Polynomial Selection in Spectral Graph Neural Networks: An Error-Sum of Function Slices Approach

Anonymous Author(s)

ABSTRACT

Spectral graph neural networks are proposed to harness spectral information inherent in graph-structured data through the application of polynomial-defined graph filters, recently achieving notable success in graph-based web applications. Existing studies reveal that various polynomial choices greatly impact spectral GNN performance, underscoring the importance of polynomial selection. However, this selection process remains a critical and unresolved challenge. Although prior work suggests a connection between the approximation capabilities of polynomials and the efficacy of spectral GNNs, there is a lack of theoretical insights into this relationship, rendering polynomial selection a largely heuristic process.

To address the issue, this paper examines polynomial selection from an error-sum of function slices perspective. Inspired by the conventional signal decomposition, we represent graph filters as a sum of disjoint function slices. Building on this, we then bridge the polynomial capability and spectral GNN efficacy by proving that the construction error of graph convolution layer is bounded by the sum of polynomial approximation errors on function slices. This result leads us to develop an advanced filter based on trigonometric polynomials, a widely adopted option for approximating narrow signal slices. The proposed filter remains provable parameter efficiency, with a novel Taylor-based parameter decomposition that achieves streamlined, effective implementation. With this foundation, we propose TFGNN, a scalable spectral GNN operating in a decoupled paradigm. We validate the efficacy of TFGNN via benchmark node classification tasks, along with an example graph anomaly detection application to show its practical utility.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Spectral graph neural networks, Polynomial graph filters, Polynomial approximation, Node classification

ACM Reference Format:

Anonymous Author(s). 2018. Polynomial Selection in Spectral Graph Neural Networks: An Error-Sum of Function Slices Approach. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graph neural networks (GNNs) [73, 84] have emerged as powerful tools to capture structural information from graph data, facilitating advanced performance across numerous web applications, such as web search [5, 79], recommender system [27, 74], social network analysis [9], anomaly detection [14, 18, 46], etc. Among GNN varieties, spectral GNNs stand out for their ability to exploit the spectral properties of graph data using polynomial-defined graph filters, recently achieving notable success in graph-related tasks [73].

Numerous existing studies have empirically revealed that various polynomial choices greatly impact spectral GNN performance [24–26, 31, 38, 72], underscoring the importance of polynomial selection. However, despite the various works that incorporate different polynomials, their primary focus has been on other factors, such as convergence rate [24, 72], rather than explicitly targeting the enhancement of spectral GNN efficacy. As far as we are aware, there is no existing work that directly associates spectral GNN efficacy with polynomial capability, which renders polynomial selection a crucial yet unresolved challenge, often approached heuristically.

To tackle this issue, we investigate polynomial selection through a novel lens of error-sum of function slices in this paper. Drawing inspiration from signal decomposition techniques [21], we uniformly represent graph filters as a sum of disjoint function slices. We present the first proof establishing that the construction error of graph convolution layers is bounded by the sum of polynomial approximation errors on these function slices. This explicitly links the capability of polynomials to the effectiveness of spectral GNNs, supported by intuitive numerical validations that affirm the practicality of our theoretical framework. This finding emphasizes that enhanced spectral GNN efficacy can be attained by utilizing graph filters created with "narrow slice-preferred polynomials". Consequently, we introduce an innovative filter based on trigonometric polynomials [86], a standard approach for approximating narrow signal slices in the signal processing domain. Our proposed filter showcases proven parameter efficiency, leveraging a novel Taylor-based parameter decomposition that facilitates streamlined and effective implementation. Building upon this foundation, we introduce TFGNN, a scalable spectral GNN operating in a widely adopted decoupled GNN architecture [10, 19, 25, 38, 81]. Empirically, we validate TFGNN's capacity via benchmark node classification tasks and highlight its real-world efficacy with an example graph anomaly detection application. Our contributions are summarized below:

- We provide the inaugural proof that connects the efficacy of spectral GNN to their polynomial capabilities, framed through the lens of approximation error on function slices. Our numerical experiments reinforce the practical utility of this connection. This finding offers an informed strategy to refine polynomial selection, leading to enhanced spectral GNNs.
- We introduce an advanced graph filter based on trigonometric polynomials, showcasing provable parameter efficiency. Our

novel approach incorporates a Taylor-based parameter decomposition to achieve a streamlined implementation. Based on this filter, we further develop TFGNN, a scalable spectral GNN characterized by its decoupled architecture.

- We validate TFGNN's effectiveness with extensive experiments in benchmark node classification and an illustrative application in graph anomaly detection. The results reveal that TFGNN not only exceeds previous methods in standard tasks but also yields results comparable to specialized models in real-world settings, demonstrating its significant practical value.

2 BACKGROUNDS AND PRELIMINARIES

Graph notations. Let $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X})$ be an undirected and unweighted graph with adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n\times n}$ and node feature $\boldsymbol{X} \in \mathbb{R}^{n\times m}$. In addition, $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is the *normalized graph Laplacian* [11], with $\boldsymbol{I}$, $\boldsymbol{D}$ being the identity matrix and the degree matrix, respectively. The eigen-decomposition of $\boldsymbol{L}$ is given by $\boldsymbol{L} = \boldsymbol{U}diag(\boldsymbol{\lambda})\boldsymbol{U}^T$, where $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ denotes the eigenvectors, and $\boldsymbol{\lambda} \in [0,2]^n$ represents the corresponding eigenvalues.

Graph filters. The concept of graph filters originates in the field of Graph Signal Processing (GSP) [55, 61, 64], a field dedicated to developing specialized tools for processing signals generated on graphs, grounded in spectral graph theory [11]. A graph filter is specifically a point-wise mapping $f : [0,2] \mapsto \mathbb{R}$ applied to graph Laplacian's eigenvalues, $\boldsymbol{\lambda}$, facilitating the processing of the graph signal $\boldsymbol{x} \in \mathbb{R}^n$ through a filtering operation as shown below [62]:

$$\boldsymbol{z} \triangleq \boldsymbol{U}diag(f(\boldsymbol{\lambda}))\boldsymbol{U}^T\boldsymbol{x}\,, \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^n$ represents the filtered output. This formulation is often identified as the *graph convolution* [61] operation. Due to the intensive computation cost associated with eigendecomposition, the mapping f is typically implemented via polynomial approximations in practice, resulting in the derivation of Eq. 1 as below:

$$\boldsymbol{z} = \boldsymbol{U}diag\left(\sum_{d=0}^{D}\theta_d\mathbf{T}_d(\boldsymbol{\lambda})\right)\boldsymbol{U}^T\boldsymbol{x} = \sum_{d=0}^{D}\theta_d\mathbf{T}_d(\boldsymbol{L})\boldsymbol{x}\,. \tag{2}$$

$\mathbf{T}_d$ denotes the d-th term of a polynomial, with coefficient θ_d.

Spectral-based GNNs. Spectral-based GNNs emerge from the integration of graph filters with graph-structured data. By treating each column of the node feature matrix $\boldsymbol{X}$ as an individual graph signal, a L-layer spectral GNN is architected as multi-layer neural network that processes the hidden feature through filtering operations, as formulated below [3]:

$$\boldsymbol{H}^{(l+1)} = \sigma^{(l)}\left[\sum_{d=0}^{D}\theta_{dl}\mathbf{T}_d(\boldsymbol{L})\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right], \quad \boldsymbol{H}^{(0)} \triangleq \boldsymbol{X}\,. \tag{3}$$

Here, $\boldsymbol{H}^{(l)}$ and $\boldsymbol{W}^{(l)}$ correspond to the hidden layer representation and weight matrix at the l-th layer, respectively, with $\sigma^{(l)}$ representing a non-linear function commonly applied in neural networks. Each l-th layer is termed a *graph convolution layer*, representing a critical building block in spectral GNNs and the subsequent developments in the field [13, 23, 24, 26, 30, 31, 38, 39, 72, 81].

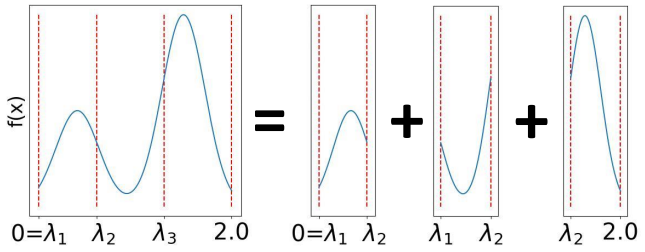


Figure 1: Example of function slicing. $f(x)$ is dissected into three components, determined by its eigenvalues.

3 CONNECTING POLYNOMIAL CAPABILITY WITH SPECTRAL GNN EFFICACY

This section seeks to connect polynomial capability with the efficacy of spectral GNN. We examine the relationship between polynomial approximation errors and feature construction errors in graph convolution layer, providing theoretical analysis alongside intuitive numerical evaluations. This exploration yields vital insights that contribute to the progression of spectral GNNs in a polynomial context. We begin by defining several essential concepts.

Definition 3.1. (*Function slices*). Let $f : [0,2] \mapsto \mathbb{R}$ be a continuous and differentiable filter mapping. Denote the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ of $\boldsymbol{L}$, satisfying $0 = \lambda_1 \le \lambda_2 \le ... \le \lambda_n \le 2$. The function slices of $f(x)$ are given by a set of disjoint functions f_s, $s = 1, 2, ..., n$, satisfying the following conditions:

$$f_s(x) = \begin{cases} f(x) & x \in [\lambda_{s-1}, \lambda_s]\,, \\ 0 & Otherwise\,. \end{cases} \tag{4}$$

Therefore, for any arbitrary function f, we can represent it by summing its slices, as illustrated below:

$$f(x) = \sum_{s=1}^{n} f_s(x)\,. \tag{5}$$

Figure 3.1 provides an intuitive example of function slicing. This concept parallels the signal decomposition techniques found in the conventional signal processing field [21].

Definition 3.2. (*Polynomial's approximation error*). Let $\mathbf{T}_{0:D}(x; f)$ represent a polynomial function of degree D that achieves the least squares error (LSE) [58, 65] in approximating a specified function $f(x)$. Accordingly, we can define both continuous and discrete forms of the approximation error relative to the target filter function $f(x)$ using $\mathbf{T}_{0:D}$ as follows:

$$\text{(Continuous)} \qquad \epsilon \triangleq \int_0^2 ||\mathbf{T}_{0:D}(x; f) - f(x)||^2\, dx\,, \tag{6}$$

$$\text{(Discrete)} \qquad \epsilon \triangleq |||\mathbf{T}_{0:D}(\boldsymbol{\lambda}; f) - f(\boldsymbol{\lambda})|||_F\,, \tag{7}$$

where $\|\cdot\|_F$ denotes Frobenius norm [67].

Our analysis centers on the continuous form, with derived insights adapted to the discrete form for application in spectral GNNs.

Definition 3.3. (*Construction error of graph convolution layer*). Let $\boldsymbol{Y}$ denote the target output of a graph convolution layer, expressed as $\boldsymbol{Y} = \boldsymbol{U}diag(f(\boldsymbol{\lambda}))\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{W}$, where f serves as the "optimal" filter function for constructing $\boldsymbol{Y}$. The construction error of the graph

convolution layer on Y through a D-degree polynomial filter function $\mathbf{T}_{0:D}$ is defined as:

$$\xi \triangleq \|Udiag(\mathbf{T}_{0:D}(\boldsymbol{\lambda}; f) - f(\boldsymbol{\lambda}))\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{W}\|_F \,. \tag{8}$$

Note that the error ξ, analogous to ϵ in Definition 3.2, is measured as the difference between the target function f and the polynomial $\mathbf{T}_{0:D}$ that achieves the least squares error (LSE) approximation. The graph convolution layer introduced in Definition 3.3 aligns with a *one-layer linear GNN*, a configuration similarly explored in previous studies [72, 76]. These prior works have examined the effectiveness of a one-layer linear GNN in constructing node labels to evaluate the overall performance of GNNs, which inspired us to examine the construction error within the graph convolution layer.

3.1 Theoretical insights

Polynomial capability is quantified by the function approximation error [58, 65], whereas spectral GNN efficacy is typically reflected by prediction error in downstream tasks [10, 30, 35, 38, 68, 72, 81]. Consequently, a natural step toward linking polynomial capabilities with spectral GNN efficacy is to establish a bridge between the polynomial approximation error, ϵ, as defined in Definition 3.2, and the graph convolution layer's construction error, ξ.

