ we have,

$$\mathbf{V}^{(*)} \mathbf{e}_1 = \mathbf{Q} \mathbf{R} \mathbf{e}_1 = \mathbf{Q} \mathbf{e}_1 = \mathbf{q}_1 \quad (29)$$

Now, assume $i = 2$, then we have,

$$\mathbf{V}^{(*)} \mathbf{e}_2 = \boldsymbol{\Omega} \mathbf{q}_1 = (\boldsymbol{\Omega} \mathbf{Q} - \tilde{\mathbf{O}}) \mathbf{e}_1 = \mathbf{Q} \mathbf{T} \mathbf{e}_1 = \mathbf{Q} \mathbf{R} \mathbf{e}_2 \quad (30)$$

Assuming that the claim holds for $i = K-1$, we have:

$$\begin{aligned} \mathbf{V}^{(*)} \mathbf{e}_{K-1} &= \mathbf{\Omega}^{K-1} \mathbf{q}_1 = (\mathbf{\Omega}^{K-1} \mathbf{Q} - \tilde{\mathbf{O}}) \mathbf{e}_1 \\ &= \mathbf{Q} \mathbf{T}^{K-1} \mathbf{e}_1 = \mathbf{Q} \mathbf{Re}_{K-1} \end{aligned} \quad (31)$$

Now, we need to show Equation 31 holds for $i = K$. First notice that from the assumption, we have $\mathbf{\Omega}^K \mathbf{q}_1 = \mathbf{\Omega}(\mathbf{\Omega}^{K-1} \mathbf{q}_1) = \mathbf{\Omega}(\mathbf{Q} \mathbf{T}^{K-1} \mathbf{e}_1) = (\mathbf{\Omega} \mathbf{Q}) \mathbf{T}^{K-1} \mathbf{e}_1$, so we can write:

$$\begin{aligned} \mathbf{\Omega}^K \mathbf{q}_1 &= (\mathbf{\Omega} \mathbf{Q}) \mathbf{T}^{K-1} \mathbf{e}_1 = (\mathbf{Q} \mathbf{T} - \tilde{\mathbf{O}}) \mathbf{T}^{K-1} \mathbf{e}_1 \\ &= \mathbf{Q} \mathbf{T}^K \mathbf{e}_1 - \tilde{\mathbf{O}} \mathbf{T}^{K-1} \mathbf{e}_1 = \mathbf{Q} \mathbf{T}^K \mathbf{e}_1 = \mathbf{Q} \mathbf{Re}_{K+1} \end{aligned} \quad (32)$$

Finally, we have $\mathbf{V}^{(*)} \mathbf{e}_{K+1} = \mathbf{Q} \mathbf{Re}_{K+1}$. This, prove that $\|e\| \times \mathbf{QR}$ is QR of Vardermonde matrix. This completes the proof.

Proof 3 (Proof of Theorem 3) Assume $K = r$, then \mathbf{Q} is full rank and $\text{rank}(\mathbf{Q}) = r$ since ω s are unique. When $K \leq r$ the proof still holds since least square fitting is norm-wise backward stable [25]. Ignoring the constant term, r , from the construction, \mathbf{Q} is orthonormal. Thus, we have $\mathbf{Q}^T = \mathbf{Q}^{-1}$ since $\mathbf{Q} \mathbf{Q}^T = \mathbf{I}$. Therefore, in theory, $\kappa(\mathbf{Q}) = 1$. However, in practice, due to machine round-off error, this number might change, especially, when Ritz values, i.e., eigenvalues of \mathbf{T} , converge to eigenvalues of $\mathbf{\Omega}$. To illustrate this point, let ε be machine epsilon and let us represent floating point by fl , i.e., stored version of \mathbf{Q} in the machine is $fl(\mathbf{Q})$. We want to measure the difference between $\mathbf{Q} \mathbf{Q}^{-1}$ and \mathbf{I} . That is, we want to compute:

$$fl(\mathbf{Q} \mathbf{Q}^{-1} - \mathbf{I})$$

Using Golub and Van [26] (2.4.15) and (2.4.18), we have:

