A Piece-wise Polynomial Filtering Approach for Graph Neural Networks

✉Vijay Lingam*, Manan Sharma*, Chanakya Ekbote*, Rahul Ragesh, Arun Iyer, and Sundararajan Sellamanickam

Microsoft Research India
{vijaylingam0810,manan2908}@gmail.com, {t-cekbote, rahulragesh,ariy,ssrajan}@microsoft.com

Abstract. Graph Neural Networks (GNNs) exploit signals from node features and the input graph topology to improve node classification task performance. Recently proposed GNNs work across a variety of homophilic and heterophilic graphs. Among these, models relying on polynomial graph filters have shown promise. We observe that polynomial filter models need to learn a reasonably high degree polynomials without facing any over-smoothing effects. We find that existing methods, due to their designs, either have limited efficacy or can be enhanced further. We present a spectral method to learn a bank of filters using a piece-wise polynomial approach, where each filter acts on a different subsets of the eigen spectrum. The approach requires eigendecomposition only for a few eigenvalues at extremes (i.e., low and high ends of the spectrum) and offers flexibility to learn sharper and complex shaped frequency responses with low-degree polynomials. We theoretically and empirically show that our proposed model learns a better filter, thereby improving classification accuracy. Our model achieves performance gains of up to $\sim$6% over the state-of-the-art (SOTA) models while being only $\sim$2x slower than the recent spectral approaches on graphs of sizes up to $\sim$169K nodes.

Keywords: Graph Neural Networks · Representation Learning · Polynomial Filtering.

1 Introduction

We are interested in the problem of classifying nodes in a graph where a graph with features for all nodes, and labels for a few nodes are made available for learning. Inference is done using the learned model for the remaining nodes (*aka* transductive setting). Graph Neural Networks (GNNs) perform well on such problems [1]. Most GNNs predict a node's label by aggregating information from its neighbours in a certain way, making them dependent on some correlation between the structure and the node labels[1]. For example, in the simplest case, GNNs work well when the node and its

* Equal contribution. Work done while author was at Microsoft Research India

[1] Characterizing the correlation between the graph structure and node features/labels is an active area of research. Several metrics have been proposed including edge homophily [13,5], node homophily [4], class homophily [27]. All these metrics show that standard GNNs perform well when the graphs and node labels are positively correlated.

neighbours share similar labels. However, the performance can be poor if this criterion is not satisfied. Recently, several modeling approaches have been proposed to build/learn robust GNN models. Some modify the aggregation mechanism [4,5,3], while others propose to estimate and leverage the label-label compatibility matrix as a prior [6].

More recent approaches have tackled this problem from a graph filter learning perspective [7,8,38,37,32,39]. With eigenvalues having frequency interpretations [26], one or more filters (i.e., a bank of filters) that selectively accentuates and suppresses various spectral components of graph signals are learned using task-specific available information. The filtering operation enables learning better node representation which translates to improved classification accuracy.

Designing effective graph filters is a challenging problem, and most recent methods [10,8,38,37] suggest interesting ways to learn polynomial filters having finite impulse response (FIR) characteristics. These models are efficient and attractive, as they make use of local neighborhood (i.e., using sparse adjacency matrix repeatedly) and do not require to pre-compute eigendecomposition, which is expensive (when done over the entire spectrum, i.e., for all eigenpairs). Though these models are able to learn better filters and give good performance gains, they are still unable to learn richer and complex frequency responses, which require higher-order polynomials. One key reason for their inability to learn effective high-order polynomials is that they only *mitigate* the over-smoothing problem. This aspect of the problem becomes clear when we analyze a general class of FIR filters (GFIR) and find that the over-smoothing problem exists for the whole class, of which simplified GCN [16], GPR-GNN [8] and several other models are special cases. We also find that while constraining the model space of GFIR (e.g., [8]) helps to mitigate over-smoothing, it is still unable to learn complex-shaped and sharper frequency responses. Considering this background, our interest lies in learning a bank of effective filters in spectral domain to model complex shaped frequency responses, as needed for graphs with diverse label correlations. Our contributions are:

1. We propose a novel piece-wise polynomial filtering approach to learn a filter bank tuned for the task at hand. Since full eigendecomposition is expensive, we present an efficient method that makes use of only a few extremal eigenpairs and leverages GPR-GNN to learn multiple filters. (While computing the extremal eigenpairs does lead to an increased computational cost, we show in A.7 that such a cost is indeed managable, i.e. the model is only $\sim$2x slower than recent spectral SOTA methods.)
2. We analyze, theoretically and experimentally, the shortcomings of a general class of FIR (GFIR) filters. We show that the proposed piece-wise polynomial GNN (PP-GNN) solution is more expressive and is capable of modeling richer and complex frequency responses.
3. We conduct a comprehensive experimental study to compare PP-GNN with a wide range of methods ($\sim$20), covering both spatial and spectral convolution based methods on nearly a dozen datasets. Experimental results show that PP-GNN performs significantly better, achieving up to $\sim$6% gains on several datasets.

2 Related Work

Graph Neural Networks (GNNs) have become increasingly popular models for semi-supervised classification with graphs. [11] set the stage for early GNN models, which was

then followed by various modifications [12,1,9,2] and improvements along with several different directions such as improved aggregation and attention mechanisms [9,2,3], efficient implementation of spectral convolution [12,16], incorporating random walk information [15,13,14], addressing over-smoothing [15,10,13,14,28,29,30], etc.

Another line of research explored the question of where GNNs help. The key understanding is that the performance of GNN is dependent on the correlation of the graphs with the node labels. Several approaches [13,5,31] considered edge homophily and proposed a robust GNN model by aggregating information from several higher-order hops. [3] also considered edge homophily and mitigated the issue by learning robust attention models. [4] talks about node homophily and proposes to aggregate information from neighbours in the graph and neighbours inferred from the latent space. [6] proposes to estimate label-label compatibility matrix and uses it as a prior to update posterior belief on the labels.

Recent approaches motivated by the developments in graph signal processing [25], focus on learning graph filters with filter functions that operate on the eigenvalues of the graph directly or indirectly, adapting the frequency response of graph filters for the desired task. [7] models the filter function as an attention mechanism on the edges, which learns the difference in the proportion of low-pass and high-pass frequency signals. [8] proposes a polynomial filter on the eigenvalues that directly adapts the graph for the desired task. [32] decompose the graph into low-pass and high-pass frequencies, and define a framelet based convolutional model. [38] propose to learn graph filters using Bernstein approximation of arbitrary filtering function. [37] suggest to learn adaptive graph filters for different feature channels and frequencies by stacking multiple layers. Our work is closely related to these lines of exploration. All these works still need high-degree polynomials when sharper frequency responses are needed; however, though improved performance is observed and over-smoothing is mitigated, further improvements seem possible. Another class of Infinite Impulse Response (IIR) filters have been proposed to learn complex filter responses. ARMA [39] achieves this by using auto-regressive moving average, but empirically have been found to have limited effectiveness. Implementing precise ARMA filters for graphs is a challenging problem and has high computation costs. [39] proposes several approximations to mitigate the issues, but these come with limited efficacy. In our work, we propose to learn a filter function as a sum of polynomials over different subsets of the eigenvalues (in essence, a bank of filters) by operating directly in the spectral domain, enabling design of effective filters to model task-specific complex frequency responses with compute trade-offs.

3 Problem Setup and Motivation

We focus on the problem of semi-supervised node classification on a simple graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. Let $\mathbf{A} \in \{0,1\}^{n \times n}$ be the adjacency matrix associated with $\mathcal{G}$, where $n = |\mathcal{V}|$ is the number of nodes. Let $\mathcal{Y}$ be the set of all possible class labels. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the d-dimensional feature matrix for all the nodes in the graph. Given a training set of nodes $\mathcal{D} \subset \mathcal{V}$ whose labels are known, along with $\mathbf{A}$ and $\mathbf{X}$, our goal is to predict the labels of the remaining nodes. Let $\mathbf{A_I} = \mathbf{A} + \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. Let $\mathbf{D_{A_I}}$ be the degree matrix of $\mathbf{A_I}$

and $\widetilde{\mathbf{A}} = \mathbf{D}_{\mathbf{A_I}}^{-1/2}\mathbf{A_I}\mathbf{D}_{\mathbf{A_I}}^{-1/2}$. Let $\widetilde{\mathbf{A}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ be the eigendecomposition. The spectral convolution of $\mathbf{X}$ on the graph $\mathbf{A}$ can be defined via the reference operator $\widetilde{\mathbf{A}}$ and a general Finite Impulse Response (FIR) filter ([40]), parameterized by $\mathbf{\Theta}$ as:

$$\mathbf{Z} = \sum_{j=1}^{k} \widetilde{\mathbf{A}}^j \mathbf{X}\mathbf{\Theta}_j \tag{1}$$

The term, $\widetilde{\mathbf{A}}^j\mathbf{X}$ uniformly converges to a stationary value as the value of j increases, making the node features indistinguishable (often referred to as the problem of over-smoothing), thereby reducing the importance of the corresponding term for the task at hand. We formalize the argument via commenting on the Dirichlet energy of the higher-order terms [41]. Dirichlet energy reveals the embedding smoothness with the weighted node pair distance. A smaller value is highly related to over-smoothing [42]. Under some conditions, the upper bound of Dirichlet energy of higher terms is theoretically proved to converge to 0 in the limit of infinite layers. In other words, all nodes converge to a trivial fixed point in the embedding space and hence do not contribute to the discriminative signals. This is formalized as follows:

Proposition 3.1: The upper bound of Dirichlet energy for the higher-order terms in the general FIR model exponentially decreases to 0 with the order, k. Formally, with $\mathbf{S}$ as any graph shift operator (in our case, the normalized adjacency), and $\mathbf{\Theta}_k$ be the set of parameters, indexed by k:

$$E(\mathbf{S}^k\mathbf{X}\mathbf{\Theta}_k) \leq (1-\lambda)^{2k} s_{\Theta_k} E(\mathbf{X})$$

where, λ is the positive eigenvalue of the graph Laplacian Δ that is closest to 0; s_{Θ_k} is the largest singular value of $\mathbf{\Theta}_k$. We relegate the proof of the corollary as well as the formal definition of a few terms in section A.3 of the supplementary material.

The family of general FIR filters is ubiquitous and gives rise to various other filter families (eg. polynomial) simply by placing constraint on the form of parameterization. We experiment with placing simple constraints on the bare GFIR model in section A.5 of supplementary and observe that while constraining helps improving the performance, it doesn't helps in learning complex responses. It is not difficult to see that the models of [12],[8], etc. are just instantiations of the GFIR family. Particularly, by restricting $\mathbf{\Theta}_j = \alpha_j\mathbf{I}$, we recover the linear model (without MLP) of [8], which can now be interpreted as the polynomial filter function h operating on the eigenvalues, in the Fourier domain [25,8] as,

$$\mathbf{Z} = \sum_{j=1}^{k} \alpha_j \widetilde{\mathbf{A}}^j \mathbf{X} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^T\mathbf{X} \tag{2}$$

with $h : \mathbb{R} \to \mathbb{R}$ is defined as $h(\lambda;\ \boldsymbol{\alpha}) = \sum_{i=1}^{k} \alpha_i \lambda^i$ where α_i's are coefficients of the polynomial, k is the order of the polynomial and λ is any eigenvalue from $\mathbf{\Lambda}$. $h()$ is applied element-wise across $\mathbf{\Lambda}$ in Eq.2. In this process, the filter function is essentially adapting the graph for the desired task at hand.

It is well-known that polynomial filters can approximate any graph filter [26,25]. Since polynomial filters are a class of the GFIR filter family, they inherit the same problem of over-smoothing as the order of the polynomial becomes higher. [8] show that they

achieve the diminishing of the contribution of higher-order terms by showing that their coefficients converge to zero during training. While this mitigates the over-smoothing problem, use of lower-order polynomials results in an imprecise approximation when the dataset requires a complex spectral filter for obtaining a superior performance, which we will show is the case for certain datasets (See Figure 1 and supplementary's A.4). Empirical results demonstrating the key points discussed in this section: a) smoothening of the higher-order terms (can be found in Figure 5a of the supplementary material) and b) their effect on the test performance on a few datasets (can be found in Figure 5b and 5c of supplementary material). These problems indicate the need for a method that can approximate arbitrarily complex filters better and at the same time mitigate the effects of over-smoothing.

4 Proposed Approach

We propose to learn a bank of polynomial filters with each filter operating on different parts of the spectrum, taking task-specific requirements into account. We show that our proposed filter design can approximate the latent optimal graph filter better than a single polynomial, and the resultant class of learnable filters is richer.

4.1 Piece-wise Polynomial Filters

We start with the expression (2) for node embedding rewritten with an MLP network transforming input features, $\mathbf{X}$, :

$$\mathbf{Z} = \sum_{i=1}^{n} h(\lambda_i)\mathbf{u}_i\mathbf{u}_i^T\mathbf{Z}_x(\mathbf{X};\mathbf{\Theta}) \tag{3}$$

where $\mathbf{u}_i$ is the eigenvector corresponding to the eigenvalue, λ_i, and $\mathbf{Z}_x(\mathbf{X};\mathbf{\Theta})$ is an MLP network with parameters $\mathbf{\Theta}$. Our goal is to learn a filtering function, $h(\lambda)$ jointly with MLP network, using which we compute the node embedding, $\mathbf{Z}$. We model $h(\lambda)$ as a piece-wise polynomial or spline function where each polynomial is of a lower degree (e.g., a cubic polynomial). We partition the spectrum in $[-1, 1]$ (or $[0, 2]$ as needed) into contiguous intervals and approximate the desired frequency response by fitting a low degree polynomial in each interval. This process helps us to learn a more complex shaped frequency response as needed for the task. Let $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ denote a partition of the spectrum, containing m contiguous intervals and $h_{i,k_i}(\lambda;\ \alpha_i)$ denote a k_i-degree polynomial filter function defined over the interval σ_i (and 0 elsewhere) with polynomial coefficients α_i. We define piece-wise polynomial GNN (PP-GNN) filter function as:

$$h(\lambda) = \sum_{\sigma_i \in \mathcal{S}} h_{i,k_i}(\lambda;\ \alpha_i) \tag{4}$$

and learn a smooth filter function by imposing additional constraints to maintain continuity between polynomials of contiguous intervals at different endpoints (*aka* knots). This class of filter functions is rich, and its complexity is controlled by choosing intervals (i.e.,

endpoints and number of partitions) and polynomial degrees. Given the filter function, we compute the PP-GNN node embedding matrix as:

$$\mathbf{Z} = \sum_{\sigma_i \in \mathcal{S}} \mathbf{U}_i h(\lambda_{\sigma_\mathbf{i}}) \mathbf{U}_i^T \mathbf{Z}_x(\mathbf{X}; \mathbf{\Theta}) \tag{5}$$

where $\mathbf{U}_i$ is a matrix with columns as eigenvectors corresponding to eigenvalues that lie in σ_i and $h(\lambda_\sigma)$ is the diagonal matrix with diagonals containing the h_i evaluated at the eigenvalues lying in σ_i. Thus, the node embedding, $\mathbf{Z}$, is computed as a sum of outputs from a bank of polynomial filters with each filter operating over a spectral interval, σ_i.