In particular, as described in Definition 3.1, for an "optimal" filter function $f(x)$, the approximation error of a D-degree polynomial $\mathbf{T}_{0:D}(x; f)$ satisfies the conditions outlined in the following Lemma:

Lemma 3.4. *Let $f(x)$ be a function composed of function slices $f_s(x)$, $s = 1, 2, ..., n$. Let $\mathbf{T}_{0:D}(x; f)$ be a D-degree polynomial that provides LSE approximation of $f(x)$ with error ϵ. Specially, define ϵ_s, $s = 1, 2, ..., n$, as the polynomial approximation error of each slice $f_s(x)$ when approximated by the D-degree polynomial $\mathbf{T}_{0:D}(x; f_s)$. An inequality that bounds ϵ in terms of ϵ_s are formulated below:*

$$\sum_{s=1}^{n} \epsilon_s \leq \epsilon \leq (\sum_{s=1}^{n} \sqrt{\epsilon_s})^2 \,. \tag{9}$$

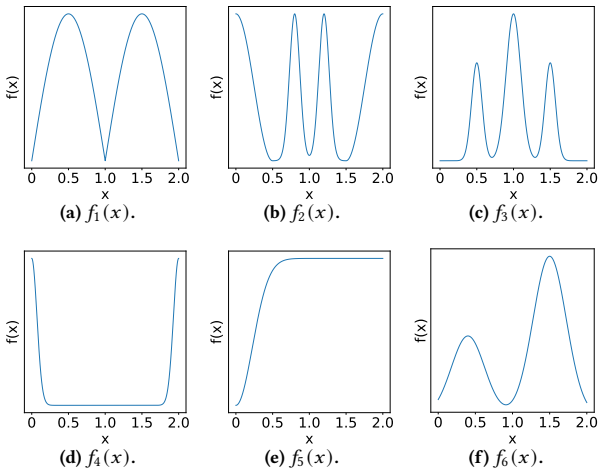
Proof can be found in Appendix B. Lemma 3.4 establishes both upper and lower bounds for the approximation error of a polynomial in relation to an arbitrary function f, based on the errors associated with its slices f_s. This result suggests that the capacity of the polynomial can be equivalently evaluated through the approximation error of its slices.

Drawing from the insights of bounded error above, we can now propose an inequality that bounds the construction error of the graph convolution layer, utilizing the polynomial approximation error as outlined in the theorem below:

Theorem 3.5. *Let $\delta_{\boldsymbol{X}}$ and $\delta_{\boldsymbol{W}}$ denote the minimum singular values of $\boldsymbol{X}$ and $\boldsymbol{W}$, respectively. Consider a regularization on the weight matrix $\boldsymbol{W}$, namely L_2 regularization, expressed as $\|\boldsymbol{W}\|_F \leq r$. The construction error ξ, satisfies the following inequality:*

$$\delta_{\boldsymbol{X}}\delta_{\boldsymbol{W}} \sum_{s=1}^{n} \epsilon_s \leq \xi \leq r\|\boldsymbol{X}\|_F (\sum_{s=1}^{n} \sqrt{\epsilon_s})^2 \,. \tag{10}$$

Proof can be found in Appendix C. Theorem 3.5 establishes a direct connection between the polynomial approximation error and the construction error of the graph convolution layer through the approximation error of function slices, ϵ_s. This insight is novel and, to our knowledge, has not been documented before.



(a) $f_1(x)$. (b) $f_2(x)$. (c) $f_3(x)$.

(d) $f_4(x)$. (e) $f_5(x)$. (f) $f_6(x)$.

Figure 2: The functions served as target filters. Additional mathematical details are available in Appendix E.1.

3.2 Numerical validation

We conduct extensive numerical experiments to validate our theoretical findings. Inspired by filter learning experiments from prior spectral GNN studies [24, 25, 47, 70], we design more challenging tasks with (i) increased graph sizes and (ii) complex target functions for learning. Specifically, we generate random graphs with 50,000 nodes, substantially larger than the typical 10,000-node setups in previous studies. Additionally, we utilize six intricate target filter functions, visualized in Figure 2. The experiments comprise two primary tasks:

- Using eigenvalue-based slices of each function, we assess the approximation quality of five polynomials commonly adopted in spectral GNN literature, with the sum of squared errors (SSE) across 50000 slices as the metric.
- With a random 50000×100 matrix as node feature $\boldsymbol{X}$, we apply six target functions as filters, obtaining output $\boldsymbol{Y}_1$ to $\boldsymbol{Y}_6$. We train spectral GNNs on $(\boldsymbol{X}, \boldsymbol{Y})$ to learn the target functions with polynomial filters, with the Frobenius norm of the difference between the learned and target filters as the metric.

Numerical insights. Table 1 reveals that reducing the sum of the polynomial approximation error over function slices yields lower filter learning errors in spectral GNNs, consistently ranking both tasks. Although these results are derived from numerical experiments and may introduce certain biases, they confirm our theoretical analysis, showing a strong positive relationship between the polynomial's capability and the efficacy of spectral GNNs.

3.3 Summary

In this section, we summarize the significant findings from the preceding analysis and delve into discussions on enhancing spectral GNNs through the introduction of informed polynomial selection.

Specifically, as discussed in Section 3.1 and 3.2, the construction error of spectral GNNs is intricately connected to the polynomial approximation error summed over function slices. Moreover, referring to Theorem 3.5, note that $\boldsymbol{X}$ is typically a constant property of

Table 1: Numerical experiment results. # Avg Rank 1 denotes the average rank in polynomial approximation, and # Avg Rank 2 refers to the average rank in filter learning.

Method		Slice-wise approximation						Filter Learning						# Avg	# Avg
Polynomial	GNN	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	Rank 1	Rank 2
Monomial	GPRGNN [10]	139.9	289.1	466.1	398.3	1.83	97.83	167.2	366.4	566.3	468.7	15.91	139.2	5	5
Bernstein	BernNet [25]	32.78	247.3	398.5	306.5	0.058	22.92	68.23	313.2	448.2	415.2	7.79	95.84	4	4
Chebyshev	ChebNetII [26]	23.45	85.19	244.8	187.2	0.018	13.13	64.22	168.4	402.5	347.5	6.83	86.25	3	3
Jacobian	JacobiConv [72]	22.18	80.77	239.2	155.3	0.017	11.82	48.56	95.92	338.1	266.4	5.33	65.13	2	2
Learnable	OptBasis [24]	20.75	80.53	225.7	152.7	0.017	11.20	43.44	89.48	289.5	238.1	4.98	61.70	1	1

the graph data, the construction error of graph convolution layer, ξ, therefore depends entirely on the slice-wise errors ϵ_s, $s = 1, 2, ..., n$. Consequently, for a graph data $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X})$ with node label $\boldsymbol{Y}$, an intuitive solution to reduce spectral GNN construction error is to utilize polynomials adept at approximating these slices.

Furthermore, practical graphs often contain millions of nodes [29, 44] and complex target filters [10, 25, 45, 75]. These characteristics result in very narrow and sharp slices of the target functions. Consequently, to minimize construction errors in spectral GNNs and improve their effectiveness, it is vital to incorporate "narrow function-preferred" polynomials in the development of graph filters. This insight not only represents a key contribution of this paper but also illuminates potential avenues for advancing spectral GNNs, paving the way for the introduction of a more sophisticated method in the subsequent section.

4 THE PROPOSED TFGNN

Building on our previous analysis, this section introduces a novel trigonometric polynomial-based graph filter to enhance spectral GNNs. We begin with the trigonometric filter, discuss its efficient implementation through Taylor-based parameter decomposition (TPD), and present our **T**rigonometric **F**ilter **G**raph **N**eural **N**etwork (TFGNN) as a decoupled GNN. A complexity analysis concludes the section.

4.1 Parameter-efficient trigonometric filter

Trigonometric polynomials, among the most extensively utilized, have found widespread applications in approximating the functions with complicated patterns [21, 63, 86]. More importantly, extensive prior studies in traditional signal processing domain have consistently highlighted the effectiveness of trigonometric polynomials over other polynomial types in modeling the functions localized within narrow intervals [12, 16, 22, 71, 78, 80]. This prompts us to pioneer the development of graph filters through leveraging the power of trigonometric polynomials. Explicitly, the definition of our trigonometric graph filter is as follows:

$$f_{\text{Trigo}}(\boldsymbol{\lambda}) = \sum_{k=0}^{K} \left[\alpha_k \sin(k\omega\boldsymbol{\lambda}) + \beta_k \cos(k\omega\boldsymbol{\lambda})\right] . \tag{11}$$

Here, K denotes the order of the truncated trigonometric polynomial. The coefficients α_k, and β_k are parameterized, while the hyperparameter ω (base frequency) is chosen from within the range $(0, \pi)$, enabling the trigonometric polynomial approximation to cover the interval $[0, 2]$, which corresponds to the range of $\boldsymbol{\lambda}$. similar to other types of polynomials, trigonometric polynomials offer considerable approximation capability for arbitrary functions, thus ensuring comprehensive filter coverage in practical applications [63, 86].

Guaranteed parameter-efficiency. Apart from their recognized approximation capabilities, trigonometric polynomials grant the trigonometric graph filter f_{Trigo} with a unique, provable efficiency regarding its parameters (α_k, β_k), $k \in \mathbb{N}$, as demonstrated in the theorem below.

Theorem 4.1. *(Parameter-efficiency). Given a $f(x)$ formulated as $f_{Trigo}(x)$, its coefficients α_k and β_k satisfies:*

$$\lim_{k\to+\infty} \alpha_k = 0 , \quad \lim_{k\to+\infty} \beta_k = 0 . \tag{12}$$

Proof can be found in Appendix D. Theorem 4.1 establishes a solid basis by revealing that polynomial terms with larger values of k within $f_{\text{Trigo}}(x)$ correspond to smaller weights. This insight indicates that the contribution of high-order terms is relatively insignificant, allowing for their practical omission without substantial loss in approximation accuracy. As a result, $f_{\text{Trigo}}(x)$ can achieve substantial effectiveness with only a small K, reducing the complexity of the trigonometric graph filters while retaining significant accuracy. This reinforces the superiority of $f_{\text{Trigo}}(x)$ over other graph filter designs.

4.2 Taylor-based parameter decomposition

As detailed in Eq. 11, implementing the standard $f_{\text{Trigo}}(x)$ requires an eigen-decomposition, which imposes substantial computational complexity and limits scalability compared to alternative methods. We tackle this challenge through the introduction of Taylor-based parameter decomposition (TPD). TPD first reformulates the trigonometric terms $\sin(k\omega\boldsymbol{\lambda})$ and $\cos(k\omega\boldsymbol{\lambda})$ into polynomial forms through the Taylor expansion [1, 58, 65], as shown below:

$$\sin(k\omega\boldsymbol{\lambda}) = \sum_{d=0}^{D} \gamma_{kd}\boldsymbol{\lambda}^d, \ \cos(k\omega\boldsymbol{\lambda}) = \sum_{d=0}^{D} \theta_{kd}\boldsymbol{\lambda}^d . \tag{13}$$

The constants γ_{kd} and θ_{kd} depend exclusively on the types of functions (sine and cosine), the index k, and the hyperparameter ω. The effectiveness of Taylor expansion for modeling functions within localized intervals has been thoroughly established in the literature [32, 50, 52, 54, 57, 83], especially for trigonometric functions [7, 33, 36]. Since $\boldsymbol{\lambda}$ is restricted to the range $[0, 2]$, the Taylor expansion emerges as a viable and crucial strategy for efficiently approximating these trigonometric functions.