$$fl(\mathbf{Q} \mathbf{Q}^{-1} - \mathbf{I}) = (\mathbf{Q} \mathbf{Q}^{-1} - \mathbf{I}) + \mathbf{E}_1 \quad \mathbf{E}_1 \leq \varepsilon \|\mathbf{Q} \mathbf{Q}^{-1} - \mathbf{I}\|$$

and

$$fl(\mathbf{Q} \mathbf{Q}^{-1}) = \mathbf{Q} \mathbf{Q}^{-1} + \mathbf{E}_1 \quad \mathbf{E}_2 \leq \varepsilon \times r \|\mathbf{Q} \mathbf{Q}^{-1}\| + O(\varepsilon^2)$$

By using the above floating point analysis for matrix multiplication and addition, $\kappa(\mathbf{Q})$ perturbs from 1 around 0.01 due to $O(\varepsilon^2)$ error term. Thus, we have

$$\kappa(\mathbf{Q}) \approx 1.01$$

6.2 Hyper-parameters

In the body of the paper, we perform three sets of experiments in which the following hyperparameters are used:

Experiment 1. For these experiments, we use two hyper-parameters that are – the number of samples, r , used to approximate the filter functions, and dimension of \mathbf{Q} , K . We set these two parameters to $r, K = 40$.

Experiment 2. In this experiment, we use ARNOLDI-GCN in Equation 24 in which coefficients are computed by Vandermonde 20 and Arnoldi in Algorithm 1. For these experiments, we limit ourselves to semi-supervised node classification task and consider the random split into training/validation/test samples, which is a sparse splitting (%2.5 / %2.5 / %95). For fair comparison, we use the same filter functions for both Vandermonde and Arnoldi. Namely, for homophilic datasets, we use *RW filter function*, i.e., $g_{RW}(\omega) = \frac{1}{1-\omega}$, where $\omega \in [-0.9, 0.9]$, and for heterophilic datasets, we use *low pass filter function*, i.e., $g_{LP}(\omega) = \exp\{-10\omega\}$, where $\omega \in [\varepsilon, 2]$, ε is 10^{-5} . Furthermore, we choose number of samples and dimension of \mathbf{Q} as $r, K = 10$ and use a 2-layer (MLP) with 64 hidden units for the NN component. For other hyperparameters, we fix $\{\text{learning rate} / \text{weight decay} / \text{dropout}\}$ to $\{0.002 / 0.0005 / 0.5\}$, respectively.

Experiment 3. In these experiments, we use eight representative filter functions in Equations 7- 14 and approximate them by using four representative polynomial samples (Equispaced, Chebyshev, Legendre, and Jacobi roots) in Equations 16- 19. We use bound $[-0.9, 0.9]$ for simple filter functions ($g_0 - g_3$) and $[10^{-5}, 2]$ for complex filter functions ($g_4 - g_7$). For semi-supervised node classification tasks we again randomly split the data into training/validation/test samples as (%2.5 / %2.5 / %95); and for full-supervised node classification tasks, we randomly split the data into training/validation/test samples as (%60 / %20 / %20). Again, we choose number of samples and dimension of \mathbf{Q} as $r, K = 10$ and a 2-layer MLP with 64 hidden units are used, with weight decay = 0.0005. Learning rate is tuned within 0.001, 0.002, 0.01, 0.05 and dropout within 0.1 to 0.9. The best combinations are in the tables.

For fair comparison, we use the best hyperparameters from state-of-the-art Spectral GCN algorithms (GCN, GAT, APPNP, ChebNet, JKNet, GPR-GNN, BernNet, JacobiConv). For random walk length in APPNP, ChebNet, GPR-GNN, BernNet, and JacobiConv we use $K = 10$. We generate polynomials using Python's *polyld*. The hyperparameter combinations that are used to generate the results in Table 2 are shown in Table 4.

Table 4: Hyperparameters used for ARNOLDI-GCN and G-ARNOLDI-GCN for node classification.