4.2 Practical and Implementation Considerations

The filter function (5) requires computing full eigendecomposition of $\widetilde{\mathbf{A}}$ and is expensive, therefore, not scalable for very large graphs. We address this problem by performing eigendecomposition only for a few extreme values (i.e., at low and high ends of the spectrum) for sparse matrices, for which efficient algorithms exist [43] with corresponding off-the-shelf implementations. The primary motivation is that many recent works including GPR-GNN investigated the problem of designing robust graph neural networks that work well across homophilic and heterophilic graphs, and, they found that graph filters that amplify or attenuate low and high-frequency components of signals (i.e., low-pass and high-pass filters) are critical to improving performance on several benchmark datasets. However, there is still one question: *how do we extract signals from the remaining (middle) portion of the spectrum, and that too efficiently?* We answer this question as follows. Using the observation that the GPR-GNN method learns a graph filter but operates on the entire spectrum by sharing the filter coefficients across the spectrum, our proposal is to use an efficient variant of (4) as:

$$\tilde{h}(\lambda) = \eta_l \sum_{\sigma_i \in \mathcal{S}^l} h_i^{(l)}(\lambda; \gamma_i^{(l)}) + \eta_h \sum_{\sigma_i \in \mathcal{S}^h} h_i^{(h)}(\lambda; \gamma_i^{(h)}) + \eta_{gpr} h_{gpr}(\lambda; \gamma) \tag{6}$$

where $\mathcal{S}^l$ consists of partitions over low-frequency components, $\mathcal{S}^h$ consists of partitions over high-frequency components, the first and second terms fit piece-wise polynomials[2] in low/high-frequency regions, as indicated through superscripts. We refer PP-GNN models using only filters corresponding to the first and second terms alone in (6) as PP-GNN (Low) and PP-GNN (High), respectively. We extract any useful information from other frequencies in the middle region by adding the GPR-GNN filter function, $h_{gpr}(\lambda; \gamma)$ (the final term in 6), which is computationally efficient. Since $h_{gpr}(\lambda; \gamma)$ is a special case of (4) and the terms in (6) are additive, it is easy to see that (6) is same as (4) with a modified set of polynomial coefficients. Furthermore, we can control the contributions from each term by setting or optimizing over hyperparameters, η_l, η_h and η_{gpr}. Thus, the proposed model offers richer capability and flexibility to learn complex frequency response and balance computation costs over GPR-GNN. Please see Section A.3 for implementation details.

[2] For brevity, we dropped the polynomial degree dependency.

Model Training. Like GPR-GNN, we apply SOFTMAX activation function on (5) and use the standard cross-entropy loss function to learn the sets of polynomial coefficients (γ) and classifier model parameters ($\mathbf{\Theta}$) using labeled data. To ensure smoothness of the learned filter functions, we add a regularization term that penalizes squared differences between the function values of polynomials of contiguous intervals at each other's interval end-points. More details can be found in the supplementary material (A.3).

Discussion. In our model (4), we alleviate the over-smoothing problem using low-order polynomials, and learning complex and sharper frequency responses is feasible as we approximate higher-order polynomial functions effectively using several low-order piece-wise polynomials. However, this comes with eigendecomposition compute cost for a few (k) extreme eigenvalues, but is controllable by choosing k in an affordable way[3]. We observe this cost is (one time) pre-training cost and can be amortized over multiple rounds of model training required for the optimization of hyperparameters. Also, we need to compute each filter specific embedding with non-local eigen-graphs (via the operations, $\mathbf{U}_i H_i(\gamma_\mathbf{i})\mathbf{U}_i^T\mathbf{Z}_x(\mathbf{X};\mathbf{\Theta})$); thus, we lose (spatial) local neighborhood property of conventional methods like GPR-GNN. We compute node embeddings afresh whenever the model parameters are updated, thereby incurring an additional cost (over GPR-GNN) of $O(nkL)$ where k and L denote the number of selected low/high eigenvalues and classes, respectively. We conduct a comprehensive experimental study to assess the time taken by our method, compare against other state-of-the-art methods and present our findings in the experiment section.

4.3 Analysis

This section is arranged as follows: (a) Theorem 1 establishes superior capabilities of our model in approximating arbitrary filters than a standard polynomial filter; (b) Theorem 2 demonstrates the new space of filters that our model learns from, each region of which induces a controllable, strong bias towards certain parts of the spectrum while at the same time has dimension of the same order as the corresponding polynomial family.

Theorem 1. *For any frequency response h^*, and an integer $K \in \mathbb{N}$, let $\tilde{h} := h + h_f$, with h_f having a continuous support over a subset of the spectrum, σ_f. Assume that h and h_f are parameterized by independent K and K'-order polynomials, p and p_f, respectively, with $K' \leq K$. Then there exists $\tilde{h}$, such that $\min \|\tilde{h} - h^*\|_2 \leq \min \|h - h^*\|_2$, where the minimum is taken over the polynomial parameterizations. Moreover, for multiple polynomial adaptive filters $h_{f_1}, h_{f_2}, ..., h_{f_m}$ parameterized by independent K'-degree polynomials with $K' \leq K$ but having disjoint, contiguous supports, the same inequality holds for $\tilde{h} = h + \sum_{i=1}^{m} h_{f_i}$.*

For a detailed proof please refer to A.3 of the supplementary. We also conducted an experiment to illustrate the main conclusion of the above theorem in Section A.2 of the supplementary material.

[3] Most algorithms for this task utilize Lanczos' iteration, convergence bounds of which depends on the input matrix' spectrum [34,35], which although have superlinear convergence, but are observed to be efficient in practice.

Next, we note that since an actual waveform is not observed in practice and instead, we estimate it by optimizing over the observed labels via learning a graph filter, we theoretically show that the family of filters that we learn is a strict superset of the polynomial filter family. The same result holds for the families of the resulting adapted graphs.

Theorem 2. *Define* $\mathbb{H} := \{h(\cdot) \mid \forall$ *possible K-degree polynomial parameterizations of* $h\}$ *to be the set of all K-degree polynomial filters, whose arguments are* $n \times n$ *diagonal matrices, such that a filter response over some* $\mathbf{\Lambda}$ *is given by* $h(\mathbf{\Lambda})$ *for* $h(\cdot) \in \mathbb{H}$*. Similarly* $\mathbb{H}' := \{\tilde{h}(\cdot) \mid \forall$ *possible polynomial parameterizations of* $\tilde{h}\}$ *is set of all filters learnable via* PP-GNN *, with* $\tilde{h} = h + h_{f_1} + h_{f_2}$*, where* h *is parameterized by a K-degree polynomial supported over entire spectrum,* h_{f_1} *and* h_{f_2} *are localized adaptive filters parameterized by independent* K'*-degree polynomials which only act on top and bottom* t *diagonal elements respectively, with* $t < n/2$ *and* $K' \leq K$*; then* $\mathbb{H}$ *and* $\mathbb{H}'$ *form a vector space, with* $\mathbb{H} \subset \mathbb{H}'$*. Also,* $\frac{dim(\mathbb{H}')}{dim(\mathbb{H})} = \frac{K+2K'+3}{K+1}$.

Corollary 1. *The corresponding adapted graph families* $\mathbb{G} := \{\mathbf{U}h(\cdot)\mathbf{U}^T \mid \forall h(\cdot) \in \mathbb{H}\}$ *and* $\mathbb{G}' := \{\mathbf{U}\tilde{h}(\cdot)\mathbf{U}^T \mid \forall \tilde{h}(\cdot) \in \mathbb{H}'\}$ *for any unitary matrix* $\mathbf{U}$ *form a vector space, with* $\mathbb{G} \subset \mathbb{G}'$ *and* $\frac{dim(\mathbb{G}')}{dim(\mathbb{G})} = \frac{K+2K'+3}{K+1}$.

The above theorem can be trivially extended to an arbitrary number of adaptive filters with arbitrary support. The presence of each adaptive filter induces a bias in the model towards learning a bank of filters that operate only on the corresponding support. Since the number of filters and their support sizes are hyperparameters, tuning them offers control and flexibility to model richer frequency responses over the entire spectrum. Thus, our model learns from a more diverse space of filters and the corresponding adapted graphs. The result also implies that our model learns from a space of filters that is only $O(1)$-fold greater than that of polynomial filters[4]. Note that learning from this diverse region is feasible. This observation comes from the proofs of Theorem 4.2 and Corollary 4.2.1 (A.3 and A.3 in supplementary). Using the localized adaptive filters without any filter with the entire spectrum as support results in learning a set of adapted graphs, $\hat{\mathbb{G}}$. This set is disjoint from $\mathbb{G}$, with $\mathbb{G}' = \mathbb{G} \oplus \hat{\mathbb{G}}$. We conduct various ablative studies where we demonstrate the effectiveness of learning from $\hat{\mathbb{G}}$ and $\mathbb{G}'$.

Our model formulation is a generalization of the formulation by [8], and we show in Section A.3 of the supplementary material by extending their analysis to our model that it still inherits their property of mitigating oversmoothing effects when high degree polynomial is used. Our experiments show that we are able to obtain superior performance without needing the higher-order polynomials.

4.4 Comparison against other Filtering Methods

General FIR filter are a generalization of the polynomial filter family and thus a precursor to the models based on the latter. As per the study conducted in section A.5 of the supplementary, constraining the model is required to obtain better performance. Restricting to polynomial filters can be seen as having an implicit regularization on the same

[4] We leave the formal bias-variance analysis for adapted graph families as future work.

and we also empirically observe that such a restriction (restricting to polynomial filters) gives much better performance than constraining GFIR (see 5.1 and A.5) by simpler regularization methods such as L2 and/or dropout. We have also shown in theorem 2 that PP-GNN increases the space of graph filters (over GPR-GNN) and we observe in 5.1 that this increase in graph space results in an increased performance, over other polynomial filter methods. Thus, it requires a careful balance of the constraints imposed on the filter family, while also appropriately increasing the graph space to obtain better performance. A comprehensive study of this balance is beyond the scope of this work and we leave that as future work. Below, we first show the different ways of constraining the space (via polynomial filters) and compare them against PP-GNN.

Polynomial filters are a class of filters constructed and evaluated from polynomials. These filters can be constructed via multiple bases (for instance monomial, Bernstein) in the polynomial vector space.APPNP, GPR-GNN, and BERNNET are all instances of polynomial graph filters defined in different bases and with different constraints. Below, we illustrate the differences between these three methods and also discuss the shortcomings of each of them.

APPNP: One of the early works, APPNP [10], can be interpreted as a fixed polynomial graph filter that works with monomial basis. The polynomial coefficients correspond to Personalised PageRank (PPR) [44]. The node embeddings are learnt by APPNP as described in A.6. The main shortcoming of this method is the assumption that the optimal coefficients for the polynomial filter (for all tasks) are PPR coefficients, which need not necessarily be the case.

GPR-GNN: GPR-GNN builds on APPNP by overcoming this shortcoming by making the coefficients γ_k (see A.6) learnable. [8] identified that negative coefficients allows the model to exploit high frequency signals required for better performance on heterophilic graphs. GPR-GNN, like APPNP, uses the monomial basis. The node embeddings are learnt by GPR-GNN as described in A.6. While this method is an improvement over APPNP, adapting an arbitrary filter response which requires a high-order polynomial is difficult due to the oversmoothing problem. GPR-GNN mitigates oversmoothing by showing that the higher order terms' coefficients uniformly converge to zero during training. Mitigating the oversmoothing problem limits the complexity of the filter learnt, and therefore making GPR-GNN ineffective at learning complex frequency responses.

BERNNET: While oversmoothing is one shortcoming of GPR-GNN, BERNNET identified another shortcoming that GPR-GNN and other polynomial filtering based methods can result in ill-posed solutions and face optimization issues (converging to saddle points) by not constraining the filter response to non-negative values. [38] proposed a model that learns a non-negative frequency response, a constraint that can be easily enforced by modifying the learning problem from learning the coefficients of the monomial basis functions to learning the coefficients of the Bernstein basis functions, since the latter are non-negative in their standard domain. [38] argue that constraining coefficients to take on non-negative values is required for stability and interpretability of the learned filters and is the main reason for performance improvements. The node embeddings are learnt as described in A.6. Note that in the expression referenced, $\theta_k(\forall k)$ are learnable coefficients and are constrained to non-negative values. We first replace

$\frac{1}{2^K}\binom{K}{r}\sum_{p=0}^{q}\binom{K-r}{q-p}\binom{r}{p}(-1)^p$ with α_{rq} and then subsequently replace $\sum_{r=0}^{K}\theta_r\alpha_{rq}$ with w_q. Such an exercise was done to show that the filter defined by BERNNET does indeed fall into the class of polynomial filters. We tabulate the important attributes of each of the polynomial filters described above in Table 10 of the supplementary material.