With the updated formulations, TPD alters the convolution operation with $f_{\text{Trigo}}(x)$ on the node feature $\boldsymbol{X}$ as detailed below:

$$\begin{aligned} \boldsymbol{Z} &= \boldsymbol{U} \sum_{k=0}^{K} \left[\alpha_k \sum_{d=0}^{D} \gamma_{kd} diag(\boldsymbol{\lambda}^d) + \beta_k \sum_{d=0}^{D} \theta_{kd} diag(\boldsymbol{\lambda}^d) \right] \boldsymbol{U}^T \boldsymbol{X} \,, \\ &= \sum_{d=0}^{D} \boldsymbol{L}^d \boldsymbol{X} \left(\boldsymbol{\alpha} \boldsymbol{\Gamma}_{:d} + \boldsymbol{\beta} \boldsymbol{\Theta}_{:d} \right) \,. \end{aligned} \tag{14}$$

Here, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ denote the $K+1$-dimensional vectors with elements being α_k and β_k, respectively. $\boldsymbol{\Gamma}$ and $\boldsymbol{\Theta}$ refer to the $(K+1)\times(D+1)$ matrices formed with γ_{kd}, θ_{kd}. Eq. 14 illustrates a streamlined convolution with $f_{\text{Trigo}}(x)$, offering two significant benefits:

- **Reduced complexity.** Utilizing the TPD, the graph convolution with $f_{\text{Trigo}}(x)$ eliminates the need for computation-intensive eigen-decomposition. This reduction in computational overhead brings the costs in line with those of standard polynomial-based filters, leading to significant efficiency gains.
- **Parameter decomposition.** TPD integrates all trigonometric functions into polynomial forms, allowing for increases in K to only influence trivial computations of $\boldsymbol{\alpha}\boldsymbol{\Gamma}$ and $\boldsymbol{\beta}\boldsymbol{\Theta}$, enhancing precision of $f_{\text{Trigo}}(x)$ with negligible additional cost.

4.3 Modeling TFGNN as decoupled paradigm

TFGNN is a decoupled GNN architecture that separates graph convolution from feature transformation. This design principle, first proposed by [19], has become a de facto choice in modern spectral GNNs for its significant efficacy and computational efficiency [10, 24–26, 30, 31, 37–39, 72], and even stands out as a promising solution for scalable GNNs [42, 43]. Specifically, incorporating the trigonometric filter $f_{\text{Trigo}}(x)$ with the introduced Taylor-based parameter decomposition, we present two versions of TFGNN to cater to different graph sizes:

❶ In the case of **medium-to-large** graphs like Cora [77] and Arxiv [29], TFGNN operates as described below:

$$\boldsymbol{Z} = \sum_{d=0}^{D} \boldsymbol{L}^d \boldsymbol{H} \left(\boldsymbol{\alpha} \boldsymbol{\Gamma}_{:d} + \boldsymbol{\beta} \boldsymbol{\Theta}_{:d} \right), \quad \boldsymbol{H} = \text{MLP}(\boldsymbol{X}) \,. \tag{15}$$

$\text{MLP}(\cdot)$ denotes a multi-layer perceptron for feature transformation.

❷ In the case of **exceptionally large** graphs, such as Wiki [44] and Papers100M [29], TFGNN is implemented as follows:

$$\boldsymbol{Z} = \text{MLP}(\boldsymbol{H}), \quad \boldsymbol{H} = \sum_{d=0}^{D} \boldsymbol{L}^d \boldsymbol{X} \left(\boldsymbol{\alpha} \boldsymbol{\Gamma}_{:d} + \boldsymbol{\beta} \boldsymbol{\Theta}_{:d} \right) \,, \tag{16}$$

The different implementations of TFGNN come from hardware constraints and introduce notable benefits: (i) for medium-to-large graphs, the graph data can be fully stored on GPUs; therefore, by simply reducing the feature dimensions with MLP, the subsequent convolution process could achieves high efficiency; (ii) for exceptionally large graphs, where GPU memory limitations become a substantial challenge, TFGNN precomputes features, $\boldsymbol{L}^k\boldsymbol{X}$, $k = 1, 2, ...K$, and stores them as static data files. This allows for efficient graph convolution operation via repeated reads of precomputed features, mitigating the intense computational complexity associated with GNN training; an efficient MLP is applied later.

Table 2: Complexity comparison of TFGNN against others. The complexity pertains to the graph convolution layers.

Method	Computation	Parameter
GPRGNN [10]	$O(mED)$	$O(K+1)$
ChebNetII [26]	$O(mED)$	$O(K+1)$
OptBasis [24]	$O(mED)$	$O(K+1)$
JacobiConv [72]	$O(mED)$	$O(K+1)$
UniFilter [31]	$O(mED)$	$O(K+1)$
TFGNN (**ours**)	$O(mED)$	$O(2(K+1))$

4.4 Complexity analysis of TFGNN

This subsection presents the complexity analysis of TFGNN, with a particular emphasis on graph convolution layers, as the complexity of feature transformation MLPs is already well-understood. To start, we consider a graph $\mathcal{G}$ with n nodes, E edges, and m feature dimensions. Across all spectral GNNs, the maximum polynomial order is set to D. The trigonometric polynomial degree is capped at K. A summary of the complexity analysis is outlined in Table 2.

Computational complexity. As shown in Eq. 14, for each order d, TFGNN first computes $\boldsymbol{\alpha}\boldsymbol{\Gamma}_{:d}$ and $\boldsymbol{\beta}\boldsymbol{\Theta}_{:d}$, requiring $2(K+1)$ operations, followed by propagating with $\boldsymbol{L}$, which requires mE computations. Since the number of edges E is millions of times larger than both K and D, the practical complexity is governed by mE for each order d, leading to an overall complexity of $O(mED)$. This is comparable to other spectral GNNs like GPRGNN [10], which utilizes recursive computation of the propagated feature $\boldsymbol{L}^d\boldsymbol{X}$. Thus, our TFGNN achieves complexity on par with prior methods. Additionally, for exceptionally large graphs, TFGNN reduces complexity further by precomputing all $\boldsymbol{L}^d\boldsymbol{X}$, thus avoiding redundant repeated computations during training.

Parameter complexity. Our TFGNN achieves a parameter complexity of $O(2(K+1))$, in contrast to traditional spectral GNNs' $O(K+1)$, where each polynomial basis order is assigned a parameterized coefficient. This increase is, however, trivial, as the feature transformation MLP constitutes the majority of parameters, greatly outweighing the graph convolution layers. As such, TFGNN's parameter complexity remains effectively on par with that of other spectral GNNs when considering the entire model.

5 EMPIRICAL STUDIES

This section details the empirical evaluations, including numerical experiments same as those in Section 3.2, a benchmark node classification task, and a practical application in graph anomaly detection. A demo code implementation is available through the anonymous link https://anonymous.4open.science/r/TFGNN-7ED8/.

5.1 Slice approximation and filter learning

We conduct the same numerical experiments as outlined in Section 3.2 to evaluate the proposed trigonometric graph filters and TFGNN. To ensure a fair and informative comparison, the trigonometric polynomial used in our numerical experiments is not in its naive form; rather, we employ the formulation that incorporates a 10 degree Taylor-based parameter decomposition (TPD), akin in that of Section 4.2. The trigonometric polynomial degree, K, is set

Table 3: Node classification results on medium-to-large graphs. #Improv. denotes the performance gain of TFGNN over the best baseline result. Boldface represents the first result, while underlined indicates the runner-up.

GNN Type	Method	homophilic graphs				heterophilic graphs				
		Cora	Cite.	Pubmed	Arxiv	Roman.	Amazon.	Ques.	Gamers	Genius
i	H2GCN	87.33±0.6	75.11±1.2	88.39±0.6	71.93±0.4	61.38±1.2	37.17±0.5	64.42±1.3	64.71±0.4	90.12±0.2
	GLOGNN	88.12±0.4	76.23±1.4	88.83±0.2	72.08±0.3	71.17±1.2	42.19±0.6	74.42±1.3	65.62±0.3	90.39±0.3
	LINKX	84.51±0.6	73.25±1.5	86.36±0.6	71.14±0.2	67.55±1.2	41.57±0.6	63.85±0.8	65.82±0.4	91.12±0.5
	OrderGNN	87.55±0.2	75.46±1.2	88.31±0.3	71.90±0.5	71.69±1.6	40.93±0.5	70.82±1.0	66.09±0.3	89.45±0.4
	LRGNN	87.48±0.3	75.29±1.0	88.65±0.4	71.69±0.3	72.35±1.4	42.56±0.4	71.82±1.1	66.29±0.5	90.38±0.7
ii	GCN	86.48±0.4	75.23±1.0	87.29±0.2	71.77±0.1	72.33±1.6	42.09±0.6	75.17±0.8	63.29±0.5	86.73±0.5
	GCNII	86.77±0.2	76.57±1.5	88.86±0.4	71.72±0.4	71.62±1.7	40.89±0.4	72.32±1.0	65.11±0.3	90.60±0.6
	ChebNet	86.83±0.7	74.39±1.3	85.92±0.5	71.52±0.3	64.44±1.5	38.81±0.7	70.42±1.2	63.62±0.4	87.42±0.2
	ACMGCN	87.21±0.4	76.03±1.4	87.37±0.4	71.70±0.3	66.48±1.2	39.53±0.9	67.84±0.5	64.73±0.3	83.45±0.7
	Specformer	88.19±0.6	75.87±1.5	88.74±0.2	71.88±0.2	71.69±1.4	42.06±0.8	70.75±1.2	65.80±0.2	89.39±0.6
iii	GPRGNN	88.26±0.5	76.24±1.2	88.81±0.2	71.89±0.2	64.49±1.6	41.48±0.6	64.58±1.2	66.23±0.1	90.92±0.6
	BernNet	87.57±0.4	75.81±1.8	88.48±0.3	71.72±0.3	65.44±1.4	40.74±0.7	65.53±1.6	65.74±0.3	89.75±0.3
	ChebNetII	88.17±0.4	76.41±1.3	88.98±0.4	72.13±0.3	66.77±1.2	42.44±0.9	71.28±0.6	66.44±0.5	90.60±0.2
	OptBasis	88.35±0.6	76.22±1.4	89.38±0.3	72.10±0.2	64.28±1.8	41.63±0.8	69.60±1.2	66.81±0.4	90.97±0.5
	JacobiConv	88.53±0.8	76.27±1.3	89.51±0.2	71.87±0.3	70.10±1.7	42.18±0.4	72.16±1.3	64.17±0.3	89.32±0.5
	NFGNN	88.06±0.4	76.22±1.4	88.43±0.4	72.15±0.3	72.46±1.2	42.19±0.3	75.49±0.9	66.64±0.4	90.87±0.5
	AdaptKry	88.23±0.7	76.54±1.2	88.38±0.6	72.33±0.3	71.40±1.3	42.31±1.1	72.55±1.0	66.27±0.3	90.55±0.3
	UniFilter	88.31±0.7	76.38±1.1	89.30±0.4	72.87±0.4	71.22±1.5	41.37±0.6	73.83±0.8	65.75±0.4	90.66±0.2
	TFGNN (**Ours**)	**89.21±0.4**	**77.68±0.8**	**90.00±0.2**	**75.23±0.2**	**74.94±1.1**	**45.04±0.6**	**81.55±0.9**	**69.46±0.2**	**92.40±0.2**
	#Improv.	0.68%	1.11%	0.49%	2.36%	2.48%	2.48%	6.06%	2.65%	1.28%

Table 4: Comparison of trigonometric filter with counterparts. Full results are presented in Table 12.

Method		Poly. approx.			Filter Learn.		
Poly.	GNN	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
Cheby.	ChebNetII	85.19	244.8	187.2	168.4	402.5	347.5
Jacobi.	JacobiConv	80.77	239.2	155.3	95.92	338.1	266.4
Learn.	OptBasis	80.53	225.7	152.7	89.48	289.5	238.1
Trigo.	**TFGNN**	**23.69**	**71.13**	**59.88**	**65.19**	**102.3**	**105.3**

to 10, yielding $K+1$ coefficients in total. Thus, TFGNN preserves both the maximum order of $\boldsymbol{\lambda}$ and the number of coefficients used by other counterparts.

Results. Table 4 summarizes the performance of TFGNN alongside the three leading alternatives—Chebyshev, Jacobian, and Learnable—with boldface marking the highest scores due to space constraints. A comprehensive comparison can be found in Appendix F.1. According to these results, Trigonometric polynomials (with our TPD) and TFGNN consistently outperform other methods. Particularly, for target functions exhibiting complex patterns, such as $f_2(x)$, $f_3(x)$, and $f_4(x)$, TFGNN obtains notable improvements over competitors, showing the efficacy of our method.

The following sections will show how TFGNN attains leading performance on real-world datasets, affirming that the numerical outcomes correspond well to real-world scenarios.

5.2 Benchmark node classification tasks

This section evaluates TFGNN against counterparts through benchmark node classification tasks.