Datasets	Samplings	Filter	Propagation layer learning rate	Propagation layer dropout rate
<i>Semi-Supervised classification</i>				
Cora	Jacobi	g_0 (Scaled RW)	0.001	0.7
Citeseer	Jacobi	g_2 (Self-Depressed RW)	0.002	0.6
Pubmed	Chebyshev	g_3 (Neighbor-Depressed RW)	0.001	0.1
Photos	Chebyshev	g_0 (Scaled RW)	0.002	0.1
Computers	Equispace	g_2 (Self-Depressed RW)	0.002	0.1
Texas	Legendre	g_4 (High-Pass)	0.01	0.2
Cornell	Jacobi	g_7 (Band-Rejection)	0.05	0.9
Actor	Jacobi	g_7 (High-Pass)	0.001	0.5
Chameleon	Legendre	g_1 (RW)	0.05	0.8
Squirrel	Equispace	g_0 (Self-Depressed RW)	0.01	0.2
roman-empire	Chebyshev	g_5 (High-Pass)	0.01	0.2
amazon-ratings	Legendre	g_5 (High-Pass)	0.01	0.3
minesweeper	Jacobi	g_3 (Neighbor Depressed-RW)	0.01	0.1
tolokers	Jacobi	g_5 (High-Pass)	0.002	0.5
questions	Chebyshev	g_7 (Band-Rejection)	0.002	0.1
<i>Fully-Supervised classification</i>				
Datasets	Samplings	Filter	Propagation layer learning rate	Propagation layer dropout rate
Cora	Jacobi	g_3 (Neighbor-Depressed RW)	0.05	0.4
Citeseer	Chebyshev	g_3 (Neighbor-Depressed RW)	0.05	0.4
Pubmed	Equispace	g_5 (Low-Pass)	0.001	0.1
Photos	Equispace	g_1 (RW)	0.002	0.5
Computers	Chebyshev	g_5 (Low-Pass)	0.002	0.5
Texas	Legendre	g_2 (Self-Depressed RW)	0.05	0.4
Cornell	Chebyshev	g_2 (Self-Depressed RW)	0.05	0.4
Actor	Equispace	g_2 (Self-Depressed RW)	0.05	0.4
Chameleon	Legendre	g_0 (Scaled RW)	0.05	0.5
Squirrel	Jacobi	g_2 (Self-Depressed RW)	0.05	0.3
roman-empire	Chebyshev	g_5 (High-Pass)	0.001	0.2
amazon-ratings	Chebyshev	g_5 (High-Pass)	0.002	0.1
minesweeper	Jacobi	g_3 (Neighbor Depressed-RW)	0.01	0.1
tolokers	Equispace	g_2 Neighbor Depressed-RW)	0.002	0.5
questions	Chebyshev	g_7 (Band-Rejection)	0.002	0.1

References

- [1] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34:14239–14251, 2021.
- [2] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018*, pages 593–607, 2018.
- [3] Prakash Chandra Rathi, R Frederick Ludlow, and Marcel L Verdonk. Practical high-quality electrostatic potential surfaces for drug discovery using a graph-convolutional deep neural network. *Journal of medicinal chemistry*, 63(16):8778–8790, 2019.
- [4] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*, 13(1):1–23, 2021.
- [5] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv:2006.07988*, 2020.

- [6] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [7] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.
- [8] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International conference on learning representations*, 2020.
- [9] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *Proceedings of the Web Conference 2021*, pages 1215–1226, 2021.
- [10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.
- [11] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE TSP*, 67(1):97–109, 2018.
- [12] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pages 23341–23362. PMLR, 2022.
- [13] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.
- [14] Masanori Matsuhara and KO Hill. Optical-waveguide band-rejection filters: Design. *Applied Optics*, 13(12):2886–2888, 1974.
- [15] Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.
- [16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.
- [17] SHI Xingjian, Zhouong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NeuroIPS*, pages 802–810, 2015.
- [18] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462, 2018.
- [19] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48, 2016.
- [20] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv:1811.05868*, 2018.
- [21] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2), 2021.
- [22] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2019.
- [23] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: Are we really making progress? In *ICLR*, 2022.
- [24] Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. Characterizing graph datasets for node classification: Homophily-heterophily dichotomy and beyond. *NeuroIPS*, 36, 2024.
- [25] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.
- [26] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.