All of these approaches run into the oversmoothing issue with an increase in the degree of the polynomial filter (A.1 of supplmentary). PP-GNN, owing to its piecewise definition, can model more complex shaped responses better without the need to increase the degree. Our proposed model only requires extremal eigendecomposition (i.e. computing only the extreme eigenpairs), for which there exists efficient algorithms to compute [45,46]. Further, as mentioned earlier, this is a one time pre-training cost, that can be amortized over training multiple models for hyper-parameter tuning. We illustrate this through a comprehensive empirical study in section A.7 of the supplementary material. In the next section, we experimentally show the benefits of PP-GNN.

5 Experiments

We conduct extensive experiments to demonstrate the effectiveness and competitiveness of the proposed method over standard baselines and state-of-the-art (SOTA) GNN methods. We conduct ablative studies to demonstrate the usefulness of different filters and the number of eigenpairs required in PP-GNN. We also compare the quality of the embeddings learned and the time to train different models. We first describe our experimental setup along with baselines and information on hyper-parameter tuning.

Datasets: We evaluate our model on several real-world heterophilic and homophilic datasets. The heterophilic datasets include **Texas**[5], **Wisconsin**[5], **Chameleon**, **Squirrel** [18] and **Flickr**. The homophilic datasets include **Ogbn-Arxiv**, **Wiki-CS**, **Citeseer**, **Pubmed**, **Cora**, **Computer**, and **Photos** borrowed from [3]. Please refer to A.4 for details on dataset statistics, splits and other preprocessing steps. We report the mean and standard deviation of test accuracy over splits to compare model performance.

Baselines. We compare our method against three category of methods: (a) standard LR (Logistic Regression) and MLP (Multi-Layer Perceptron), (b) traditional and spatial convolution-based GNN models including GCN, SGCN, SUPERGAT, TDGNN, H_2GCN, and GEOM-GCN, and (c) recent spectral convolution-based methods (with emphasis on graph filters) such as GPR-GNN, FAGCN, APPNP, LGC, ARMA, ADAGNN, BERNNET, GFIR, and UFG. Our tabular results are organized as per this grouping, along with references. In some cases, the grouping of spatial convolution-based methods is somewhat overlapping with spectral filtering based-methods since spectral interpretations are available for the former. Detailed descriptions for all the baselines, hardware and software specifications are provided in the supplementary material (A.4 and A.4). Our model implementation details along with hyper-parameters ranges description can be found in A.4.

5.1 PP-GNN versus SOTA Models

Heterophilic Datasets. We perform comprehensive experiments to show the effectiveness of PP-GNN on several Heterophilic graphs and tabulate the results in Table 1.

[5] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb

Table 1: Results on Heterophilic Datasets. '*' indicates that the results were borrowed from the corresponding papers. Bold indicates the best performing model; underline for second-best.

	Texas	Wisconsin	Squirrel	Chameleon	Cornell
LR	81.35 (6.33)	84.12 (4.25)	34.73 (1.39)	45.68 (2.52)	83.24 (5.64)
MLP	81.24 (6.35)	84.43 (5.36)	35.38 (1.38)	51.64 (1.89)	**83.78 (5.80)**
SGCN [16]	62.43 (4.43)	55.69 (3.53)	45.72 (1.55)	60.77 (2.11)	62.43 (4.90)
GCN [1]	61.62 (6.14)	58.82 (4.89)	47.78 (2.13)	62.83 (1.52)	62.97 (5.41)
SuperGAT [3]	61.08 (4.97)	56.47 (3.90)	31.84 (1.26)	43.22 (1.71)	57.30 (8.53)
Geom-GCN [4]	67.57*	64.12*	38.14*	60.90*	60.81*
H2GCN [5]	84.86 (6.77)*	86.67 (4.69)*	37.90 (2.02)*	58.40 (2.77)	82.16 (4.80)*
TDGNN [31]	83.00 (4.50)*	85.57 (3.78)*	43.84 (2.16)	55.20 (2.30)	82.92 (6.61)*
GFIR (unconstrained)	73.24 (6.91)	77.84 (3.21)	36.50 (1.12)	51.71 (3.11)	72.43 (7.62)
GFIR (constrained)	74.59 (4.45)	79.41 (3.10)	41.12 (1.17)	61.27 (2.42)	74.05 (7.77)
FAGCN [7]	82.43 (6.89)	82.94 (7.95)	42.59 (0.79)	55.22 (3.19)	79.19 (9.79)
APPNP [10]	81.89 (5.85)	85.49 (4.45)	39.15 (1.88)	47.79 (2.35)	81.89 (6.25)
LGC [22]	80.20 (4.28)	81.89 (5.98)	44.26 (1.49)	61.14 (2.07)	74.59 (3.42)
GPR-GNN [8]	81.35 (5.32)	82.55 (6.23)	46.31 (2.46)	62.59 (2.04)	78.11 (6.55)
AdaGNN [37]	71.08 (8.55)	77.70 (4.91)	53.50 (0.96)	65.45 (1.17)	71.08 (8.36)
BernNET [38]	83.24 (6.47)	84.90 (4.53)	52.56 (1.69)	62.02 (2.28)	80.27 (5.41)
ARMA [39]	79.46 (3.65)	82.75 (3.56)	47.37 (1.63)	60.24 (2.19)	80.27 (7.76)
UFG-ConvR [32]	66.22 (7.46)	68.63 (4.98)	42.06 (1.55)	56.29 (1.58)	69.19 (6.40)
PP-GNN	**89.73 (4.90)**	**88.24 (3.33)**	**59.15 (1.91)**	**69.10 (1.37)**	82.43 (4.27)

Table 2: Results on Homophilic Datasets.

	Cora-Full	Wiki-CS	Citeseer	Pubmed	Cora	Computer	Photos
LR	39.10 (0.43)	72.28 (0.59)	72.22 (1.54)	87.00 (0.40)	73.94 (2.47)	64.92 (2.59)	77.57 (2.29)
MLP	43.03 (0.82)	73.74 (0.71)	73.83 (1.73)	87.77 (0.27)	77.06 (2.16)	64.96 (3.57)	76.96 (2.46)
SGCN	61.31 (0.78)	78.30 (0.75)	76.77 (1.52)	88.48 (0.45)	86.96 (0.78)	80.65 (2.78)	89.99 (0.69)
GCN	59.63 (0.86)	77.64 (0.49)	76.47 (1.33)	88.41 (0.46)	87.36 (0.91)	82.50 (1.23)	90.67 (0.68)
SuperGAT	57.75 (0.97)	77.92 (0.82)	76.58 (1.59)	87.19 (0.50)	86.75 (1.24)	83.04 (1.02)	90.31 (1.22)
Geom-GCN	NA	NA	77.99*	90.05*	85.27*	NA	NA
H2GCN	57.83 (1.47)	OOM	77.07 (1.64)*	89.59 (0.33)*	87.81 (1.35)*	OOM	91.17 (0.89)
TDGNN	OOM	79.58 (0.51)	76.64 (1.54)*	89.22 (0.41)*	88.26 (1.32)*	84.52 (0.92)	92.54 (0.28)
GFIR (unconstrained)	60.87 (0.78)	79.15 (0.65)	75.83 (1.94)	88.47 (0.45)	87.93 (0.90)	78.39 (1.09)	89.26 (1.00)
GFIR (constrained)	60.92 (0.80)	79.15 (0.63)	76.24 (1.43)	88.47 (0.39)	87.46 (1.26)	79.57 (2.12)	89.38 (1.03)
FAGCN	60.07 (1.43)	79.23 (0.66)	76.80 (1.63)	89.04 (0.50)	88.21 (1.37)	82.16 (1.48)	90.91 (1.11)
APPNP	60.83 (0.55)	79.13 (0.50)	76.86 (1.51)	89.57 (0.53)	88.13 (1.53)	82.03 (2.04)	91.68 (0.62)
LGC	**61.84 (0.90)**	79.82 (0.49)	76.96 (1.73)	88.78 (0.51)	88.02 (1.44)	83.44 (1.77)	91.56 (0.74)
GPR-GNN	61.37 (0.96)	79.68 (0.50)	76.84 (1.69)	89.08 (0.39)	87.77 (1.31)	82.38 (1.60)	91.43 (0.89)
AdaGNN	59.57 (1.18)	77.87 (4.95)	74.94 (0.91)	89.33 (0..57)	86.72 (1.29)	81.27 (2.10)	89.93 (1.22)
BernNET	60.77 (0.92)	79.75 (0.52)	77.01 (1.43)	89.03 (0.55)	88.13 (1.41)	83.69 (1.99)	91.61 (0.51)
ARMA	60.23 (1.21)	78.94 (0.32)	78.15 (0.74)	88.73 (0.52)	87.37 (1.14)	78.55 (2.62)	90.26 (0.48)
UFG-ConvR	60.98 (0.82)	78.56 (0.43)	76.74 (1.33)	85.68 (0.62)	87.93 (1.52)	80.01 (1.78)	90.20 (1.41)
PP-GNN	61.42 (0.79)	**80.04 (0.43)**	**78.25 (1.76)**	**89.71 (0.32)**	**89.52 (0.85)**	**85.23 (1.36)**	**92.89 (0.37)**

Table 3: Results on Large Datasets.

	LR	MLP	GCN	SGCN	SuperGAT	H2GCN	FAGCN	APPNP	LGC	GPR-GNN	BernNet	TDGNN	UFG-ConvR	ARMA	AdaGNN	PP-GNN
Flickr	46.51	46.93	53.40	50.75	53.47	OOM	OOM	50.33	51.67	52.74	52.35	OOM	OOM	53.79	52.30	**55.30**
OGBN-arXiv	52.53	54.96	69.37	68.51	55.1*	OOM	OOM	69.20	**69.64**	68.44	69.21	OOM	OOM	69.49	69.44	69.28

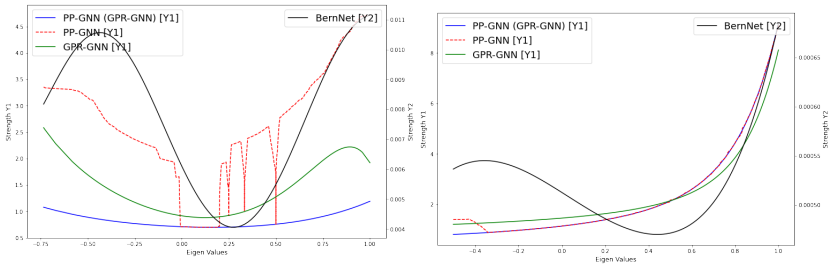
Table 4: Performance of different filters

Test Acc	Squirrel	Chameleon	Citeseer	Cora
PP-GNN (Low)	45.75 (1.69)	56.73 (4.03)	76.23 (1.54)	88.03 (0.79)
PP-GNN (High)	58.70 (1.60)	**69.19 (1.88)**	55.50 (6.38)	73.76 (2.03)
PP-GNN (GPR-GNN+Low)	50.96 (1.26)	63.71 (2.69)	78.07 (1.71)	**89.56 (0.93)**
PP-GNN (GPR-GNN + High)	60.39 (0.91)	67.83 (2.30)	**78.30 (1.60)**	89.42 (0.97)
GPR-GNN	42.06 (1.55)	56.29 (1.58)	76.74 (1.33)	87.93 (1.52)
PP-GNN	**59.15 (1.91)**	69.10 (1.37)	78.25 (1.76)	89.52 (0.85)

Datasets like Texas, Wisconsin, and Cornell contain graphs with high levels of Heterophily and *rich node features*. Standard non-graph baselines like LR and MLP perform competitively or better on these datasets compared to many spatial and spectral-based methods. PP-GNN offers significant lifts in performance with gains of up to $\sim$6%. The node features in datasets like Chameleon and Squirrel are not adequately discriminative, and significant improvements are possible via convolutions, as we compare non-graph and graph-based methods in Table 1. Spatial GNN methods, in general, offer improvements over non-graph counterparts. In specific, methods like GCN, which also have a spectral connotation, show better performance on these datasets. We observe from the Table that Spectral methods offer additional improvements over models like GCN. The difference in performance among spectral methods majorly comes from their ability to learn better frequency responses of graph filters. Our proposed model shows significant lifts over all the baselines with gains up to $\sim$6% and $\sim$4% on the Squirrel and Chameleon datasets. These improvements empirically support the efficacy of PP-GNN's filter design.

Homophilic Datasets. The input graphs for these datasets contain informative signals, and one can expect competitive task performance from even basic spatial-convolution based methods as observed in Table 2. We can see that spatial models are among the top performers for several Homophilic datasets. Existing spectral methods marginally improve over spatial methods on a few datasets. Not surprisingly, our PP-GNN model with effective filter design can exploit additional discriminatory signals from an already rich informative source of signals. PP-GNN offers additional gains up to 1.3% over other baselines.

Large Datasets. We also observe gains on moderately large datasets like Flickr and perform competitively on the OGBN-arXiv dataset. Please note that our latter numbers are slightly inferior for baselines like GCN compared to the leaderboard [6] numbers. These differences are because we turn off the optimization tricks like Batch Normalization.



(a) Squirrel (b) Citeseer

Fig. 1: Learned filter responses of PP-GNN,GPR-GNN, and BERNNET.

[6] https://tinyurl.com/oarxiv

5.2 PP-GNN Model Investigation

We conducted several experimental studies to understand and illustrate how the PP-GNN model works. Our studies include: (a) how does the frequency response of PP-GNN look like?, (b) what happens when we learn only individual sub-filter banks (e.g., PP-GNN (Low), PP-GNN(Low + GPR-GNN)? and (c) does PP-GNN learn better embedding?