5.2.1 Datasets and baselines.

Datasets. We utilize 13 benchmark datasets with varied sizes and heterophily levels [82]. For homophilic datasets, we comprise citation graphs (Cora, CiteSeer, PubMed)[77] and large OGB graphs (ogbn-Arxiv, ogbn-Products, ogbn-Papers100M)[29]. For heterophilic datasets, we select three latest datasets (Roman-empire, Amazon-ratings, Questions) [59] and four large ones (Gamers, Genius, Snap-patent, Pokec) [44]. (We exclude conventional dataset choices [56] due to the recognized data-leakage issues in these datasets [59].)

Baselines and settings. We include 18 advanced baselines tailored for both heterophilic and homophilic scenarios, which can be categorized into three classes as follows:

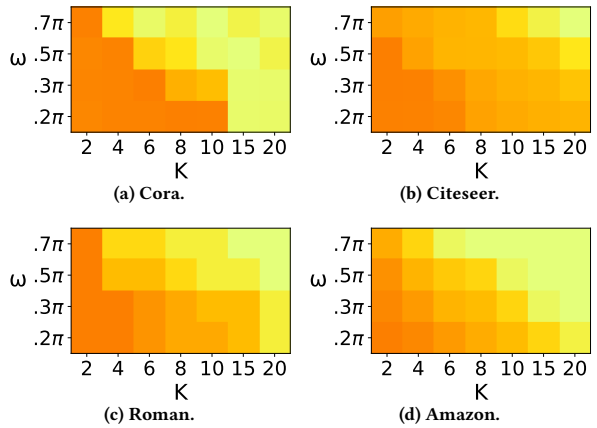
- **Non-spectral GNNs:** H2GCN [85], GLOGNN [40], LINKX [44], OrderGNN [66], LRGNN [41].
- **Non-decoupled spectral GNNs:** GCN [35], GCNII [8], ChebNet [13], ACMGCN [49], Specformer [2].
- **Decoupled spectral GNNs:** GPRGNN [10], BernNet [25], ChebNetII [26], OptBasis [24], NFGNN [81], JacobiConv [72], AdaptKry [30], UniFilter [31].

For the widely adopted baselines (GCN and ChebNet), we adopt consistent implementations drawn from previous research [24–26, 30, 39, 68, 72, 81]. For the remaining baselines, we inherit the hyperparameter tuning settings from their original publications.

Implementation of TFGNN. To ensure experimental fairness, we fix the order of the Trigonometric Polynomial Decomposition (TPD), denoted as D, to 10, aligning with other baselines such as GPRGNN and ChebNetII. We employ a grid search to optimize the parameters

Table 5: Node classification and runtime (*hours*) results on exceptionally large graphs. "OOM" denotes "Out-Of-Memory".

Method	Products		Papers100M		Snap		Pokec	
	Test acc	Runtime	Test acc	Runtime	Test acc	Runtime	Test acc	Runtime
GCN	$76.37_{\pm 0.2}$	1.2	OOM	-	$46.66_{\pm 0.1}$	1.9	$74.78_{\pm 0.2}$	1.2
SGC	$75.16_{\pm 0.2}$	0.9	$64.02_{\pm 0.2}$	10.2	$31.11_{\pm 0.2}$	1.6	$60.29_{\pm 0.1}$	0.9
GPRGNN	$79.45_{\pm 0.1}$	1.3	$66.13_{\pm 0.2}$	11.1	$48.88_{\pm 0.2}$	2.0	$79.55_{\pm 0.3}$	1.2
BernNet	$79.82_{\pm 0.2}$	1.3	$66.08_{\pm 0.2}$	11.2	$47.48_{\pm 0.3}$	2.1	$80.55_{\pm 0.2}$	1.3
ChebNetII	$81.66_{\pm 0.3}$	1.2	$\underline{67.11_{\pm 0.2}}$	11.0	$51.74_{\pm 0.2}$	1.9	$81.88_{\pm 0.3}$	1.2
JacobiConv	$79.35_{\pm 0.2}$	1.0	$65.45_{\pm 0.2}$	10.5	$50.66_{\pm 0.2}$	1.7	$73.83_{\pm 0.2}$	1.0
OptBasis	$81.33_{\pm 0.2}$	1.3	$67.03_{\pm 0.3}$	11.2	$53.55_{\pm 0.1}$	2.1	$82.09_{\pm 0.3}$	1.3
NFGNN	$81.11_{\pm 0.2}$	1.3	$66.38_{\pm 0.2}$	11.3	$\underline{57.83_{\pm 0.3}}$	2.1	$81.56_{\pm 0.3}$	1.4
AdaptKry	$\underline{81.70_{\pm 0.3}}$	1.4	$67.07_{\pm 0.2}$	11.3	$55.92_{\pm 0.2}$	2.1	$82.16_{\pm 0.2}$	1.4
UniFilter	$80.33_{\pm 0.2}$	1.2	$66.79_{\pm 0.3}$	11.0	$52.06_{\pm 0.1}$	2.1	$\underline{82.23_{\pm 0.3}}$	1.3
TFGNN (**Ours**)	$\mathbf{84.05_{\pm 0.2}}$	1.2	$\mathbf{68.65_{\pm 0.2}}$	11.0	$\mathbf{64.38_{\pm 0.2}}$	1.9	$\mathbf{85.55_{\pm 0.2}}$	1.2
#Improv.	2.35%	-	1.54%	-	6.55%	-	3.32%	-



(a) Cora. (b) Citeseer. (c) Roman. (d) Amazon.

Figure 3: Ablation studies on K and ω. Darker shades indicate higher results. Additional results are in Appendix F.2.

Table 6: Ablation studies on Taylor expansion degree.

Degree	5	10	15	20	25
Cora	$88.66_{\pm 0.3}$	$89.21_{\pm 0.4}$	$89.15_{\pm 0.2}$	$89.53_{\pm 0.3}$	$89.28_{\pm 0.3}$
Arxiv	$73.14_{\pm 0.2}$	$75.23_{\pm 0.2}$	$74.74_{\pm 0.2}$	$75.06_{\pm 0.2}$	$74.92_{\pm 0.2}$
Roman.	$72.67_{\pm 1.0}$	$74.94_{\pm 1.1}$	$74.83_{\pm 1.1}$	$74.92_{\pm 1.2}$	$75.02_{\pm 1.1}$
Genius	$90.02_{\pm 0.3}$	$92.40_{\pm 0.2}$	$91.88_{\pm 0.3}$	$91.83_{\pm 0.2}$	$92.05_{\pm 0.3}$

ω within $\{0.2\pi, 0.3\pi, 0.5\pi, 0.7\pi\}$ and K from $\{2, 4, 6, 8, 10, 15, 20\}$. Additional details are in Appendix E.2.

5.2.2 Main results and discussions.

Effectiveness of TFGNN. Our TFGNN achieves remarkable advancements in performance on both heterophilic and homophilic graphs. Specifically, across all 13 datasets, TFGNN not only leads in performance but does so with improvements of up to **6.55** over the closest competitor on the Snap-patents dataset.

Furthermore, the advantages of TFGNN are significantly more pronounced when evaluated on heterophilic datasets. This trend is corroborated by the numerical findings in Table 4, which reveal TFGNN's enhanced capacity to construct functions that can accommodate complex patterns. Existing studies have empirically shown that heterophilic graphs generally require significantly more complex target filters than the low-pass filters used for homophilic graphs [25, 30, 75]. While these complex functions can complicate performance for other methods, TFGNN utilizes its advanced trigonometric filters to navigate these challenges, yielding substantial improvements on heterophilic scenarios.

Scalability and Efficiency. Table 5 presents a comparative analysis of our TFGNN method alongside leading counterparts, with each baseline recognized for its exceptional scalability and efficiency on large graphs. Notably, TFGNN demonstrates superior performance, significantly exceeding all baselines across every dataset while maintaining efficiency comparable to the top-performing methods. These findings align with our expectations, as the model complexity—both in terms of computation and parameters—of TFGNN is on par with that of other approaches, as detailed in Section 4.4.

5.2.3 Ablation studies.

Ablation studies on K and ω. We conduct ablation studies on the two pivotal hyperparameters, K and ω, associated with our trigonometric filters. Partial results are illustrated in Figure 3, while a more comprehensive analysis can be found in Appendix F.2.

The results reveal a notable trend: for all datasets, the optimal values for K and ω tend to fall within low ranges, specifically $K \in \{2, 4, 6\}$ and $\omega \in \{0.2\pi, 0.3\pi, 0.5\pi\}$. Furthermore, their product $K \cdot \omega$ consistently converges to a similar range across all datasets, approximately $K \cdot \omega \in (0.6\pi, 1.2\pi)$. This finding aligns with Theorem 4.1, which indicates that high-degree terms contribute unnecessary complexity. We thus recommend initializing K, ω, and $K \cdot \omega$ within these ranges for efficient use of our models, with further fine-tuning as needed for performance optimization.

Ablation studies on D. We perform ablation studies on the degree of Taylor expansion, D. Table 6 shows that while increasing the degree improves performance to a certain extent, accuracy eventually stabilizes. This is consistent with prior studies and can be understood in terms of polynomial approximation. Higher-degree orthogonal bases tend to minimize approximation loss; however,

Table 7: Graph anomaly detection results. ‡ Improvements are relative to general-purpose methods rather than GAD baselines.

Type	Dateset	YelpChi (1%)		Amazon (1%)		T-Finance (1%)	
	Metric	F1-macro	AUROC	F1-macro	AUROC	F1-macro	AUROC
GAD Models	PC-GNN	60.55	75.29	82.62	91.61	83.40	**91.85**
	CARE-GNN	61.68	73.95	75.78	88.79	86.03	91.17
	GDN	65.72	75.33	90.49	**92.07**	77.38	89.42
GAD-specialized spectral GNNs	BWGNN	**66.52**	77.23	90.28	89.19	85.56	91.38
	GHRN	62.77	74.64	86.65	87.09	80.70	91.55
General-purpose spectral GNNs	GCN	50.66	54.31	69.79	85.18	75.26	87.05
	GPRGNN	60.45	67.44	83.71	85.28	77.53	85.69
	OptBasis	62.03	68.32	86.12	85.02	79.28	86.22
	AdaptKry	63.40	66.18	83.30	84.58	80.67	85.41
	NFGNN	60.66	67.36	85.61	86.88	82.38	86.59
Ours	TFGNN	65.60	**78.79**	**91.10**	90.12	**87.02**	91.42
	#Improv.‡	2.20%	10.47%	4.98%	3.24%	4.64%	4.37%

beyond an optimal degree, further improvements become negligible [58, 65].

5.3 Application example: graph anomaly detection (GAD)

This section investigates an application example of TFGNN for the graph anomaly detection (GAD) task, which is typically recognized as binary node classification task (normal vs. abnormal) [51, 60].

5.3.1 Datasets and baselines.

Datasets. We adopt three datasets (YelpChi, Amazon, and T-Finance) with a low label-rate of 1% set across all datasets, while T-Finance additionally utilizes a higher label-rate of 40%, as described in [69].

Baselines and model implementations. We include 10 baseline methods, organized into three types below:

- **GAD models:** PC-GNN [46], CARE-GNN [14], GDN [18].
- **GAD-specialized spectral GNNs:** BWGNN [69], GHRN [17].
- **General-purpose spectral GNNs:** GCN [35], GPRGNN [10], OptBasis [24], AdaptKry [30], NFGNN [81].

The specifications for implementing common baselines (PC-GNN, CARE-GNN, BWGNN, GCN) are derived from [69]. In our TFGNN and other general-purpose methods, we utilize a two-layer MLP with 64 hidden units for the feature transformation module, maintaining alignment with the GAD-specialized models. The hyperparameters for TFGNN are optimized as detailed in Section 5.2, while the other baselines follow the configurations outlined in their original papers. More experimental details are in Appendix E.3.

5.3.2 Main results and discussions.

Improvements on specific task. Table 7 highlights the #Improv. metric, showing that TFGNN outperforms general-purpose models significantly, achieving increases of up to 11.34% on the YelpChi dataset. This suggests that while general-purpose spectral GNNs can perform well in benchmark node classification tasks, they often underperform in specialized applications. In contrast, TFGNN, with its advanced graph filters, consistently provides notable improvements across both standard and specialized tasks, demonstrating the effectiveness and versatility of our approach.

Comparable to GAD-specialized spectral GNNs. Table 7 indicates that TFGNN's performance rivals that of specialized spectral GNNs for GAD. Models like BWGNN and GHRN, which are built on the same graph spectrum principles, incorporate specific features aimed at enhancing performance. For example, BWGNN [69] effectively addresses the "right-shift" phenomenon with its customized beta wavelets, while GHRN [17] focuses on filtering out high-frequency components to prune inter-class edges in heterophilic graphs. In contrast, TFGNN offers a unique and effective filtering strategy for GAD tasks, showcasing impressive outcomes. This reflects a promising direction for improving spectral GNNs through the introduction of more advanced polynomial graph filters.