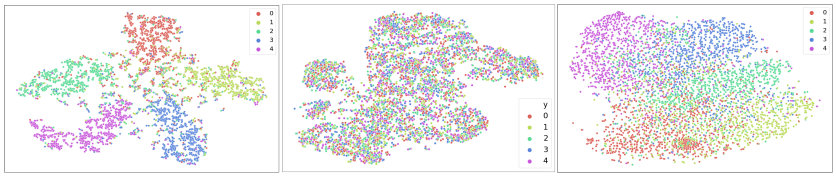
Frequency Response. In Figure 1a and 1b, we show the learned frequency responses (i.e., $h(\lambda)$) of the overall PP-GNN model, GPR-GNN component of PP-GNN (PP-GNN (GPR-GNN)), stand-alone (GPR-GNN) model and BERNNET model on the Squirrel and Citeseer datasets. For Squirrel (a heterophilic dataset), we can observe that while GPR-GNN and BERNNET learns the importance of low and high-frequency signals, it is unable to capture their relative strengths/importance adequately, and this happens due to the restriction of learning a single polynomial globally. PP-GNN learns sharper and richer responses at different parts of the spectrum, thereby improving classification accuracy. For Citeseer (a homophilic dataset) we can observe that all the models in comparison learn a smooth polynomial, GPR-GNN is not able to capture the complex transition that can be seen at the lower end of the spectrum, while BERNNET is doing it some degree. This inability to capture the complex transition leads to a lower classification accuracy. A similar trend can be found on two other datasets in A.4.

Role of Different Filter-banks. Recall that the PP-GNN model is a filter-bank model comprising several polynomial filters operating at different parts of the spectrum. We evaluate PP-GNN's performance by learning each group of filters alone (e.g., PP-GNN(Low), PP-GNN(High)) and report results on several datasets in Table 4. We see that the Heterophilic datasets (like Squirrel and Chameleon) largely benefit from high-frequency signals. In contrast, Homophilic datasets (Cora and Citeseer) exhibit a reverse trend. Incorporating the GPR-GNN filter as part of the PP-GNN filter helps to get improved performance over individual filters (PP-GNN(Low) or PP-GNN (High)) and shows considerable improvements over a wide variety of datasets.

Quality of learned embeddings: We qualitatively assess the difference in the learned embedding of PP-GNN, GPR-GNN and BERNNET. Towards this, we generated t-SNE plots of the learned node embeddings and visually inspected them. From Figure 2a, 2b and 2c, we observe that PP-GNN discovers more discriminative features resulting in discernible clusters on the Squirrel dataset compared to GPR-GNN and BERNNET, enabling PP-GNN to achieve significantly improved performance.

5.3 Training Time Comparison

We conducted a comprehensive training time evaluation study to compare the running-time performance of various models on diverse datasets. This study included measurement of the time taken for end-end training and by individual components over several hyperparameter configurations. Due to space constraints, we present several key observations here. We emphasize that eigenpairs' computation is a one time cost, and this cost can be amortized over the model training cost required to optimize on total hyper-parameters configurations. We also observe that the eigenpairs' compute cost, even for medium-sized graphs like Ogbn-ArXiv and Flickr is relatively low. Our end-end training time comparison results show that PP-GNN is $\sim$4x slower than GCN, $\sim$2x slower than GPR-GNNand BERNNET, and $\sim$2x faster than ADAGNN. Please refer to section A.7 of supplementary for more details.



(a) PP-GNN (b) GPRGNN (c) BERNNET

Fig. 2: t-SNE plots of learned embeddings on the Squirrel dataset

5.4 Additional Experiments

We summarize a list of experiments that can be found in the supplementary material. Studies on varying the number of eigenvectors used by PP-GNN and the importance of MLP can be found in Sections A.4 and A.4 respectively. Analysis on the effect of varying the order of GPR-GNN's polynomial on performance is presented in A.1. Experimental details for PP-GNNwith boundary regularization is in A.3.

6 Conclusion

Several recently proposed methods attempt to build robust models for diverse graphs exhibiting different correlations between graph and node labels. We build on the filter-based approach of GPR-GNN which can be extended further with Generalized FIR models. This work proposed an effective polynomial filter bank design using a piece-wise polynomial filtering approach. We combine GPR-GNN with additional polynomials resulting in a bank of filters that adapt to low and high-end spectrums using multiple polynomial filters. While our method makes an unconventional choice of extremal eigen-decomposition, it does help to get improved performance, albeit with some additional but manageable cost. Our experiments demonstrate that the proposed approach can learn effective filter functions that improve node classification accuracy significantly across diverse graphs. While our work shows merit, it is still founded upon the polynomial formulation, and even though piecewise polynomial filters are more expressive than conventional polynomial filters, they still retain the properties of the polynomial filters locally. Hence, there is still room for even more expressive filter formulations that are well motivated, and we leave their exploration as future work.

References

1. Kipf, T. & Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *International Conference On Learning Representations (ICLR)*. (2017)
2. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. & Bengio, Y. Graph Attention Networks. *International Conference On Learning Representations (ICLR)*. (2018)
3. Kim, D. & Oh, A. How to Find Your Friendly Neighborhood: Graph Attention Design with Self-Supervision. *International Conference On Learning Representations (ICLR)*. (2021)
4. Pei, H., Wei, B., Chang, K., Lei, Y. & Yang, B. Geom-GCN: Geometric Graph Convolutional Networks. *International Conference On Learning Representations (ICLR)*. (2020)

5. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L. & Koutra, D. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *Neural Information Processing Systems (NeurIPS)*. (2020)
6. Zhu, J., Rossi, R., Rao, A., Mai, T., Lipka, N., Ahmed, N. & Koutra, D. Graph Neural Networks with Heterophily. *Association For The Advancement Of Artificial Intelligence (AAAI)*. (2021)
7. Bo, D., Wang, X., Shi, C. & Shen, H. Beyond Low-frequency Information in Graph Convolutional Networks. *Association For The Advancement Of Artificial Intelligence (AAAI)*. (2021)
8. Chien, E., Peng, J., Li, P. & Milenkovic, O. Adaptive Universal Generalized PageRank Graph Neural Network. *International Conference On Learning Representations (ICLR)*. (2021)
9. Hamilton, W., Ying, R. & Leskovec, J. Inductive Representation Learning on Large Graphs. *Neural Information Processing Systems (NeurIPS)*. (2017)
10. Klicpera, J., Bojchevski, A. & Günnemann, S. Combining Neural Networks with Personalized PageRank for Classification on Graphs. *International Conference On Learning Representations (ICLR)*. (2019)
11. Bruna, J., Zaremba, W., Szlam, A. & LeCun, Y. Spectral Networks and Locally Connected Networks on Graphs. *International Conference On Learning Representations (ICLR)*. (2014)
12. Defferrard, M., Bresson, X. & Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Neural Information Processing Systems (NeurIPS)*. (2016)
13. Galstyan, S. MixHop: Higher-Order Graph Convolution Architectures via Sparsified Neighborhood Mixing. *International Conference On Machine Learning (ICML)*. (2019)
14. Lee, S. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. *Conference On Uncertainty In Artificial Intelligence (UAI)*. (2019)
15. Li, Q., Han, Z. & Wu, X. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *Association For The Advancement Of Artificial Intelligence (AAAI)*. (2018)
16. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T. & Weinberger, K. Simplifying Graph Convolutional Networks. *International Conference On Machine Learning (ICML)*. (2019)
17. Tang, J., Sun, J., Wang, C. & Yang, Z. Social Influence Analysis in Large-Scale Networks. *ACM SIGKDD International Conference On Knowledge Discovery And Data Mining (KDD)*. (2009)
18. Rozemberczki, B., Allen, C. & Sarkar, R. Multi-Scale attributed node embedding. *Journal Of Complex Networks*. (2021)
19. Kingma, D. & Ba, J. Adam: A Method for Stochastic Optimization. *International Conference On Learning Representations (ICLR)*. (2015)
20. Chua, T., Tang, J., Hong, R., Li, H., Luo, Z. & Zheng, Y. NUS-WIDE: A Real-World Web Image Database from National University of Singapore. *Proc. Of ACM Conf. On Image And Video Retrieval (CIVR'09)*.
21. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M. & Leskovec, J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *ArXiv Preprint ArXiv:2005.00687*. (2020)
22. Navarin, N., Erb, W., Pasa, L. & Sperduti, A. Linear Graph Convolutional Networks. *28th European Symposium On Artificial Neural Networks, Computational Intelligence And Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*. pp. 151-156 (2020)
23. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. *ArXiv*. **abs/1907.10902** (2019)
24. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances In Neural Information Processing Systems 32*. pp. 8024-8035 (2019)
25. Tremblay, N., Gonçalves, P. & Borgnat, P. Design of graph filters and filterbanks. (2017)

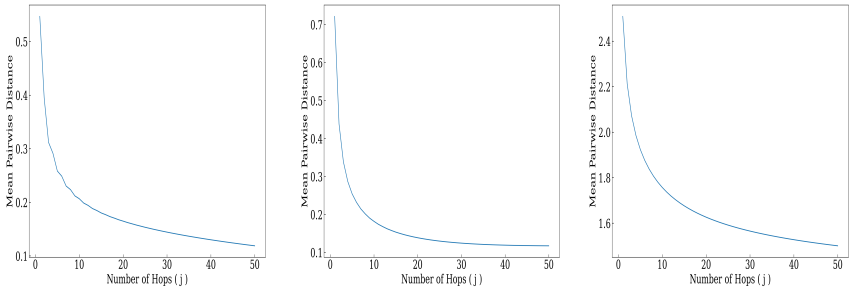
26. Shuman, D., Narang, S., Frossard, P., Ortega, A. & Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*. **30**, 83-98 (2013)
27. Lim, D., Li, X., Hohne, F. & Lim, S. New Benchmarks for Learning on Non-Homophilous Graphs. *The WebConf Workshop On Graph Learning Benchmarks (GLB-WWW)*. (2021)
28. Lukovnikov, D. & Fischer, A. Improving Breadth-Wise Backpropagation in Graph Neural Networks Helps Learning Long-Range Dependencies.. *Proceedings Of The 38th International Conference On Machine Learning*.
29. Chamberlain, B., Rowbottom, J., Gorinova, M., Bronstein, M., Webb, S. & Rossi, E. GRAND: Graph Neural Diffusion. *Proceedings Of The 38th International Conference On Machine Learning*. **139** pp. 1407-1418 (2021,7,18)
30. Yang, Y., Liu, T., Wang, Y., Zhou, J., Gan, Q., Wei, Z., Zhang, Z., Huang, Z. & Wipf, D. Graph Neural Networks Inspired by Classical Iterative Algorithms. *Proceedings Of The 38th International Conference On Machine Learning*.
31. Wang, Y. & Derr, T. Tree Decomposed Graph Neural Network. *Conference On Information And Knowledge Management*. (2021)
32. Zheng, X., Zhou, B., Gao, J., Wang, Y., Lió, P., Li, M. & Montufar, G. How Framelets Enhance Graph Neural Networks. *Proceedings Of The 38th International Conference On Machine Learning*.
33. Navarin, N., Erb, W., Pasa, L. & Sperduti, A. Linear Graph Convolutional Networks. *ESANN*. pp. 151-156 (2020)
34. Saad, Y. On the Rates of Convergence of the Lanczos and the Block-Lanczos Methods. *SIAM Journal On Numerical Analysis*. **17**, 687-706 (1980)
35. LI, R. SHARPNESS IN RATES OF CONVERGENCE FOR THE SYMMETRIC LANCZOS METHOD. *Mathematics Of Computation*. **79**, 419-435 (2010)
36. Cullum, J. & Willoughby, R. Lanczos Algorithms for Large Symmetric Eigenvalue Computations. (Society for Industrial,2002)
37. Dong, Y., Ding, K., Jalaian, B., Ji, S. & Li, J. Graph Neural Networks with Adaptive Frequency Response Filter. (2021)
38. He, M., Wei, Z., Huang, Z. & Xu, H. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. (2021)
39. Bianchi, F., Grattarola, D., Livi, L. & Alippi, C. Graph Neural Networks with Convolutional ARMA Filters. *IEEE Transactions On Pattern Analysis And Machine Intelligence*. pp. 1-1 (2021)
40. Gama, F., Marques, A., Leus, G. & Ribeiro, A. Convolutional Neural Network Architectures for Signals Supported on Graphs. *IEEE Transactions On Signal Processing*. **67**, 1034-1049 (2019,2)
41. Cai, C. & Wang, Y. A Note on Over-Smoothing for Graph Neural Networks. (2020)
42. Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S. & Hu, X. Dirichlet Energy Constrained Learning for Deep Graph Neural Networks. (2021)
43. Davidson, E. & Thompson, W. Monster Matrices: Their Eigenvalues and Eigenvectors. *Computers In Physics*. **7**, 519-522 (1993)
44. Wang, H., Wei, Z., Gan, J., Wang, S. & Huang, Z. Personalized PageRank to a Target Node, Revisited. *CoRR*. **abs/2006.11876** (2020)
45. Stewart, G. A Krylov–Schur Algorithm for Large Eigenproblems. *SIAM Journal On Matrix Analysis And Applications*. **23**, 601-614 (2002)
46. Lehoucq, R., Sorensen, D. & Yang, C. ARPACK Users Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.. (1997)
47. Shchur, O., Mumme, M., Bojchevski, A. & Günnemann, S. Pitfalls of graph neural network evaluation. *ArXiv Preprint ArXiv:1811.05868*. (2018)

A Appendix

The appendix is structured as follows. In Section A.1, we present additional evidence of the limitations of GPR-GNN. In Section A.2, we show a representative experiment that motivates Section 4. In Section A.3, we provide proofs for theorems, propositions and corollaries defined in Section 3 and 4.3. In Section A.4, we provide more details regarding the baselines, datasets, their respective splits and additional implementation details including hyper-parameter ranges. We also provide details and results of additional experiments. In Section A.5, we provide details on GFIR and compare our proposed model against it. Section A.6 provides additional information on differences between our model and other polynomial filtering methods. In Section A.7, we provide a comprehensive timing analysis.