6 CONCLUSIONS

In this paper, we address the polynomial selection problem in spectral GNNs, linking polynomial capabilities to their effectiveness. We present the first proof that the construction error of graph convolution layers is bounded by the sum of polynomial approximation errors on function slices, supported by intuitive numerical validations. This insight motivates the use of "narrow function-preferred" polynomials, leading to the introduction of our advanced trigonometric graph filters. The proposed filters not only demonstrate provable parameter-efficiency but also employ Taylor-based parameter decomposition for streamlined implementation. Building upon this, we introduce TFGNN, a scalable spectral GNN featuring a decoupled architecture. The efficacy of TFGNN is confirmed through benchmark node classification tasks and a practical example in graph anomaly detection, highlighting the adaptability and real-world relevance of our theoretical contributions.

Limitations and future works. Our theoretical framework is grounded in the concept of function slices, which are inherently linked to target filters. However, in practical scenarios, the diversity and variability of target filters can hinder the specificity of our theoretical results, potentially leading to suboptimal solutions. A promising future research is to categorize these filters and analyze their numerical properties. With these insights, we can refine our theoretical framework, thereby enabling more consistent enhancements in spectral GNNs.

REFERENCES

[1] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. 2021. Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective. In International Conference on Learning Representations. https://openreview.net/forum?id=-qh0M9XWxnv

[2] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. 2023. Specformer: Spectral Graph Neural Networks Meet Transformers. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=0pdSt3oyJa1

[3] Deyu Bo, Xiao Wang, Yang Liu, Yuan Fang, Yawen Li, and Chuan Shi. 2023. A Survey on Spectral Graph Neural Networks. arXiv:2302.05631 [cs.LG]

[4] Salomon Bochner and Komaravolu Chandrasekharan. 1951. Fourier Transforms. The Mathematical Gazette 35, 312 (1951), 140–141. https://doi.org/10.2307/3609365

[5] Fedor Borisyuk, Shihai He, Yunbo Ouyang, Morteza Ramezani, Peng Du, Xiaochen Hou, Chengming Jiang, Nitin Pasumarthy, Priya Bannur, Birjodh Tiwana, Ping Liu, Siddharth Dangi, Daqi Sun, Zhoutao Pei, Xiao Shi, Sirou Zhu, Qianqi Shen, Kuang-Hsuan Lee, David Stein, Baolei Li, Haichao Wei, Amol Ghoting, and Souvik Ghosh. 2024. LiGNN: Graph Neural Networks at LinkedIn (KDD '24). Association for Computing Machinery, New York, NY, USA, 4793–4803. https://doi.org/10.1145/3637528.3671566

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In International Conference on Learning Representations (ICLR2014).

[7] Claudio Brunelli, Heikki Berg, and David Guevorkian. 2009. Approximating sine functions using variable-precision Taylor polynomials. In 2009 IEEE Workshop on Signal Processing Systems. 57–62. https://doi.org/10.1109/SIPS.2009.5336225

[8] 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 (Proceedings of Machine Learning Research, Vol. 119). PMLR, 1725–1735. https://proceedings.mlr.press/v119/chen20v.html

[9] Zhengdao Chen, Lisha Li, and Joan Bruna. 2019. Supervised Community Detection with Line Graph Neural Networks. In International Conference on Learning Representations. https://openreview.net/forum?id=H1g0Z3A9Fm

[10] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In International Conference on Learning Representations. https://openreview.net/forum?id=n6jl7fLxrP

[11] Fan Chung. 1997. Spectral Graph Theory. Vol. 92. CBMS Regional Conference Series in Mathematics. https://doi.org//10.1090/cbms/092

[12] T.N. Davidson, Zhi-Quan Luo, and J.F. Sturm. 2002. Linear matrix inequality formulation of spectral mask constraints with applications to FIR filter design. IEEE Transactions on Signal Processing 50, 11 (2002), 2702–2715. https://doi.org/10.1109/TSP.2002.804079

[13] 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, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc.

[14] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. 2020. Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (Virtual Event, Ireland) (CIKM '20). Association for Computing Machinery, New York, NY, USA, 315–324. https://doi.org/10.1145/3340531.3411903

[15] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. https://doi.org/10.48550/ARXIV.1903.02428

[16] Dengwei Fu and A.N. Willson. 1999. Design of an improved interpolation filter using a trigonometric polynomial. In 1999 IEEE International Symposium on Circuits and Systems (ISCAS), Vol. 4. 363–366 vol.4. https://doi.org/10.1109/ISCAS.1999.780017

[17] Yuan Gao, Xiang Wang, Xiangnan He, Zhenguang Liu, Huamin Feng, and Yongdong Zhang. 2023. Addressing Heterophily in Graph Anomaly Detection: A Perspective of Graph Spectrum. In Proceedings of the ACM Web Conference 2023 (Austin, TX, USA) (WWW '23). Association for Computing Machinery, New York, NY, USA, 1528–1538. https://doi.org/10.1145/3543507.3583268

[18] Yuan Gao, Xiang Wang, Xiangnan He, Zhenguang Liu, Huamin Feng, and Yongdong Zhang. 2023. Alleviating Structural Distribution Shift in Graph Anomaly Detection. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining (Singapore, Singapore) (WSDM '23). Association for Computing Machinery, New York, NY, USA, 357–365. https://doi.org/10.1145/3539597.3570377

[19] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In International Conference on Learning Representations. https://openreview.net/forum?id=H1gL-2A9Ym

[20] Chenghua Gong, Yao Cheng, Xiang Li, Caihua Shan, and Siqiang Luo. 2024. Learning from Graphs with Heterophily: Progress and Future. arXiv:2401.09769 [cs.SI] https://arxiv.org/abs/2401.09769

[21] Jogh G.Proakis. 1975. Digital signal processing. IEEE Transactions on Acoustics, Speech, and Signal Processing 23, 4 (1975), 392–394. https://doi.org/10.1109/TASSP.1975.1162707

[22] Lucy J. Gudino, Joseph X. Rodrigues, and S. N. Jagadeesha. 2008. Linear phase FIR filter for narrow-band filtering. In 2008 International Conference on Communications, Circuits and Systems. 776–779. https://doi.org/10.1109/ICCCAS.2008.4657886

[23] Yuhe Guo and Zhewei Wei. 2023. Clenshaw Graph Neural Networks. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23). Association for Computing Machinery, New York, NY, USA, 614–625. https://doi.org/10.1145/3580305.3599275

[24] Yuhe Guo and Zhewei Wei. 2023. Graph Neural Networks with Learnable and Optimal Polynomial Bases. In Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202), Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 12077–12097. https://proceedings.mlr.press/v202/guo23i.html

[25] 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, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=WigDnV-_Gq

[26] Mingguo He, Zhewei Wei, and Ji-Rong Wen. 2022. Convolutional Neural Networks on Graphs with Chebyshev Approximation, Revisited. In Advances in Neural Information Processing Systems, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=jxPJ4QA0KAb

[27] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. https://doi.org/10.48550/ARXIV.2002.02126

[28] Roland F Hoskins. 2009. Delta functions: Introduction to generalised functions. Horwood Publishing.

[29] 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, Vol. 33. 22118–22133. https://proceedings.neurips.cc/paper_files/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf

[30] Keke Huang, Wencai Cao, Hoang Ta, Xiaokui Xiao, and Pietro Liò. 2024. Optimizing Polynomial Graph Filters: A Novel Adaptive Krylov Subspace Approach. In Proceedings of the ACM Web Conference 2024 (Singapore, Singapore) (WWW '24). Association for Computing Machinery, New York, NY, USA, 1057–1068. https://doi.org/10.1145/3589334.3645705

[31] Keke Huang, Yu Guang Wang, Ming Li, and Pietro Lio. 2024. How Universal Polynomial Bases Enhance Spectral Graph Neural Networks: Heterophily, Over-smoothing, and Over-squashing. In Forty-first International Conference on Machine Learning. https://openreview.net/forum?id=Z2LH6Va7L2

[32] Kafetzis Ioannis, Moysis Lazaros, and Volos Christos. 2023. Assessing the chaos strength of Taylor approximations of the sine chaotic map. Nonlinear Dynamics 111 (2023), 2755–2778. https://doi.org/10.1007/s11071-022-07929-y

[33] E. Stine James and J. Schulte Michael. 1999. The Symmetric Table Addition Method for Accurate Function Approximation. Journal of VLSI signal processing systems for signal, image and video technology 21 (1999), 167–177. https://doi.org/10.1023/A:1008004523235

[34] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/ARXIV.1412.6980

[35] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In International Conference on Learning Representations. https://openreview.net/forum?id=SJU4ayYgl

[36] B. Lee and N. Burgess. 2003. Some results on Taylor-series function approximation on FPGA. In The Thrity-Seventh Asilomar Conference on Signals, Systems and Computers, 2003, Vol. 2. 2198–2202. https://doi.org/10.1109/ACSSC.2003.1292370

[37] Runlin Lei, Zhen Wang, Yaliang Li, Bolin Ding, and Zhewei Wei. 2022. EvenNet: Ignoring Odd-Hop Neighbors Improves Robustness of Graph Neural Networks. In Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 4694–4706. https://openreview.net/forum?id=SPoiDLr3WE7

[38] Bingheng Li, Erlin Pan, and Zhao Kang. 2024. PC-Conv: Unifying Homophily and Heterophily with Two-Fold Filtering. Proceedings of the AAAI Conference on Artificial Intelligence 38, 12 (Mar. 2024), 13437–13445. https://doi.org/10.1609/aaai.v38i12.29246

[39] Guoming Li, Jian Yang, Shangsong Liang, and Dongsheng Luo. 2024. Spectral GNN via Two-dimensional (2-D) Graph Convolution. arXiv:2404.04559 [cs.LG]

[40] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. 2022. Finding Global Homophily in Graph Neural Networks When Meeting Heterophily. In Proceedings of the 39th International Conference on

Machine Learning (Proceedings of Machine Learning Research, Vol. 162), Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 13242–13256. https://proceedings.mlr.press/v162/li22ad.html

[41] Langzhang Liang, Xiangjing Hu, Zenglin Xu, Zixing Song, and Irwin King. 2023. Predicting Global Label Relationship Matrix for Graph Neural Networks under Heterophily. In Advances in Neural Information Processing Systems, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 10909–10921. https://proceedings.neurips.cc/paper_files/paper/2023/file/23aa2163dea287441ebebc1295d5b3fc-Paper-Conference.pdf

[42] Ningyi Liao, Siqiang Luo, Xiang Li, and Jieming Shi. 2023. LD2: Scalable Heterophilous Graph Neural Network with Decoupled Embeddings. In Thirty-seventh Conference on Neural Information Processing Systems. https://openreview.net/forum?id=7zkFc9TGKz

[43] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2024. Scalable decoupling graph neural network with feature-oriented optimization. The VLDB Journal 33, 3 (2024), 667–683.

[44] Derek Lim, Felix Matthew Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Prasad 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, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=DfGu8WwT0d

[45] Vijay Lingam, Manan Sharma, Chanakya Ekbote, Rahul Ragesh, Arun Iyer, and Sundararajan Sellamanickam. 2023. A Piece-Wise Polynomial Filtering Approach for Graph Neural Networks. In Machine Learning and Knowledge Discovery in Databases. Springer International Publishing, Cham, 412–452.

[46] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection. In Proceedings of the Web Conference 2021 (Ljubljana, Slovenia) (WWW '21). Association for Computing Machinery, New York, NY, USA, 3168–3177. https://doi.org/10.1145/3442381.3449989

[47] Kangkang Lu, Yanhua Yu, Hao Fei, Xuan Li, Zixuan Yang, Zirui Guo, Meiyu Liang, Mengran Yin, and Tat-Seng Chua. 2024. Improving Expressive Power of Spectral Graph Neural Networks with Eigenvalue Correction. Proceedings of the AAAI Conference on Artificial Intelligence 38, 13 (Mar. 2024), 14158–14166. https://doi.org/10.1609/aaai.v38i13.29326

[48] Sitao Luan, Chenqing Hua, Qincheng Lu, Liheng Ma, Lirong Wu, Xinyu Wang, Minkai Xu, Xiao-Wen Chang, Doina Precup, Rex Ying, Stan Z. Li, Jian Tang, Guy Wolf, and Stefanie Jegelka. 2024. The Heterophilic Graph Learning Handbook: Benchmarks, Models, Theoretical Analysis, Applications and Challenges. arXiv:2407.09618 [cs.LG] https://arxiv.org/abs/2407.09618

[49] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. 2022. Revisiting Heterophily For Graph Neural Networks. In Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 1362–1375. https://proceedings.neurips.cc/paper_files/paper/2022/file/092359ce5cf60a80e882378944bf1be4-Paper-Conference.pdf

[50] Exl Lukas, J. Mauser Norbert, and Zhang Yong. 2016. Accurate and efficient computation of nonlocal potentials based on Gaussian-sum approximation. J. Comput. Phys. 327 (2016), 629–642. https://doi.org/10.1016/j.jcp.2016.09.045

[51] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. 2023. A Comprehensive Survey on Graph Anomaly Detection With Deep Learning. IEEE Transactions on Knowledge and Data Engineering 35, 12 (2023), 12012–12038. https://doi.org/10.1109/TKDE.2021.3118815

[52] P. Del Moral and A. Niclas. 2018. A Taylor expansion of the square root matrix function. J. Math. Anal. Appl. 465, 1 (2018), 259–266. https://doi.org/10.1016/j.jmaa.2018.05.005

[53] Mark Newman. 2010. Networks: An Introduction. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

[54] Peter Nilsson, Ateeq Ur Rahman Shaik, Rakesh Gangarajaiah, and Erik Hertz. 2014. Hardware implementation of the exponential function using Taylor series. In 2014 NORCHIP. 1–4. https://doi.org/10.1109/NORCHIP.2014.7004740

[55] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. 2018. Graph Signal Processing: Overview, Challenges, and Applications. Proc. IEEE 106, 5 (2018), 808–828. https://doi.org/10.1109/JPROC.2018.2820126

[56] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In International Conference on Learning Representations. https://openreview.net/forum?id=S1e2agrFvS

[57] Chen Peng, Villa Umberto, and Ghattas Omar. 2019. Taylor approximation and variance reduction for PDE-constrained optimal control under uncertainty. J. Comput. Phys. 385 (2019), 163–186. https://doi.org/10.1016/j.jcp.2019.01.047

[58] George M Phillips. 2003. Interpolation and approximation by polynomials. Vol. 14. Springer New York. https://doi.org/10.1007/b97417

[59] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A critical look at the evaluation of GNNs under heterophily: Are we really making progress?. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=tJbbQfw-5wv

[60] Hezhe Qiao, Hanghang Tong, Bo An, Irwin King, Charu Aggarwal, and Guansong Pang. 2024. Deep Graph Anomaly Detection: A Survey and New Perspectives. arXiv:2409.09957 [cs.LG] https://arxiv.org/abs/2409.09957

[61] Aliaksei Sandryhaila and José M. F. Moura. 2013. Discrete Signal Processing on Graphs. IEEE Transactions on Signal Processing 61, 7 (2013), 1644–1656. https://doi.org/10.1109/TSP.2013.2238935

[62] Aliaksei Sandryhaila and José M. F. Moura. 2013. Discrete signal processing on graphs: Graph filters. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. 6163–6166. https://doi.org/10.1109/ICASSP.2013.6638849

[63] A. Sharma and A. K. Varma. 1965. Trigonometric interpolation. Duke Mathematical Journal 32, 2 (1965), 341–357. https://doi.org/10.1215/S0012-7094-65-03235-7

[64] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine 30, 3 (2013), 83–98. https://doi.org/10.1109/MSP.2012.2235192

[65] Gordon K Smyth. 1998. Polynomial approximation. Encyclopedia of Biostatistics 13 (1998).

[66] Yunchong Song, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. 2023. Ordered GNN: Ordering Message Passing to Deal with Heterophily and Over-smoothing. In The Eleventh International Conference on Learning Representations. https://openreview.net/forum?id=wKPmPBHSnT6

[67] Gilbert Strang. 2006. Linear algebra and its applications. Belmont, CA: Thomson, Brooks/Cole.

[68] Jiaqi Sun, Lin Zhang, Guangyi Chen, Peng Xu, Kun Zhang, and Yujiu Yang. 2023. Feature Expansion for Graph Neural Networks. In Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202). PMLR, 33156–33176.

[69] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. 2022. Rethinking Graph Neural Networks for Anomaly Detection. In Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162), Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 21076–21089. https://proceedings.mlr.press/v162/tang22b.html

[70] Qian Tao, Zhen Wang, Wenyuan Yu, Yaliang Li, and Zhewei Wei. 2023. LON-GNN: Spectral GNNs with Learnable Orthonormal Basis. arXiv:2303.13750 [cs.LG]

[71] Gabriel Taubin, Tong Zhang, and Gene Golub. 1996. Optimal surface smoothing as filter design. In ECCV '96. Springer Berlin Heidelberg, 283–292.

[72] Xiyuan Wang and Muhan Zhang. 2022. How Powerful are Spectral Graph Neural Networks. In Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162), Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 23341–23362. https://proceedings.mlr.press/v162/wang22am.html

[73] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[74] Lianghao Xia, Yong Xu, Chao Huang, Peng Dai, and Liefeng Bo. 2021. Graph Meta Network for Multi-Behavior Recommendation. In The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. 757–766. https://doi.org/10.1145/3404835.3462972

[75] Junjie Xu, Enyan Dai, Dongsheng Luo, Xiang Zhang, and Suhang Wang. 2023. Learning Graph Filters for Spectral GNNs via Newton Interpolation. arXiv:2310.10064 [cs.LG]

[76] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. 2021. Optimization of Graph Neural Networks: Implicit Acceleration by Skip Connections and More Depth. In Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139), Marina Meila and Tong Zhang (Eds.). PMLR, 11592–11602. https://proceedings.mlr.press/v139/xu21k.html

[77] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. https://doi.org/10.48550/ARXIV.1603.08861

[78] Pavel Zahradnik and Miroslav Vlcek. 2012. Perfect Decomposition Narrow-Band FIR Filter Banks. IEEE Transactions on Circuits and Systems II: Express Briefs 59, 11 (2012), 805–809. https://doi.org/10.1109/TCSII.2012.2218453

[79] Yuan Zhang, Dong Wang, and Yan Zhang. 2019. Neural IR Meets Graph Embedding: A Ranking Model for Product Search. In The World Wide Web Conference (San Francisco, CA, USA) (WWW '19). Association for Computing Machinery, New York, NY, USA, 2390–2400. https://doi.org/10.1145/3308558.3313468

[80] Ruijie Zhao and David B. Tay. 2023. Minimax design of two-channel critically sampled graph QMF banks. Signal Processing 212 (2023), 109129. https://doi.org/10.1016/j.sigpro.2023.109129

[81] Shuai Zheng, Zhenfeng Zhu, Zhizhe Liu, Youru Li, and Yao Zhao. 2023. Node-Oriented Spectral Filtering for Graph Neural Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 46, 1 (2023), 388–402. https://doi.org/10.1109/TPAMI.2023.3324937

[82] Xin Zheng, Yi Wang, Yixin Liu, Ming Li, Miao Zhang, Di Jin, Philip S. Yu, and Shirui Pan. 2024. Graph Neural Networks for Graphs with Heterophily: A Survey. arXiv:2202.07082 [cs.LG] https://arxiv.org/abs/2202.07082

[83] Yun Zhiwei and Zhang Wei. 2017. Shtukas and the Taylor expansion of L-functions. Annals of Mathematics 186, 3 (2017), 767–911. https://doi.org/10.4007/annals.2017.186.3.2

[84] Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. 2022. Graph Neural Networks: Taxonomy, Advances, and Trends. ACM Transactions on Intelligent Systems and Technology 13, 1, Article 15 (jan 2022), 54 pages. https://doi.org/10.1145/3495161

[85] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In Advances in Neural Information Processing Systems, Vol. 33. Curran Associates, Inc., 7793–7804. https://proceedings.neurips.cc/paper_files/paper/2020/file/58ae23d878a47004366189884c2f8440-Paper.pdf

[86] A. Zygmund and Robert Fefferman. 2003. Trigonometric Series (3 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781316036587

A RELATED WORKS

A.1 Spectral-based graph neural networks

Spectral-based graph neural networks form a unique branch of GNNs designed to process graph-structured data by applying graph filters to execute graph convolution (filtering) operations [3]. The pioneering spectral GNN, SpectralCNN[6], was developed as a generalization of convolutional neural networks for graph data, using principles from spectral graph theory. Subsequent refinements, such as ChebNet[13] and GCN [35], have built upon this foundation.

In recent advancements, the design of spectral GNNs has increasingly focused on incorporating various graph filters, which are central to their functionality. Polynomial approximation has become the prevailing approach for constructing these filters, providing both enhanced performance and operational efficiency. As a result, many contemporary spectral GNNs are predominantly defined by polynomial frameworks. For instance, GPRGNN [10] introduces a monomial-based graph filter, interpreted as a generalized PageRank algorithm. BernNet [13] leverages Bernstein polynomials to create nonnegative graph filters, demonstrating significant effectiveness in real-world applications. JacobiConv [72] unifies different methods by employing Jacobian polynomials. OptBasis [24] improves the design of spectral GNNs by introducing filters with optimal polynomial bases. UniFilter [31] introduces the notion of universal bases, bridging polynomial filters with graph heterophily.

A.2 Node classification with heterophily

In recent years, heterophilic graphs have drawn considerable interest in the field of graph learning. Unlike traditional homophilic graphs, where linked nodes usually share the same label, heterophilic graphs connect nodes with contrasting labels. This unique structure presents significant challenges for graph neural networks (GNNs)[20, 48, 82], which are typically designed for homophilic settings. To address these challenges, a range of GNNs tailored to heterophily have emerged. For example, H2GCN[85] introduces specialized mechanisms for embedding nodes in heterophilic environments, OrderGNN [66] restructures message-passing to account for heterophily, and LRGNN [41] leverages a global label relationship matrix to improve performance under heterophily.

Addressing heterophily with spectral GNNs. In most recent, spectral-based GNNs have shown promise in addressing these challenges by learning dataset-specific filters that extend beyond the standard low-pass filters used in conventional GNNs. By doing so, spectral GNNs demonstrate improved performance in tackling heterophilic graphs, achieving superior results in node classification under heterophily [10, 24–26, 31, 39, 72].

A.3 Graph anomaly detection

Graph-based anomaly detection (GAD) is a specialized task within anomaly detection, aimed at identifying anomalies within graph-structured data [51, 60]. The primary goal in GAD is to detect anomalous nodes (outliers) in the graph by leveraging a limited set of labeled samples, including both anomalous and normal nodes. Effectively, GAD can be viewed as a binary node classification task, where the classes represent anomaly and normalcy. The recent success of Graph Neural Networks (GNNs) in node classification has spurred the development of GAD-specialized GNN methods, such as CARE-GNN [14], PC-GNN [46], and GDN [18], with each significantly enhancing detection performance.

Spectral GNNs in GAD. Building on the success of GNN-based approaches for graph anomaly detection (GAD), recent studies leveraging spectral GNNs have yielded promising results. By framing GAD through graph spectrum analysis, these methods introduce novel perspectives on the problem. For example, BWGNN [69] utilizes beta graph wavelets for signal filtering, effectively addressing the "right-shift" phenomenon in GAD. Similarly, GHRN [17] enhances GAD by pruning inter-class edges, focusing on high-frequency graph components to improve detection performance.

B PROOF OF LEMMA 3.4

Proof. We establish this inequality by proving its right-hand and left-hand sides independently.

Proof of right hand side. For convenience, we define the L_2 norm of a function g over the interval $[0, 2]$, denoted by $\|g\|_2$, as follows:

$$\|g\|_2 \triangleq \left(\int_0^2 ||g(x)||^2 \, dx\right)^{\frac{1}{2}} \tag{17}$$

Using the norm expression defined earlier, and recalling the expression for ϵ from Eq. 6, we can derive the following inequalities by applying the Cauchy-Schwarz inequality:

$$\begin{aligned}
\epsilon &= \|f(x) - \mathbf{T}_{0:D}(x; f)\|_2^2 \,, \\
&= \|\sum_{s=1}^{n} f_s(x) - \sum_{s=1}^{n} \mathbf{T}_{0:D}(x; f_s)\|_2^2 \,, &(18)\\
&\leq \left(\sum_{s=1}^{n} \|f_s(x) - \mathbf{T}_{0:D}(x; f_s)\|_2\right)^2 = \left(\sum_{s=1}^{n} \sqrt{\epsilon_s}\right)^2 \,, &(19)
\end{aligned}$$

which is our right-hand side.