A.1 Motivation

Node Feature Indistinguishably Plots In the main paper (Figure 5a), we plot the average of pairwise distances between node features for the Cora dataset, after computing $\widetilde{\mathbf{A}}^j X$ for increasing j values, and showed that the mean pairwise node feature distance decreases as j increases. We observe that this is consistent across three more datasets: Citeseer, Chameleon and Squirrel. This is observed in Figure 3.

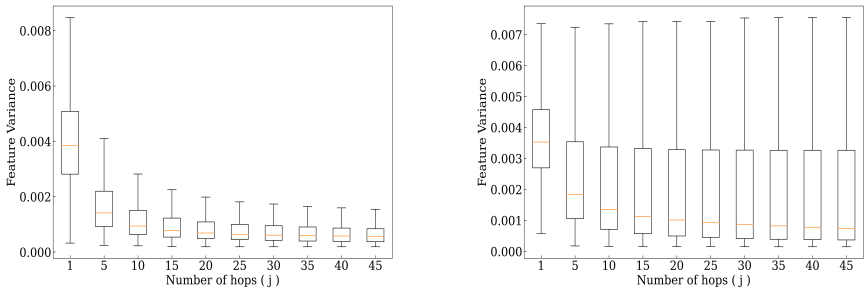


(a) Chameleon (b) Squirrel (c) Citeseer

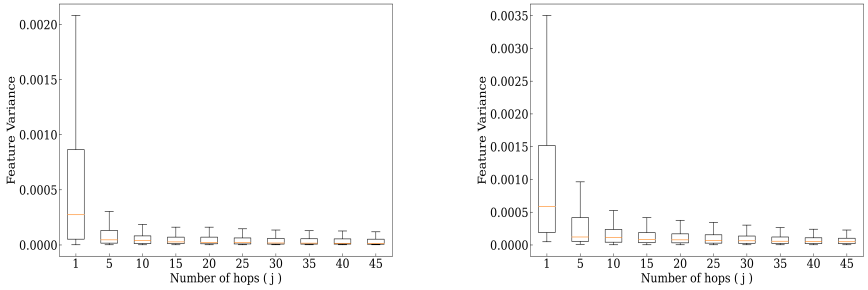
Fig. 3: Average of pairwise distances between node features, after computing $\widetilde{\mathbf{A}}^j X$, for increasing j values

We also observed the mean of the variance of each dimension of node features, after computing $\widetilde{\mathbf{A}}^j X$, for increasing j values. We observe that this mean does indeed reduce as the number of hops increase. We also observe that the variance of each dimension of node features reduces for Cora, Squirrel and Chameleon as the number of hops increase; however, we don't observe such an explicit phenomenon for Citeseer. See Figure 4.

Effect of varying the order of the GPR-GNN Polynomial In the main paper (Figure 5a), we plot the test accuracies of the GPR-GNN model while increasing the order

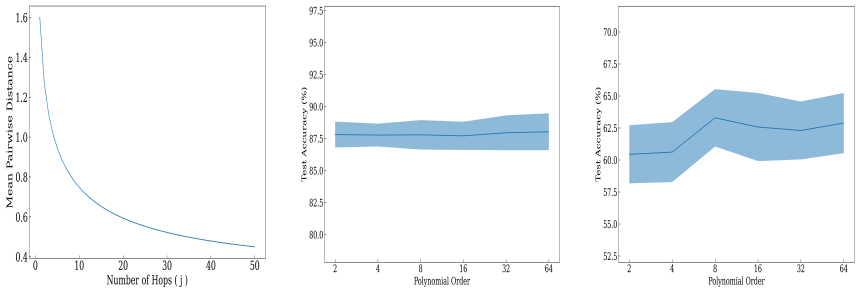


(a) Cora (b) Citeseer



(c) Squirrel (d) Chameleon

Fig. 4: Variance of each dimension of node features

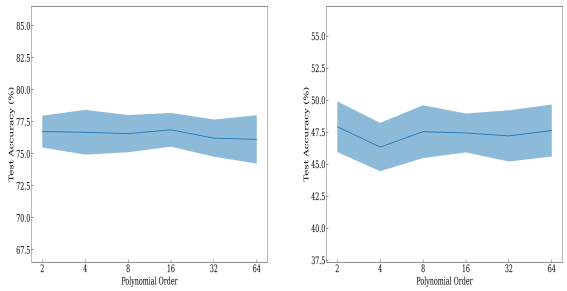


(a) (b) (c)

Fig. 5: In (a), we plot the average of pairwise distances between node features for the Cora dataset, after computing $\widetilde{\mathbf{A}}^j X$ for increasing j values. X-axis represents the various powers j and the Y-axis represents the average of pairwise distances between node features. In (b) and (c), we plot the test accuracies of the model in [8] for increasing order of polynomials for Cora and Chameleon dataset respectively. X-axis represents the order of the polynomial and Y-axis represents the test accuracy achieved for that order.

of the polynomials for the Cora and Chameleon dataset, respectively. We observe that on

increasing the polynomial order, the accuracies do not increase any further. We can show a similar phenomenon on two other datasets, Squirrel and Citeseer, in Figure 6.



(a) Citeseer (b) Squirrel

Fig. 6: Accuracy of the GPR-GNN model on inceasing the order of the polynomial

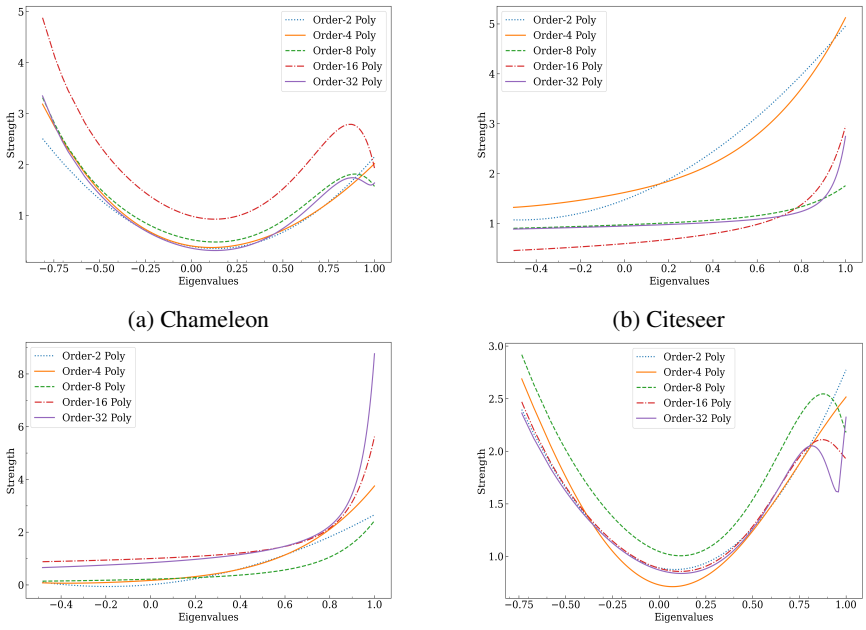
In Section 3 of the main paper, we claim that due to the over-smoothing effect, even on increasing the order of the polynomial, there is no improvement in the test accuracy. Moreover, in Figure 1 we can see that our model can learn a complicated filter polynomial while GPR-GNN cannot. This section shows that even on increasing the order of the GPR-GNNpolynomial, neither does the test accuracy increase nor does the waveform become as complicated as PP-GNN. See Figure 7.

A.2 Fictitious Polynomial

In Section 4, we claim that having multiple disjoint low order polynomials can approximate a complicated waveform more effectively than a single higher-order polynomial. To demonstrate, we create a representative experiment that shows this phenomenon by creating a fictitious complicated polynomial and try to fit it using a single unconstrained polynomial (representative of GPR-GNN), a single constrained polynomial (indicative of BERNNET, where the coefficients should be non-negative) and a disjoint piece-wise polynomial (indicative of PP-GNN). We setup a least square optimization problem to obtain the coefficients for these different polynomial variants. We evaluate and plot these polynomials in Figure 8. To quantify the effectiveness of different polynomial variants, we compute the approximation error (RMSE) with respect to the optimal waveform. We observe that piece-wise polynomials achieve much lower RMSE (1.5053) compared to constrained polynomial (3.9659) and unconstrained polynomial (3.3854)

A.3 Proposed Approach

Details regarding boundary regularization To induce smoothness in the learned filters, we add a regularization term that penalizes squared differences between function values of polynomials at knots (endpoints of contiguous bins). Our regularization term looks as follows:



(a) Chameleon (b) Citeseer

(c) Cora (d) Squirrel

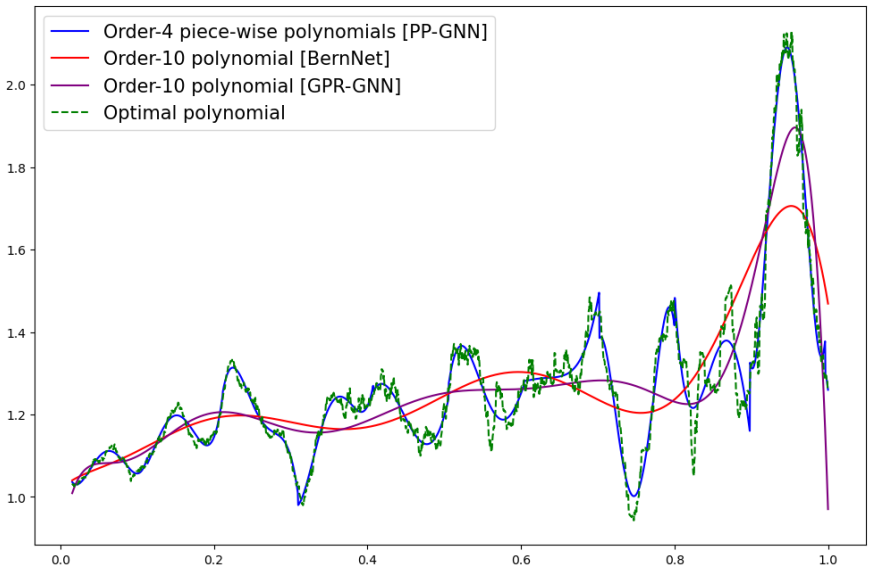
Fig. 7: Varying Polynomial Order in GPR-GNN

$$\sum_{i=1}^{m-1} \exp^{-(\sigma_i^{max} - \sigma_{i+1}^{min})^2} (h_i(\sigma_i^{max}) - h_{i+1}(\sigma_{i+1}^{min}))^2 \tag{7}$$

In equation 7, σ_i^{max} and σ_i^{min} refer to the maximum and minimum eigenvalues in σ_i (Refer to Section 4). This regularization term is added to the Cross-Entropy loss. We perform experiments with this model and report the performance in Table 5. We observe that we are able to reach similar performance even without the presence of this regularization term. Therefore, majority of the results reported in our Main paper are without this regularization term.

Table 5: Results with and without boundary regularization

Test Acc	**Computer**	**Chameleon**	**Citeseer**	**Cora**	**Squirrel**
PPGNN (With Reg)	83.53 (1.67)	67.92 (2.05)	76.85 (2)	88.19 (1.19)	55.42 (2.1)
PPGNN	85.23 (1.36)	69.10 (1.37)	78.25 (1.76)	89.52 (0.85)	59.15 (1.91)



(a) Fitting single 10-degree polynomial

Fig. 8: To demonstrate the effectiveness of adaptive polynomial filter, we try to approximate a complex waveform (green dashed line) via (a) 10 disjoint adaptive polynomial filters of order 4 (colored blue) (b) a single constrained order 10 polynomial (colored red), (c) an unconstrained order 10 polynomial (colored purple). The corresponding RMSE values are: (a) 1.5053, (b) 3.9659, (c) 3.3854

Implementing the Filter in Practice We provide more details to explain the filtering operation. An Equation similar to Equation 2 can be derived for our model by substituting Equation 6 from the paper into Equation 5. On substitution, we get:

$$Z = \eta_l \sum_{\sigma_i \in S^l} \sum_{j=1}^{k_1} \gamma_{ij}^{(l)} U_{\sigma_i}^{(l)} \left(\Lambda_{\sigma_i}^{(l)}\right)^j U_{\sigma_i}^{(l)T} Z_0(X;\theta) + \eta_h \sum_{\sigma_i \in S^h} \sum_{j=1}^{k_2} \gamma_{ij}^{(h)} U_{\sigma_i}^{(h)} \left(\Lambda_{\sigma_i}^{(h)}\right)^j U_{\sigma_i}^{(h)T} Z_0(X;\theta) + \eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta) \quad (8)$$

where

$$\widetilde{A}_i^{(l)} = U_{\sigma_i}^{(l)} \Lambda_{\sigma_i}^{(l)} U_{\sigma_i}^{(l)T}, \sigma_i \in S^l$$
$$\widetilde{A}_i^{(h)} = U_{\sigma_i}^{(h)} \Lambda_{\sigma_i}^{(h)} U_{\sigma_i}^{(h)T}, \sigma_i \in S^h$$

$U^{(l)}_{(\sigma_i)}$, $\Lambda^{(l)}_{(\sigma_i)}$ are the matrices containing eigenvectors and eigenvalues corresponding to the partition σ_i of the low frequency components and $U^{(h)}_{(\sigma_i)}$, $\Lambda^{(h)}_{(\sigma_i)}$ are the matrices containing eigenvectors and eigenvalues corresponding to the partition σ_i of the high frequency components and $\tilde{A}$ (See Equation 1 in the main paper) where $U_{(\sigma_i)} \in \mathbb{R}^{n \times |\sigma_i|}$ and $\Lambda_{(\sigma_i)} \in \mathbb{R}^{|\sigma_i| \times |\sigma_i|}$ with latter being a diagonal matrix.