Proof of Left-hand side. To proceed without loss of generality, we consider f to be nonnegative over the entire interval. Recalling the definition of ϵ from Eq. 6, it follows that

$$\begin{aligned}
\epsilon &= \|\mathbf{T}_{0:D}(x; f) - f(x)\|_2^2 \,, \\
&= \|\sum_{s=1}^{n} f_s(x) - \sum_{s=1}^{n} \mathbf{T}_{0:D}(x; f_s)\|_2^2 \,, &(20)\\
&= 2 \sum_{1 \leq p \leq q \leq n} \left\| \sqrt{det \begin{vmatrix} f_p(x) & f_q(x) \\ \mathbf{T}_{0:D}(x; f_q) & \mathbf{T}_{0:D}(x; f_p) \end{vmatrix}} \right\|_2^2 \\
&\quad + \sum_{s=1}^{n} \sqrt{\epsilon_s}^2 \,, &(21)\\
&\geq 0 + \sum_{s=1}^{n} \sqrt{\epsilon_s}^2 = \sum_{s=1}^{n} \epsilon_s \,, &(22)
\end{aligned}$$

which is our left-hand side.

Thus, combining the two parts of the proof above, we confirm that Lemma 3.4 holds for the continuous form of error. □

Adaptation to the disctrete error form. This is due to the applicability of the Cauchy-Schwarz inequality to the discrete version of norm inequalities, ensuring that the right-hand side holds for

the discrete form of the error. The left-hand side, which is based solely on fundamental non-negative relations, also maintains the inequality in the discrete setting. Consequently, Lemma 3.4 can be directly adapted to the discrete scenario.

C PROOF OF THEOREM 3.5

Proof. We establish this inequality by proving its right-hand and left-hand sides independently.

Proof of right hand side. Recalling the expression of ξ from Eq. 8, we can derive the following inequality using the submultiplicative property of Frobenius norm:

$$\begin{aligned}\xi =&\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{W}\|_F\\ \leq&\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\|_F\cdot\|\boldsymbol{X}\|_F\cdot\|\boldsymbol{W}\|_F\,, \quad (23)\\ \leq&r\cdot\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\|_F\cdot\|\boldsymbol{X}\|_F\,. \quad (24)\end{aligned}$$

Note that $\boldsymbol{U}$ is orthogonal matrix, which will not influence the Frobenius norm of any matrices in product operation. Thus, the inequality above can further be derived as:

$$\begin{aligned}\xi \leq&r\cdot\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\|_F\cdot\|\boldsymbol{X}\|_F\\ =&r\cdot\|diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\|_F\cdot\|\boldsymbol{X}\|_F\,, \quad (25)\\ =&r\cdot\epsilon\cdot\|\boldsymbol{X}\|_F\,, \quad (26)\\ \leq&r\|\boldsymbol{X}\|_F(\sum_{s=1}^{n}\sqrt{\epsilon_s})^2\,, \quad (27)\end{aligned}$$

which is the right-hand side.

Proof of left hand side. Using the basic of the matrix perturbation theory, we can derive the following inequality:

$$\begin{aligned}\xi =&\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{W}\|_F\\ \geq&\delta_{\boldsymbol{X}}\delta_{\boldsymbol{W}}\|\boldsymbol{U}diag(\mathbf{T}_{0:D}(\boldsymbol{\lambda};f)-f(\boldsymbol{\lambda}))\boldsymbol{U}^T\|_F\,, \quad (28)\\ \geq&\delta_{\boldsymbol{X}}\delta_{\boldsymbol{W}}\sum_{s=1}^{n}\epsilon_s\,, \quad (29)\end{aligned}$$

which is the left-hand side.

Thus, combining the two parts of the proof above, we confirm that Theorem 3.5 holds. □

D PROOF OF THEOREM 4.1

Proof. To begin with, note that α_k and β_k can be computed as follows:

$$\begin{aligned}\alpha_k &= \frac{\omega}{\pi}\int_0^{\frac{2\pi}{\omega}} f(x)\sin(k\omega x)\,dx\,,\\ \beta_k &= \frac{\omega}{\pi}\int_0^{\frac{2\pi}{\omega}} f(x)\cos(k\omega x)\,dx\,. \quad (30)\end{aligned}$$

We alternatively consider such a *real function* $f'(x)$ which is defined as follows:

$$f'(x)=\begin{cases}f(x)\,, & x\in\left[0,\frac{2\pi}{\omega}\right]\,;\\ 0\,, & others\,.\end{cases} \quad (31)$$

Notice that such $f'(x)$ satisfies the *Dirichlet conditions* [28], the *Fourier transform* of $f'(x)$ exists according to [21] and is defined as follows:

$$\begin{aligned}F'(\Omega) &= \int_{-\infty}^{+\infty} f'(x)e^{-i\Omega x}\,dx\,,\\ &= \int_{-\infty}^{+\infty} f'(x)[\cos(\Omega x)-i\cdot\sin(\Omega x)]\,dx\,, \quad (32)\end{aligned}$$

where Ω denotes the variable in frequency domain, and i is the imaginary unit.

Furthermore, the $f'\in L^1(\mathbb{R}^n)$ is an integrable function, making $f'(x)$ satisfy the *Riemann–Lebesgue lemma* which is defined as follows:

Theorem D.1. ***(Riemann–Lebesgue lemma [4])***. *Let $g\in L^1(\mathbb{R}^n)$ be an integrable function, and let G be the Fourier transform of g. Then the G vanishes at infinity, which is defined as follows:*

$$\lim_{|\Omega|\to\infty}|G(\Omega)|=0\,. \quad (33)$$

Thus, the limit of $F'(\Omega)$ as Ω approaches $+\infty$ equals 0, which is defined as

$$\begin{aligned}\lim_{\Omega\to+\infty}F'(\Omega) &= \underbrace{\lim_{\Omega\to+\infty}\left[\int_{-\infty}^{+\infty}f'(x)[\cos(\Omega x)-i\cdot\sin(\Omega x)]\,dx\right]}_{\text{Formula 1}}\,,\\ &= 0\,. \quad (34)\end{aligned}$$

Since the $f'(x)$ is a real function, the Formula 1 equals 0 if and only if the following equations hold:

$$\begin{aligned}\lim_{\Omega\to+\infty}\int_{-\infty}^{+\infty}f'(x)\cos(\Omega x)\,dx=0\,,\\ \lim_{\Omega\to+\infty}\int_{-\infty}^{+\infty}f'(x)\sin(\Omega x)\,dx=0\,. \quad (35)\end{aligned}$$

With combing the definition of $f'(x)$ in Eq. 31, the Eq. 35 above can be further derived as follows:

$$\begin{aligned}\lim_{\Omega\to+\infty}\int_{0}^{\frac{2\pi}{\omega}}f(x)\cos(\Omega x)\,dx=0\,,\\ \lim_{\Omega\to+\infty}\int_{0}^{\frac{2\pi}{\omega}}f(x)\sin(\Omega x)\,dx=0\,. \quad (36)\end{aligned}$$

Finally, with replacing the Ω, $\Omega\to+\infty$ with $k\omega$, $k\to+\infty$ in Eq. 36, and further considering the Eq. 30, we obtain the following equations:

$$\begin{aligned}\lim_{k\to+\infty}\alpha_k &= \frac{\omega}{\pi}\cdot\lim_{k\to+\infty}\int_0^{\frac{2\pi}{\omega}}f(x)\sin(k\omega x)\,dx=0\,,\\ \lim_{k\to+\infty}\beta_k &= \frac{\omega}{\pi}\cdot\lim_{k\to+\infty}\int_0^{\frac{2\pi}{\omega}}f(x)\cos(k\omega x)\,dx=0\,, \quad (37)\end{aligned}$$

and the proof is completed. □

E EXPERIMENTAL DETAILS

This section outlines the extensive experimental settings relevant to the studies conducted in Section 3.2 and Section 5. Experiments are conducted using an NVIDIA Tesla V100 GPU with 32GB of memory, running on Ubuntu 20.04 OS and CUDA version 11.8.

E.1 Experimental details of numerical validation

Descriptions to target functions. The six target functions employed in the numerical experiments are defined by the expressions presented in Table 8.

Random graph construction. We construct random graphs using the *Erdős-Rényi* model, specifically denoted as $G(n, p)$ [53]. In our experiments, we set the number of nodes n to 50,000 and the edge creation probability p to 0.5.

Random node feature construction. We construct random node features $\boldsymbol{X}$ drawn from a Gaussian distribution. Each entry in the feature matrix $\boldsymbol{X}$ is independently sampled and follows a standard normal distribution, $N(0, 1)$.

Experimental settings. To start, we randomly generate ten pairs of graphs and features, denoted as $(\mathcal{G}_1, \boldsymbol{X}_1)$, $(\mathcal{G}_2, \boldsymbol{X}_2)$, ..., $(\mathcal{G}_{10}, \boldsymbol{X}_{10})$. For each pair $(\mathcal{G}_j, \boldsymbol{X}_j)$, we apply six different graph filters, resulting in six filtered outputs: $\boldsymbol{Y}_{1j}, \boldsymbol{Y}_{2j}$, ..., $\boldsymbol{Y}_{6j}$. This process involves performing graph convolution on $\boldsymbol{X}$ using these target functions.

As discussed in Section 3.2, we pursue two tasks: the first task involves approximating function slices, while the second focuses on filter learning.

- **Approximation of function slices.** We generate 50000 function slices for each target function based on the eigenvalues of the graph $\mathcal{G}_j$. Various polynomial bases are employed for the approximation, with the minimum sum of squared errors (SSE) serving as the evaluation metric. The final results are averaged over ten randomly generated graphs.
- **Graph filter learning.** We implement a one-layer linear spectral Graph Neural Network (GNN) that operates without a weight matrix $\boldsymbol{W}$, utilizing various polynomial bases. The input consists of pairs $(\mathcal{G}_j, \boldsymbol{X}_j)$to approximate the target output $\boldsymbol{Y}$. The learned graph filters are then employed to compute the discrepancies with the target filters, using the Frobenius norm as the evaluation metric. Finally, the results are averaged across ten randomly generated graphs to ensure robustness.

E.2 Experimental details for node classification

Dataset statistics. The statistics of the 13 datasets used in Section 5.2 are provided in Tables 9 and 10.

Baseline implementations. We provide code URLs to the public implementations for all baselines referenced in this paper. In particular, for the well-established baselines GCN and ChebNet, we employ standardized implementations based on previous research [24–26, 30, 39, 68, 72, 81]; for the remaining baselines, we resort to the publicly released code, accessible via the provided URLs as below.

- H2GCN: https://github.com/GemsLab/H2GCN
- GloGNN: https://github.com/RecklessRonan/GloGNN
- LINKX: https://github.com/CUAI/Non-Homophily-Large-Scale
- OrderGNN: https://github.com/lumia-group/orderedgnn
- LRGNN: https://github.com/Jinx-byebye/LRGNN
- GCN: https://github.com/ivam-he/ChebNetII
- SGC: https://github.com/ivam-he/ChebNetII
- GCNII: https://github.com/chennnM/GCNII
- ChebNet: https://github.com/ivam-he/ChebNetII
- ACMGCN: https://github.com/SitaoLuan/ACM-GNN
- Specformer: https://github.com/DSL-Lab/Specformer
- GPRGNN: https://github.com/jianhao2016/GPRGNN
- BernNet: https://github.com/ivam-he/BernNet
- ChebNetII: https://github.com/ivam-he/ChebNetII
- OptBasis: https://github.com/yuziGuo/FarOptBasis
- NFGNN: https://github.com/SsGood/NFGNN
- JacobiConv: https://github.com/GraphPKU/JacobiConv
- AdaptKry: https://github.com/kkhuang81/AdaptKry
- UniFilter: https://github.com/kkhuang81/UniFilter

Implementation of TFGNN. As introduced in Section 4.3, TFGNN is implemented in two distinct configurations to accommodate graphs of varying sizes. For graphs detailed in Table 3, we utilize the architecture represented by Eq. (15). For larger graphs listed in Table 5, we employ the architecture shown in Eq. (16).