The way we have implemented the filter is that we pre-compute the top and bottom eigenvalues/vectors of $\tilde{A}$ and use them to compute partition specific node embeddings. Note that Equation 8 can be rewritten as:

$$\begin{aligned} Z = \eta_l & \sum_{\sigma_i \in S^l} U^{(l)}_{\sigma_i} H^{(l)}_{\sigma_i} U^{(l)T}_{\sigma_i} Z_0(X;\theta) \\ & + \eta_h \sum_{\sigma_i \in S^h} U^{(h)}_{\sigma_i} H^{(h)}_{\sigma_i} U^{(h)T}_{\sigma_i} Z_0(X;\theta) \\ & + \eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta) \end{aligned} \tag{9}$$

where $H^{(l)}_{\sigma_i} = \sum_{j=1}^{k_1} \gamma^{(l)}_{ij} \left(\Lambda^{(l)}_{\sigma_i}\right)^j$ and $H^{(h)}_{\sigma_i} = \sum_{j=1}^{k_2} \gamma^{(h)}_{ij} \left(\Lambda^{(h)}_{\sigma_i}\right)^j$ form the effective low and high frequency component filters. Thus, a weighted combination of low and high frequency component based embeddings (i.e., the first and second term in Equation above) and the GPR-GNN term based embedding (i.e., third term) is computed. We implement our model based on equation 9.

Note that the following discussion is just for illustration purpose and we do not explicitly calculate the newer terms introduced here: We can also interpret the GPR-GNN term in terms of piece-wise polynomial filters defined on a mutually exclusive partition of the spectrum, with a difference that the coefficients and the order of the polynomial are shared across all partitions:

$$\begin{aligned} \eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta) = \eta_{gpr} (& \sum_{\sigma_i \in S^l} U^{(l)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(l)T}_{\sigma_i} \\ & + \sum_{\sigma_i \in S^h} U^{(h)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(h)T}_{\sigma_i} \\ & + \sum_{\sigma_i \in S^{mid}} U^{(mid)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(mid)T}_{\sigma_i}) Z_0(X;\theta) \end{aligned} \tag{10}$$

where $\mathcal{S}^{mid} := \mathcal{S} - (\mathcal{S}^l \cup \mathcal{S}^h)$, and $H^{(gpr)}_{\sigma_i} = \sum_{j=1}^{k_3} \gamma_j \left(\Lambda^{(mid)}_{\sigma_i}\right)^j$. Hence, we can club the respective terms of the partitions and obtain the final embeddings as:

$$
\begin{aligned}
Z = & \sum_{\sigma_i \in S^l} U_{\sigma_i}^{(l)} (\eta_l H_{\sigma_i}^{(l)} + \eta_{gpr} H_{\sigma_i}^{(gpr)}) U_{\sigma_i}^{(l)T} Z_0(X;\theta) \\
& + \sum_{\sigma_i \in S^h} U_{\sigma_i}^{(h)} (\eta_h H_{\sigma_i}^{(h)} + \eta_{gpr} H_{\sigma_i}^{(gpr)}) U_{\sigma_i}^{(h)T} Z_0(X;\theta) \\
& + \eta_{gpr} \sum_{\sigma_i \in S^{mid}} U_{\sigma_i}^{(mid)} H_{\sigma_i}^{(gpr)} U_{\sigma_i}^{(mid)T} Z_0(X;\theta)
\end{aligned} \tag{11}
$$

From equation 11, it is clear that PP-GNN also adapts the responses from the middle parts of the spectrum, albeit by a single polynomial. One can also interpret each term of equation 11 as an effective polynomial filter acting only on the corresponding part of the spectrum, with each effective polynomial filter can be influenced by a shared polynomial filter.

Notation Used Vectors are denoted by lower case bold Roman letters such as $\mathbf{x}$, and all vectors are assumed to be column vectors. In the paper, h with any sub/super-script refers to a frequency response, which is also considered to be a vector. A superscript T denotes the transpose of a matrix or vector; Matrices are denoted by bold Roman upper case letters, such as $\mathbf{M}$. A field is represented by $\mathbb{K}$; sets of real and complex numbers are denoted by $\mathbb{R}$ and $\mathbb{C}$ respectively. $\mathbb{K}[x_1, \ldots, x_n]$ denotes a multivariate polynomial ring over the field $\mathbb{K}$, in indeterminates $x_1, \ldots, x_n$. Set of $n \times n$ square matrices with entries from some set $\mathbb{S}$ are denoted by $\mathbb{M}_n(\mathbb{S})$. Moore-Penrose pseudoinverse of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^\dagger$. Eigenvalues of a matrix are denoted by λ, with $\lambda_1, \lambda_2, \ldots$ denoting a decreasing order when the eigenvalues are real. A matrix $\mathbf{\Lambda}$ denotes a diagonal matrix of eigenvalues. Set of all eigenvalues, i.e., spectrum, of a matrix is denoted by $\sigma_{\mathbf{A}}$ or simply σ when the context is clear. L_p norms are denoted by $\|\cdot\|_p$. Frobenius norm over matrices is denoted by $\|\cdot\|_F$. Norms without a subscript default to L_2 norms for vector arguments and Frobenius norm for matrices. $\oplus$ denotes a direct sum. For maps f_i defined from the vector spaces $V_1, \cdots, V_m$, with a map of the form $f : V \mapsto W$, with $V = V_1 \oplus V_2 \oplus \cdots \oplus V_m$, the the phrase "$f : V \mapsto W$ by mapping $f(\mathbf{v_i})$ to $f_i(g(\mathbf{v_i}))$" means that f maps a vector $\mathbf{v} = \mathbf{v_1} + \ldots + \mathbf{v_m}$ with $\mathbf{v}_i \in V_i$ to $f_1(g(\mathbf{v_1})) + \ldots + f_m(g(\mathbf{v_m}))$

Proof of Theorem 1

Theorem. *For any desired frequency response h^*, and an integer $K \in \mathbb{N}$, let $\tilde{h} := h + h_f$, with h_f having a continuous support over a subset of the spectrum, σ_f. Assuming h and h_f to be parameterized by independent K and K'-order polynomials p and p_f respectively, with $K' \leq K$, then there exists $\tilde{h}$, such that $\min \|\tilde{h} - h^*\|_2 \leq \min \|h - h^*\|_2$, where the minimum is taken over the polynomial parameterizations. Moreover, for multiple polynomial adaptive filters $h_{f_1}, h_{f_2}, ..., h_{f_m}$ parameterized by independent K'-degree polynomials with $K' \leq K$ but having disjoint, contiguous supports, the same inequality holds for $\tilde{h} = h + \sum_{i=1}^{m} h_{f_i}$.*

Proof. We make the following simplifying assumptions:

1. $|\sigma_{f_i}| > K, \quad \forall i \in [m]$, i.e., that is all support sizes are lower bounded by K (and hence K')
2. All eigenvalues of the reference matrix are distinct

For methods that use a single polynomial filter, the polynomial graph filter, $h_K(\mathbf{\Lambda}) = diag(\mathbf{V}\gamma)$ where γ is a vector of coefficients (i.e, γ parameterizes h), with eigenvalues sorted in descending order in components, and $\mathbf{V}$ is a Vandermonde matrix:

$$\mathbf{V} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^K \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^K \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \cdots & \lambda_n^K \end{bmatrix}$$

And to approximate a frequency response h^*, we have the following objective:

$$\begin{aligned} \min \|h - h^*\|_2^2 &:= \min_{\gamma} \|diag(h^*) - diag(\mathbf{V}\gamma)\|_F^2 \\ &= \min_{\gamma} \|h^* - \mathbf{V}\gamma\|_2^2 \\ &= \min_{\gamma} \|\mathbf{e}_p(\gamma)\|_2^2 \end{aligned}$$

Where $\|\|_F$ and $\|\|_2$ are the Frobenius and L_2 norms respectively. Due to the assumptions, the system of equations $h^* = \mathbf{V}\gamma$ is well-defined and has a unique minimizer, $\gamma^* = \mathbf{V}^\dagger h^*$, and thus $\|\mathbf{e}_p(\gamma^*)\| = \min_\gamma \|\mathbf{e}_p(\gamma)\|$. Next we break this error vector as:

$$\begin{aligned} \mathbf{e}_p(\gamma^*) &:= h^* - \mathbf{V}\gamma^* \\ &= \sum_{i=1}^{m} (h_i^* - \mathbf{V}_i\gamma^*) + (h_L^* - \mathbf{V}_L\gamma^*) \\ &:= \sum_{i=1}^{m} \mathbf{e}_{p_i}^* + \mathbf{e}_{p_L}^* \end{aligned}$$

Where $\mathbf{e}_{p_i}^* := (h_i^* - \mathbf{V}_i\gamma^*)$ with similar definition for $\mathbf{e}_{p_L}$; h_i^* is a vector whose value at components corresponding to the set $\sigma(h_{f_i})$ is same as that of h^* and rest are zero. Similarly, $\mathbf{V}_i^*$ is a matrix whose rows corresponding to the set $\sigma(h_{f_i})$ are same as that of $\mathbf{V}$ with other rows being zero. Also, $\mathbf{V}_L = \mathbf{V} - \sum_{i=0}^{m} \mathbf{V}_i$ and $h_L^* = h^* - \sum_{i=0}^{m} h_i^*$. Note that as a result of this construction, $[\mathbf{e}_{p_i^*}] \cup \mathbf{e}_{p_L^*}$ is a linearly independent set since the supports $[\sigma(h_{f_i})]$ form a disjoint set (note the theorem statement). We split the proof in two cases:

Case 1: $K' = K$. We now analyze the case where we have m polynomial adaptive filters added, all having an order of K, where the objective is $\min \|\tilde{h} - h^*\|$, which can be written as:

$$\min_{\gamma,[\gamma_i]} \left\| diag(h^*) - diag\left(\mathbf{V}\gamma + \sum_{i=0}^{m} \mathbf{V}_i\gamma_i\right)\right\|_F^2$$
$$= \min_{\gamma,[\gamma_i]} \left\| h^* - \mathbf{V}\gamma - \sum_{i=0}^{m} \mathbf{V}_i\gamma_i \right\|_2^2$$
$$= \min_{\gamma,[\gamma_i]} \|\mathbf{e}_g(\gamma,[\gamma_i])\|_2^2$$

Before characterizing the above system, we break a general error vector as:

$$\mathbf{e}_g(\gamma,[\gamma_i]) := h^* - \mathbf{V}\gamma - \sum_{i=0}^{m} \mathbf{V}_i\gamma_i$$
$$= \sum_{i=1}^{m} (h_i^* - \mathbf{V}_i(\gamma+\gamma_i)) + (h_L^* - \mathbf{V}_L\gamma)$$
$$:= \sum_{i=1}^{m} \mathbf{e}_{g_i} + \mathbf{e}_{g_L}$$

Where $\mathbf{e}_{g_i} := (h_i^* - \mathbf{V}_i(\gamma+\gamma_i))$ with similar definition for $\mathbf{e}_{g_L}$. Clearly, the systems of equations, $\mathbf{e}_{g_i} = 0, \ \ \forall i$ and $\mathbf{e}_{g_L} = 0$ are well-defined due to the assumptions 1 and 2. Since all the systems of equations have independent argument, unlike in the polynomial filter case where the optimization is constrained over a single variable; one can now resort to individual minimization of squared norms of $\mathbf{e}_{g_i}$ which results in a minimum squared norm of $\mathbf{e}_g$. Thus, we can set:

$$\gamma = \mathbf{V}_L^\dagger h_L^* = \gamma_g^* \quad \gamma_i = \mathbf{V}_i^\dagger h_i^* - \mathbf{V}_L^\dagger h_L^* = \gamma_i^*, \ \ \forall i \in [m]$$

to minimize squared norms of $\mathbf{e}_{g_i}$ and $\mathbf{e}_{g_L}$. Note that $[\mathbf{e}_{g_i}] \cup \mathbf{e}_{g_L}$ is a linearly independent set since the supports $[\sigma(h_{f_i})]$ form a disjoint set and by the above construction, this is also an orthogonal set, and hence we have $\|\mathbf{e}_g\|^2 = \sum_{i=1}^m \|\mathbf{e}_{g_i}\|^2 + \|\mathbf{e}_{g_L}\|^2$, and hence the above assignment implies:

$$\|\mathbf{e}_g(\gamma_g^*,[\gamma_i^*])\| = \min_{\gamma,[\gamma_i]} \|\mathbf{e}_g(\gamma_g,[\gamma_i])\| := \min \|\tilde{h} - h^*\|_2$$

Hence, it follows that, $\min_x \|h_i^* - \mathbf{V}_i x\|^2 = \left\|\mathbf{e}_{g_i}^*\right\|^2 \leq \left\|\mathbf{e}_{p_i}^*\right\|^2 = \|h_i^* - \mathbf{V}_i\gamma^*\|^2$ and $\min_x \|h_L^* - \mathbf{V}_L x\|^2 = \left\|\mathbf{e}_{g_L}^*\right\|^2 \leq \left\|\mathbf{e}_{p_L}^*\right\|^2 = \|h_L^* - \mathbf{V}_L\gamma^*\|^2$. Hence,

$$\sum_{i=1}^{m} \left\|\mathbf{e}_{g_i}^*\right\|^2 + \left\|\mathbf{e}_{g_L}^*\right\|^2 \leq \sum_{i=1}^{m} \left\|\mathbf{e}_{p_i}^*\right\|^2 + \left\|\mathbf{e}_{p_L}^*\right\|^2$$
$$\min \|\tilde{h} - h^*\| \leq \min \|h - h^*\|$$

Case 2: $K' < K$. We demonstrate the inequality showing the existence of an $\tilde{h}$ that achieves a better approximation error. By definition, the minimum error too will be

bounded above by this error. For this, we fix γ, the parameterization of h as $\gamma = \mathbf{V}^{\dagger}h^* = \gamma_p^*$ (say). Note that $\gamma_{p*} = \arg\min_{\gamma} \|\mathbf{e}_p(\gamma)\|$. Now our objective function becomes

$$\begin{aligned}\mathbf{e}_g(\gamma_p^*, [\gamma_i]) &:= h^* - \mathbf{V}\gamma_p^* - \sum_{i=0}^{m} \mathbf{V}_i'\gamma_i \\ &= \sum_{i=1}^{m}(h_i^* - \mathbf{V}_i\gamma_p^* + \mathbf{V}_i'\gamma_i) + (h_L^* - \mathbf{V}_L\gamma_p^*) \\ &= \sum_{i=1}^{m} \mathbf{e}'_{g_i} + \mathbf{e}'_{g_L}\end{aligned}$$

Where $h_i^*, h_L^*, \mathbf{V}_i, \mathbf{V}_L$ have same definitions as that in case 1 and $\mathbf{V}_i'$ is a matrix containing first $K'+1$ columns of $\mathbf{V}_i$ as its columns (and hence has full column rank), and, $\gamma_i \in \mathbb{R}^{K'+1}$. By this construction, we have

$$\|\mathbf{e}_g(\gamma_p^*, \mathbf{0})\| = \min_{\gamma} \|\mathbf{e}_p(\gamma)\| = \|\mathbf{e}_p(\gamma_p^*)\|$$

Our optimization objective becomes $\min_{[\gamma_i]} \|\mathbf{e}_g(\gamma_p^*, [\gamma_i])\|$, which is easy since the problem is well-posed by assumption 1 and 2. The unique minimizer of this is obtained by setting

$$\gamma_i = \mathbf{V}_i'^{\dagger}(h_i^* - \mathbf{V}_i\gamma_p^*) = \gamma_i^* \ (say) \ \ \forall i \in [m]$$

Now,

$$\|\mathbf{e}_g(\gamma_p^*, [\gamma_i^*])\| = \min_{[\gamma_i]} \|\mathbf{e}_g(\gamma_p^*, [\gamma_i])\| \leq \|\mathbf{e}_g(\gamma_p^*, \mathbf{0})\|$$

and,

$$\|\mathbf{e}_g(\gamma_p^*, \mathbf{0})\| = \min_{\gamma} \|\mathbf{e}_p(\gamma)\| = \min \|h - h^*\|$$

By the definition of minima, $\min_{\gamma,[\gamma_i]} \|\mathbf{e}_g(\gamma, [\gamma_i])\| \leq \min_{[\gamma_i]} \|\mathbf{e}_g(\gamma_p^*, [\gamma_i])\|$, and by the definition, $\min \|\tilde{h} - h^*\| = \min_{\gamma,[\gamma_i]} \|\mathbf{e}_g(\gamma, [\gamma_i])\|$, we have:

$$\min \|\tilde{h} - h^*\| \leq \min \|h - h^*\|$$

□

Proof of Theorem 2

Theorem. *Define* $\mathbb{H} := \{h(\cdot) \mid \forall$ *possible K-degree polynomial parameterizations of* $h\}$ *to be the set of all K-degree polynomial filters, whose arguments are* $n \times n$ *diagonal matrices, such that a filter response over some* $\mathbf{\Lambda}$ *is given by* $h(\mathbf{\Lambda})$ *for* $h(\cdot) \in \mathbb{H}$*. Similarly* $\mathbb{H}' := \{\tilde{h}(\cdot) \mid \forall$ *possible polynomial parameterizations of* $\tilde{h}\}$ *is set of all filters learn-able via* PP-GNN *, with* $\tilde{h} = h + h_{f_1} + h_{f_2}$*, where* h *is parameterized by a K-degree polynomial supported over entire spectrum,* h_{f_1} *and* h_{f_2} *are adaptive filters parameterized by independent* K'*-degree polynomials which only act on top and bottom* t *diagonal elements respectively, with* $t < n/2$ *and* $K' \leq K$*; then* $\mathbb{H}$ *and* $\mathbb{H}'$ *form a vector space, with* $\mathbb{H} \subset \mathbb{H}'$*. Also,* $\frac{dim(\mathbb{H}')}{dim(\mathbb{H})} = \frac{K+2K'+3}{K+1}$.

Proof. We start by constructing the abstract spaces on top of the polynomial vector space. Consider the set of all the univariate polynomials having degree at most K in the vector space over the ring $\mathbb{K}_n^x := \mathbb{K}[x_1, \ldots, x_n]$ where $\mathbb{K}$ is the field of real numbers. Partition this set into n subsets, say $V_1, \ldots, V_n$, such that for $i \in [n]$, V_i contains all polynomials of degree up to K in x_i. It is easy to see that $V_1, \ldots, V_n$ are subspaces of $\mathbb{K}[x_1, \ldots, x_n]$. Define $V = V_1 \oplus V_2 \oplus \cdots \oplus V_n$ where $\oplus$ denotes a direct sum. Define the matrix $D_i[c]$ whose $(i,i)^{\text{th}}$ entry is c and all the other entries are zero. For $i \in [n]$, define linear maps $\phi_i : V_i \to \mathbb{M}_n(\mathbb{K}_n^x)$ by $f(x_i) \mapsto D_i[f(x_i)]$. Im$(\phi_i)$ forms a vector space of all diagonal matrices, whose (i,i) entry is the an element of V_i. Generate a linear map $\phi : V \to \mathbb{M}_n(\mathbb{K}_n^x)$ by mapping $\phi(f(x_i))$ to $\phi_i(f(x_i))$ for all $i \in [n]$ as the components of the direct sum present in its argument. Note that ϕ_i for $i \in [n]$ are injective maps, making ϕ an injective map. This implies that $\mathbb{H} \subset \text{Im}(\phi)$ is a subspace with basis $\mathcal{B}_h := \{\phi(x_1^0 + \cdots + x_n^0), \phi(x_1 + \cdots + x_n), \ldots, \phi(x_1^K + \cdots + x_n^K)\}$, making $\dim(\mathbb{H}) = K + 1$. Similarly we have, $\mathbb{H}' \subset \text{Im}(\phi)$, a subspace with basis $\mathcal{B}_{h'} := \mathcal{B}_h \bigcup \{\phi(x_1^0 + \cdots + x_t^0 + 0 + \cdots + 0), \phi(x_1 + \cdots + x_t + 0 + \cdots + 0), \ldots, \phi(x_1^{K'} + \cdots + x_t^{K'} + 0 + \cdots + 0)\} \bigcup \{\phi(0 + \cdots + 0 + x_{n-t+1}^0 + \cdots + x_n^0), \phi(0 + \cdots + 0 + x_{n-t+1} + \cdots + x_n), \ldots, \phi(0 + \cdots + 0 + x_{n-t+1}^{K'} + \cdots + x_n^{K'})\}$ where x_i^0 and 0 are the corresponding multiplicative and additive identities of $\mathbb{K}_n^x$, implying $\mathbb{H} \subset \mathbb{H}'$ and $\dim(\mathbb{H}') = K + 2K' + 3$. □

Proof of Corollary 1

Corollary. *The corresponding adapted graph families* $\mathbb{G} := \{\mathbf{U}h(\cdot)\mathbf{U}^T \mid \forall h(\cdot) \in \mathbb{H}\}$ *and* $\mathbb{G}' := \{\mathbf{U}\tilde{h}(\cdot)\mathbf{U}^T \mid \forall \tilde{h}(\cdot) \in \mathbb{H}'\}$ *for any unitary matrix* $\mathbf{U}$ *form a vector space, with* $\mathbb{G} \subset \mathbb{G}'$ *and* $\frac{dim(\mathbb{G}')}{dim(\mathbb{G})} = \frac{K+2K'+3}{K+1}$.

Proof. Consider the injective linear maps $f_1, f_2 : \mathbb{M}_n(\mathbb{K}_n^x) \to \mathbb{M}_n(\mathbb{K}_n^x)$ as $f_1(\mathbf{A}) = \mathbf{U}^T\mathbf{A}$ and $f_2(\mathbf{A}) = \mathbf{A}\mathbf{U}$. Define $f_3 : \mathbb{H} \to \mathbb{M}_n(\mathbb{K}_n^x)$ and $f_4 : \mathbb{H}' \to \mathbb{M}_n(\mathbb{K}_n^x)$ as $f_3(\mathbf{A}) = (f_1 \circ f_2)(\mathbf{A})$ for $\mathbf{A} \in \mathbb{H}$ and $f_4(\mathbf{A}) = (f_1 \circ f_2)(\mathbf{A})$ for $\mathbf{A} \in \mathbb{H}'$. Since $\mathbf{U}$ is given to be a unitary matrix, f_3 and f_4 are monomorphisms. Using this with the result from Theorem 4.2, $\mathbb{H} \subset \mathbb{H}'$, we have $\mathbb{G} \subset \mathbb{G}'$. □

PPGNN mitigates oversmoothing For showing that our model mitigates oversmoothing for the higher orders, we extend a few results by [8].

Lemma 1. *Assume that the nodes in an undirected and connected graph G have one of C labels. Then, for k large enough, we have,*

$$\mathbf{H}_{:j}^k = \boldsymbol{\beta}_j \boldsymbol{\pi} + o_k(1)$$

$$(\mathbf{H}_{\sigma_i}^k)_{:j} = \begin{cases} \boldsymbol{\beta}_j \boldsymbol{\pi} + o_k(1), & \text{if } \pm 1 \in \sigma_i \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

for $j \in [C]$. *Here* $\boldsymbol{\pi}_i = \frac{\sqrt{\tilde{D}_{ii}}}{\sqrt{\sum_{v \in V} \tilde{D}_{vv}}}$ *and* $\boldsymbol{\beta}^T = \boldsymbol{\pi}^T \mathbf{H}_0$.

Proof. The first equality is a standard result. For the second equality, note that all $\mathbf{S}_{\sigma_i}$ have nullspace of dimension $n - |\sigma_i|$, and rest eigenvalues have their absolute values ≤ 1. By definition, $\hat{\mathbf{A}}$ is a doubly stochastic matrix, the stationary distribution for $\mathbf{S}_{\sigma_i}$ can only be reached if it contains an eigenvalue of absolute value 1. (easily seen that the largest eigenvalue of $\hat{\mathbf{A}}$ is 1).

□

Thus, whenever the label prediction is dominated by higher order $\mathbf{H}^k_{()}$, all nodes have a representation proportional to $\tau\boldsymbol{\beta}$, giving same label prediction for each node.

Definition 1. *(Oversmoothing). If oversmoothing occurs in PPGNN for K sufficiently large, we have $\mathbf{Z}_{:j} = c_1\boldsymbol{\beta}_j\boldsymbol{\pi},\ \forall j \in [C]$ for some $c_1 > 0$ if $\tau_k > 0$ and $\mathbf{Z}_{:j} = -c_1\boldsymbol{\beta}_j\boldsymbol{\pi},\ \forall j \in [C]$ for some $c_1 > 0$ if $\tau_k < 0$.*

Following lemma is the extended from the corresponding lemma of [8].

Lemma 2. *Let $L = \sum_{i\in\mathcal{T}} L_i = \sum_{i\in\mathcal{T}} -\log\big(\langle \mathbf{P}^T_{i:}, \mathbf{Y}^T_{i:}\rangle\big)$ be the cross-entropy loss and $\mathcal{T}$ be the training set. The gradient of τ_k for k large enough is $\frac{\partial L}{\partial \tau_k} = \sum_{i\in\mathcal{T}} \boldsymbol{\pi}_i\langle \mathbf{P}_{i:} - \mathbf{Y}_{i:}, \boldsymbol{\beta}\rangle + o_k(1)$*

Now the main result follows in same way as [8] from the above lemmas:

Theorem 3. ***(Extension of Theorem 4.2 of [8])*** *If the training set contains nodes from each of C classes, then* PP-GNN *can always avoid over-smoothing. That is, for a large enough k and for a parameter associated with a k-order term, $\tau \in [\gamma_i] \cup [\gamma_i^{(h)}] \cup [\gamma_i^{(l)}],\ i \in [K] \cup \{0\}$, we have:*

$$\frac{\partial L}{\partial \tau} = \begin{cases} \sum_{i\in\mathcal{T}} \boldsymbol{\pi}_i \left(\max_{j\in[C]} \boldsymbol{\beta}_j - \boldsymbol{\beta}_{\mathbf{1}[\mathbf{Y}_{i:}]}\right) + o_k(1), \tau > 0 \\ \sum_{i\in\mathcal{T}} \boldsymbol{\pi}_i \left(\min_{j\in[C]} \boldsymbol{\beta}_j - \boldsymbol{\beta}_{\mathbf{1}[\mathbf{Y}_{i:}]}\right) + o_k(1),\ \tau < 0 \end{cases}$$

Where, $\boldsymbol{\pi}_i = \frac{\sqrt{\tilde{D}_{ii}}}{\sqrt{\sum_{v\in V} \tilde{D}_{vv}}}$ and $\boldsymbol{\beta}^T = \boldsymbol{\pi}^T\mathbf{H}_0$. This implies that all parameters, τ and their gradients $\frac{\partial L}{\partial \tau}$ are of same sign for sufficiently high orders. Since the gradients are bounded, higher order parameters τ will approach to 0 until we escape oversmoothing.

A.4 Experiments

Datasets We evaluate on multiple benchmark datasets to show the effectiveness of our approach. Detailed statistics of the datasets used are provided in Table 6. We borrowed **Texas**, **Cornell**, **Wisconsin** from WebKB[7], where nodes represent web pages and edges denote hyperlinks between them. **Actor** is a co-occurrence network borrowed from [17], where nodes correspond to an actor, and and edge represents the co-occurrence on the same Wikipedia page. **Chameleon**, **Squirrel** are borrowed from [18]. Nodes correspond

[7] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb

to web pages and edges capture mutual links between pages. For all benchmark datasets, we use feature vectors, class labels from [3]. For datasets in (Texas, Wisconsin, Cornell, Chameleon, Squirrel, Actor), we use 10 random splits (48%/32%/20% of nodes for train/validation/test set) from [4]. We borrowed **Cora**, **Citeseer**, and **Pubmed** datasets and the corresponding train/val/test set splits from [4]. The remaining datasets were borrowed from [3]. We follow the same dataset setup mentioned in [3] to create 10 random splits for each of these datasets. We also experiment with two slightly larger datasets Flickr [20] and OGBN-arXiv [21]. We use the publicly available splits for these datasets.

Table 6: Dataset Statistics.