The MLP architecture within TFGNN is dataset-specific. For medium-sized graphs (Cora, Citeseer, Pubmed, Roman-empire, Amazon-ratings, and Questions), we use a two-layer MLP with 64 hidden units. In contrast, larger datasets are assigned three-layer MLPs with varying hidden units: 128 for Gamers and Genius, 256 for Snap-patents and Pokec, 512 for Ogbn-arxiv, and 1024 for Ogbn-papers100M.

To ensure experimental fairness, we fix the order of the Trigonometric Polynomial Decomposition (TPD), denoted as D, to 10, aligning with other baselines such as GPRGNN and ChebNetII. We employ a grid search to optimize the weight decay over $\{5e-1, 5e-2, 5e-3, 5e-4, 0\}$, learning rate over $\{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$, dropout over $\{0, 0.2, 0.5, 0.7, 0.9\}$, ω within $\{0.2\pi, 0.3\pi, 0.5\pi, 0.7\pi\}$, and K from $\{2, 4, 6, 8, 10, 15, 20\}$.

Model training and testing. We follow the dataset splitting protocols established in the literature. For the Cora, Citeseer, and Pubmed datasets, we utilize the established 60%/20%/20% train/val/test split, which has been widely adopted across numerous studies [24–26, 30, 31, 38, 39, 68, 72]. For the Roman-empire, Amazon-ratings, and Questions datasets, we implement a 50%/25%/25% train/val/test split, aligning with the protocols outlined in their original publications [59]. This 50%/25%/25% train/val/test split strategy is also applied to the Gamers, Genius, Snap-patents, and Pokec datasets, as recommended in [44]. Finally, for Ogbn-arxiv, Ogbn-products, and Ogbn-papers100M, we adopt the fixed splits defined in the original OGB dataset paper [29].

Models are trained for a maximum of 1,000 epochs, with early stopping implemented after 200 epochs if there is no improvement in validation accuracy. To handle exceptionally large graphs, we employ a mini-batch training strategy using batches of 20,000 nodes. The optimization process employs the Adam optimizer [34]. For each dataset, we generate 10 random node splits and perform 10

Table 8: Mathematical expressions of six target functions.

Functions	Expressions
$f_1(x)$	$e^{-20(x-0.5)^2} + e^{-20(x-1.5)^2}$
$f_2(x)$	$\begin{cases} e^{-100(x-0.8)^2} + e^{-100(x-1.2)^2} + 0.5 \cdot (1+\cos(2\pi x)), & x \in [0, 0.5] \\ e^{-100(x-0.8)^2} + e^{-100(x-1.2)^2}, & x \in (0.5, 1.5) \\ e^{-100(x-0.8)^2} + e^{-100(x-1.2)^2} + 0.5 \cdot (1+\cos(2\pi x)), & x \in [1.5, 2] \end{cases}$
$f_3(x)$	$e^{-100(x-0.5)^2} + e^{-100(x-1.5)^2} + 1.5e^{-50(x-1)^2}$
$f_4(x)$	$e^{-100x^2} + e^{-100(x-2)^2}$
$f_5(x)$	$1 - e^{-10x^2}$
$f_6(x)$	$e^{-10(x-0.4)^2} + 2e^{-10(x-1.5)^2}$

Table 9: Statistics for medium-to-large datasets, with # Edge homo indicating the edge homophily measure from [85].

	Cora	CiteSeer	PubMed	Ogbn-arxiv	Roman-empire	Amazon-ratings	Questions	Gamers	Genius
# Nodes	2708	3327	19,717	169,343	22,662	24,492	48,921	168,114	**421,961**
# Edges	5278	4552	44,324	1,157,799	32,927	93,050	153,540	**6,797,557**	922,868
# Features	1433	3703	500	128	300	300	301	7	12
# Classes	7	6	5	40	18	5	2	2	2
# Edge homo [85]	0.81	0.74	0.80	0.65	0.05	0.38	0.84	0.55	0.62

Table 10: Statistics for exceptionally large datasets. # Edge homo for Ogbn-papers100M is unavailable due to runtime exceedance.

	Ogbn-products	Ogbn-papers100M	Snap-patents	Pokec
# Nodes	2,449,029	**111,059,956**	2,923,922	1,632,803
# Edges	61,859,140	**1,615,685,872**	13,975,788	30,622,564
# Features	100	128	269	65
# Classes	47	172	5	2
# Edge homo [85]	0.81	-	0.07	0.45

random initializations for each baseline on these splits. This process yields a total of 100 evaluations for each dataset. The reported results for each baseline represent the average of these 100 evaluations.

E.3 Experimental details for graph anomaly detection

Dataset statistics. Table 11 presents the statistics of datasets used in Section 5.3.

Table 11: Statistics of datasets utilized for graph anomaly detection. # Anomaly represents the rate of abnormal nodes.

	YelpChi	Amazon	T-Finance
# Nodes	45,954	11,944	39,357
# Edges	3,846,979	4,398,392	21,222,543
# Features	32	25	10
# Anomaly	14.53%	6.87%	4.58%

Baseline implementations. We provide code URLs to the official implementations of all baseline models referenced in this paper. Specifically, for general-purpose spectral GNNs like GPRGNN, OptBasis, AdaptKry, and NFGNN, which are initially introduced as uniform, decoupled GNN architectures, we implement them in alignment with the TFGNN variant defined in Eq. (15). Each model uses a fixed maximum polynomial degree of 10 and a two-layer MLP with 64 hidden units for feature transformation, consistent with BWGNN [69]. The GCN baseline is similarly implemented with a two-layer setup featuring 64 hidden dimensions. For other baselines, we rely on their official implementations (links provided below). All models are rebuilt and evaluated in PyG [15] framework to maintain experimental fairness.

- PC-GNN: https://github.com/PonderLY/PC-GNN
- CARE-GNN: https://github.com/YingtongDou/CARE-GNN
- GDN: https://github.com/blacksingular/wsdm_GDN
- BWGNN: https://github.com/squareRoot3/Rethinking-Anomaly-Detection
- GHRN: https://github.com/blacksingular/GHRN
- GPRGNN: https://github.com/jianhao2016/GPRGNN
- OptBasis: https://github.com/yuziGuo/FarOptBasis
- AdaptKry: https://github.com/kkhuang81/AdaptKry
- NFGNN: https://github.com/SsGood/NFGNN

Implementation of TFGNN. In pursuit of fairness, TFGNN incorporates a decoupled architecture consistent with general-purpose spectral GNNs, featuring a maximum polynomial degree of 10 and a two-layer MLP comprising 64 hidden units for feature transformation. This approach also ensures that parameter fairness is in

alignment with BWGNN. For hyperparameter tuning, we adhere to the previous setups detailed in Appendix E.2.

Training and testing. Following the training protocol established in the BWGNN paper [69], we maintain a validation-to-test set split of $1:2$, and employ training ratios of 1% (across all datasets) and 40% (additionally for T-Finance). Baselines are trained for 100 epochs using the Adam optimizer, without early stopping. We report the test results of the models that achieved the highest Macro-F1 score on the validation set, averaging results across 10 random seeds to ensure robustness.

F ADDITIONAL RESULTS

In this section, we present additional results that bolster the experiments detailed in the main text, further substantiating our conclusions.

F.1 Full numerical experiment results

We present a detailed overview of our numerical experiment results in Table 12, including those for our TFGNN.

The data illustrates that both the trigonometric polynomial and TFGNN achieve outstanding performance, underscoring the advantages of our approach. Additionally, these results are consistent with the node classification outcomes outlined in Section 5.2, validating the real-world applicability of our analysis.

F.2 Additional ablation studies of K and ω

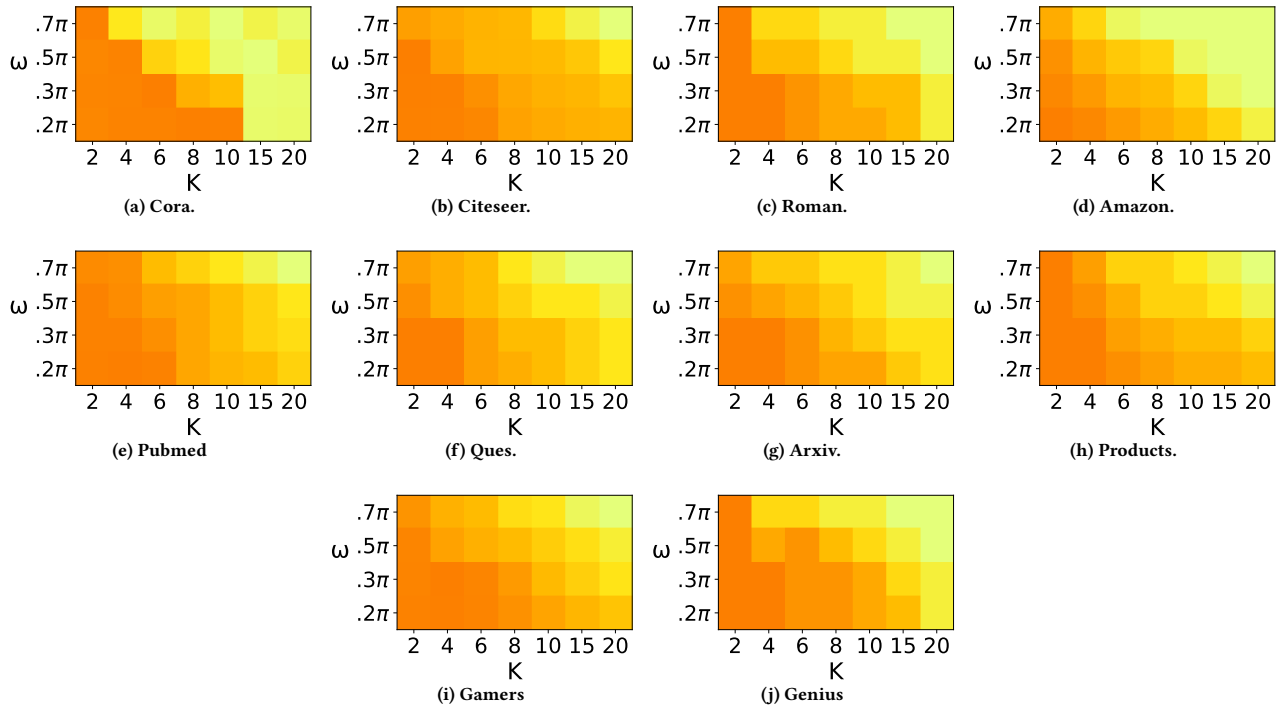
In this section, we present an extended ablation study of the key hyperparameters K and ω, complementing our findings in Section 5.2.3, with Figure 4 illustrating the outcomes.

The figures indicate a trend similar to that highlighted in Section 5.2.3, showing that the best-performing values for K, ω, and the product $K \cdot \omega$ typically lie within low ranges.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Table 12: Full numerical experiment results. introduced in Section 3.2. Both trigonometric polynomial and TFGNN are included for comprehensive evaluations.

Method		Slice-wise approximation						Filter Learning						# Avg	# Avg
Polynomial	GNN	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	Rank 1	Rank 2
Monomial	GPRGNN [10]	139.9	289.1	466.1	398.3	1.83	97.83	167.2	366.4	566.3	468.7	15.91	139.2	6	6
Bernstein	BernNet [25]	32.78	247.3	398.5	306.5	0.058	22.92	68.23	313.2	448.2	415.2	7.79	95.84	5	5
Chebyshev	ChebNetII [26]	23.45	85.19	244.8	187.2	0.018	13.13	64.22	168.4	402.5	347.5	6.83	86.25	4	4
Jacobian	JacobiConv [72]	22.18	80.77	239.2	155.3	0.017	11.82	48.56	95.92	338.1	266.4	5.33	65.13	3	3
Learnable	OptBasis [24]	20.75	80.53	225.7	152.7	0.017	11.20	43.44	89.48	289.5	238.1	4.98	61.70	2	2
Trigonometric	TFGNN	**12.35**	**23.69**	**71.13**	**59.88**	**0.017**	**6.52**	**27.23**	**65.19**	**102.3**	**105.3**	**4.05**	**48.08**	**1**	**1**



(a) Cora. (b) Citeseer. (c) Roman. (d) Amazon.

(e) Pubmed (f) Ques. (g) Arxiv. (h) Products.

(i) Gamers (j) Genius

Figure 4: Additional ablation studies on K and ω. Darker shades indicate higher performance values.