Properties	Texas	Wisconsin	Actor	Squirrel	Chameleon	Cornell	Flickr	Cora-Full	OGBN-arXiv	Wiki-CS	Citeseer	Pubmed	Cora	Computer	Photos
Homophily Level	0.11	0.21	0.22	0.22	0.23	0.30	0.32	0.59	0.63	0.68	0.74	0.80	0.81	0.81	0.85
#Nodes	183	251	7600	5201	2277	183	89250	19793	169343	11701	3327	19717	2708	13752	7650
#Edges	492	750	37256	222134	38328	478	989006	83214	1335586	302220	12431	108365	13264	259613	126731
#Features	1703	1703	932	2089	500	1703	500	500	128	300	3703	500	1433	767	745
#Classes	5	5	5	5	5	5	7	70	40	10	6	3	7	10	8
#Train	87	120	3648	2496	1092	87	446625	1395	90941	580	1596	9463	1192	200	160
#Val	59	80	2432	1664	729	59	22312	2049	29799	1769	1065	6310	796	300	240
#Test	37	51	1520	1041	456	37	22313	16349	48603	5487	666	3944	497	13252	7250

Baselines We provide the methods in comparison along with the hyper-parameters ranges for each model. For all the baseline models, we sweep the common hyper-parameters in the same ranges. Learning rate is swept over $[0.001, 0.003, 0.005, 0.008, 0.01]$, dropout over $[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$, weight decay over $[1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]$, and hidden dimensions over $[16, 32, 64]$. For model-specific hyper-parameters, we tune over author prescribed ranges. We use undirected graphs with symmetric normalization for all graph networks in comparison. For all models, we report test accuracy for the configuration that achieves the highest validation accuracy. We report standard deviation wherever applicable.

The hyper-parameter search space as described above, was prescribed in [47] where the authors have shown that with thorough tuning, GCN(s) thoroughly outperform several existing SOTA models. Training and observing the results of all the models on this hyper-parameter search space, while also maintaining a consistent number of Optuna Trials (per SOTA model, per dataset, per split), is a convincing indicator of the fact that our proposed method does hold merit, and is not attributed to a statistical anomaly.

LR and MLP: We trained Logistic Regression classifier and Multi Layer Perceptron on the given node features. For MLP, we limit the number of hidden layers to one.

GCN: We use the GCN implementation provided by the authors of [8].

SGCN: SGCN [16] is a spectral method that models a low pass filter and uses a linear classifier. The number of layers in SGCN is treated as a hyper-parameter and swept over $[1, 2]$.

SUPERGAT: SUPERGAT [3] is an improved graph attention model designed to also work with noisy graphs. SUPERGAT employs a link-prediction based self-supervised

task to learn attention on edges. As suggested by the authors, on datasets with homophily levels lower than 0.2 we use SUPERGAT$_{SD}$. For other datasets, we use SUPERGAT$_{MX}$. We rely on authors code for our experiments.

GEOM-GCN: GEOM-GCN [4] proposes a geometric aggregation scheme that can capture structural information of nodes in neighborhoods and also capture long range dependencies. We quote author reported numbers for Geom-GCN. We could not run Geom-GCN on other benchmark datasets because of the unavailability of a pre-processing function that is not publicly available.

H$_2$GCN: H$_2$GCN [5] proposes an architecture, specially for heterophilic settings, that incorporates three design choices: i) ego and neighbor-embedding separation, higher-order neighborhoods, and combining intermediate representations. We quote author reported numbers where available, and sweep over author prescribed hyper-parameters for reporting results on the rest datasets. We rely on author's code for our experiments.

FAGCN: FAGCN [7] adaptively aggregates different low-frequency and high-frequency signals from neighbors belonging to same and different classes to learn better node representations. We rely on author's code for our experiments.

APPNP: APPNP [10] is an improved message propagation scheme derived from personalized PageRank. APPNP's addition of probability of teleporting back to root node permits it to use more propagation steps without oversmoothing. We use GPR-GNN's implementation of APPNP for our experiments.

LINEARGCN (LGC): LINEARGCN (LGC) [22] is a spectrally grounded GCN that adapts the entire eigen spectrum of the graph to obtain better node feature representations.

GPR-GNN: GPR-GNN [8] adaptively learns weights to jointly optimize node representations and the level of information to be extracted from graph topology. We rely on author's code for our experiments.

TDGNN: TDGNN [31] is a tree decomposition method which mitigates feature smoothening and disentangles neighbourhoods in different layers. We rely on author's code for our experiments.

ARMA: ARMA [39] is a spectral method that uses K stacks of $ARMA_1$ filters in order to create an $ARMA_K$ filter (an ARMA filter of order K). Since [39] do not specify a hyperparameter range in their work, following are the ranges we have followed: GCS stacks (S): $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, stacks' depth($T$): $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. However we only select configurations such that the number of learnable parameters are less than or equal to those in PP-GNN. The input to the ARMAConv layer are the node features and the output is the number of classes. This output is then passed through a softmax layer. We use the implementation from the official PyTorch Geometric Library [8]

BernNet: BernNet [38] is a method that approximates any filter over the normalised Laplacian spectrum of a graph, by a K^{th} Order Bernstein Polynomial Approximation. We use the model specific hyper-parameters prescribed by the authors of the paper. We vary the Propagation Layer Learning Rate as follows: $[0.001, 0.002, 0.01, 0.05]$. We also vary the Propagation Layer Dropout as follows: $[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$. We rely on the authors code for our experiments.

[8] https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/arma_conv.html\#ARMAConv

AdaGNN: AdaGNN [37] is a method that captures the different importance's for varying frequency components for node representation learning. We use the model specific hyper-parameters prescribed by the authors of the paper. The No. of Layers hyper-parameter is varied as follows: $[2, 4, 8, 16, 32, 128]$. We rely on the authors code for our experiments.

UFG: UFG [32] decompose the graph into low-pass and high-pass frequencies, and define a framelet based convolutional model. We use the model specific hyper-parameters as prescribed by the authors of the paper. We rely on the authors code for our experiments.

GFIR - Unconstrained Setting: In this setting, we do not impose any regularization constraints such as dropout and L2 regularization.

GFIR - Constrained Setting: In this setting, we impose dropouts as well as L2 regularization on the GFIR model. Both dropouts and L2 regularization were applied on the H_k's (the learnable filter matrices from the above equation).

Links to the authors' codebases can be found in Table 7.

Table 7: Links to the codebases of certain baselines.

Method	Code Links	Commit ID
GCN	https://github.com/jianhao2016/GPRGNN	dc246833865af87ae5d4e965d189608f8832ddac
SuperGAT	https://github.com/dongkwan-kim/SuperGAT	2d3f44acbb10af5850aa17a3903dea955a29d2e2
H2GCN	https://github.com/GemsLab/H2GCN	08011c5199426e1c49b80ee2944d338dfd55e2b5
FAGCN	https://github.com/bdy9527/FAGCN	23bb10f6bf0b1d2e5874140cd4b266c60a7c63f3
APPNP	https://github.com/jianhao2016/GPRGNN	dc246833865af87ae5d4e965d189608f8832ddac
GPRGNN	https://github.com/jianhao2016/GPRGNN	dc246833865af87ae5d4e965d189608f8832ddac
TDGNN	https://github.com/YuWVandy/TDGNN	505b1af90255aace255744ec81a7033a5d682b90
BernNet	https://github.com/ivam-he/BernNet	7b9c1652dbe43730f52d647957761bf6d3f17425
AdaGNN	https://github.com/yushundong/AdaGNN	f178d3144921c8845027234cac68a7f0dd057fe2
UFG	https://github.com/YuGuangWang/UFG	229acd89b33f4f4e1bab2c0d92fb93d146127fd1

Implementation Details In this subsection, we present several important points that are useful for practical implementation of our proposed method and other experiments related details. Our approach is based on adaptation of a few eigen graphs constructed using eigen components. Following [1], we use a symmetric normalized version ($\tilde{\mathbf{A}}$) of adjacency matrix $\mathbf{A}$ with self-loops: $\tilde{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{D}^{-\frac{1}{2}}$ where $\tilde{D}_{ii} = 1 + D_{ii}$, $D_{ii} = \sum_j A_{ij}$ and $\tilde{D}_{ij} = 0, i \neq j$. We work with eigen matrix and eigen values of $\tilde{\mathbf{A}}$.

To reduce the learnable hyper-parameters, we separately partition the low-end and high-end eigen values into several contiguous bins and use shared filter parameters for each of these bins. The number of bins, which can be interpreted as number of filters, is swept in the range $[2, 4, 5]$. The orders of the polynomial filters are swept in the range $[2, 4, 6]$. The number of EVD components are swept in the range $[256, 1024]$. In our experiments, we set $\eta_l = \eta_h$ and we vary the η_l parameter in range $(0, 1)$ and $\eta_{gpr} = 1 - \eta_l$. The range of η_l, η_h and η_{gpr} is kept between $(0, 1)$ in order to have

a bounded and weighted contribution of every term. Since previous works either use directly or use a variant of the GPR term, these ranges make it feasible to carry out an analysis of how the term contributes to the learning of the representations and also to compare it with the contribution of the terms of the proposed model, as keeping parameters between $(0, 1)$, adding to 1 provides room for a weighted contribution of each term.

For optimization, we use the Adam optimizer [19]. We set early stopping to 200 and the maximum number of epochs to 1000. We utilize learning rate with decay, with decay factor set to 0.99 and decay frequency set to 50. All our experiments were performed on a machine with Intel Xeon 2.60GHz processor, 112GB Ram, Nvidia Tesla P-100 GPU with 16GB of memory, Python 3.6, and PyTorch 1.9.0 [24]. We used Optuna [23] and set the number of trials to 20 to optimize the hyperparameter search for PP-GNN. For other baseline models, we set the number of trials to 100.

Note: Several baselines report elevated results on some of our benchmark datasets. This difference is because of the difference in splits. We use the splits from [4]. Baselines including BERNNET, GPR-GNN evaluate on random splits with 60/20/20 distribution for train/val/test labels.

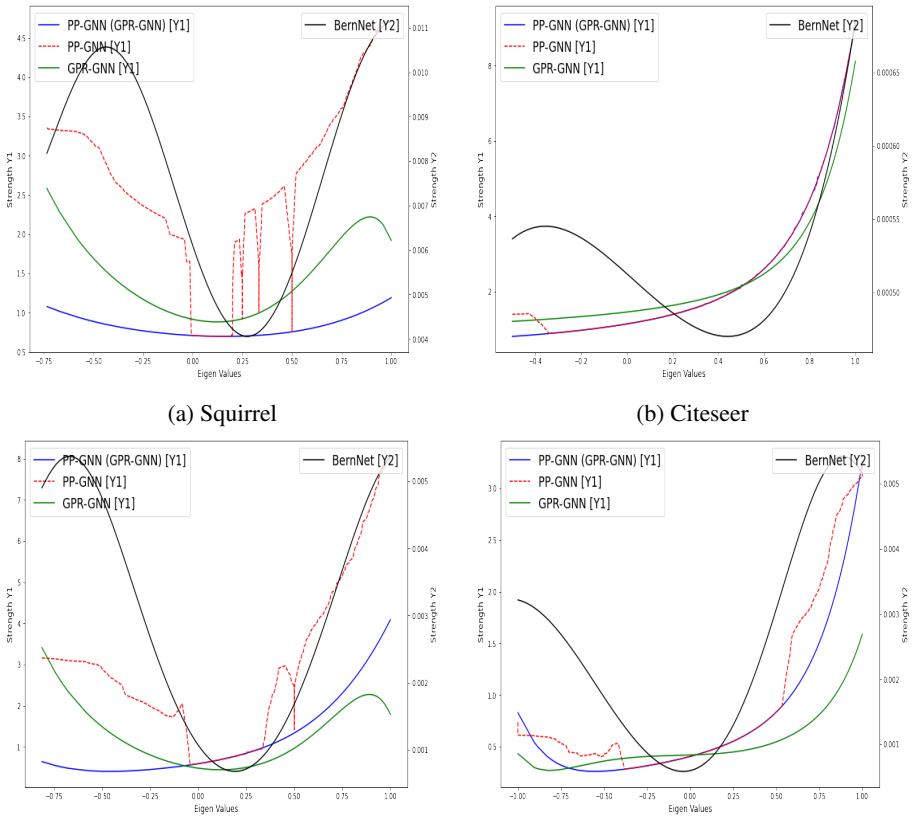
Adaptable Frequency Responses In Figure 1 of the main paper, we observe that PP-GNN learns a complicated frequency response for a heterophilic dataset (Squirrel) and a simpler frequency response for a homophilic dataset (Citeseer). We observe that this trend follows for two other datasets Chameleon (heterophilic) and Computer (homophilic). See Figure 9.

Effect of number of eigenvalues/vectors (EVs). Since the number of EVs to adapt might not be known apriori, we conducted a study to assess the effect of using different number of EVs on test performance. We report results on a few representative datasets. From Figure 10a, we see that Homophilic datasets can benefit by adapting as small as 32 eigen components. Heterophilic datasets achieve peak performance by adapting ($\sim$250-500) number of eigen components. These results indicate that the number of EVs required to get competitive/superior performance is typically small, therefore, computationally feasible and affordable.

Does the MLP even matter? In PP-GNN there is a two layered MLP (that transforms the input node features) followed by a single graph filtering layer similar to GPR-GNN.

To understand MLP's significance, we ran an additional experiment, where we have used a single linear layer, instead of the two layered MLP. We continue to observe competitive (with respect to our original PP-GNN model) performance, across most datasets. The results can be found in Table 8. Also, the two layer MLP is not the significant contributor towards performance. This can also be seen by comparing GPR-GNN's performance with that of LGC's. LGC can be interpreted as a linear version of GPR-GNN, and achieves comparable performance as GPR-GNN.

Table 8 seems to suggest that PP-GNN (Linear) is competitive to PP-GNN (Original). We can infer that adding MLP may give marginal improvements over the linear version.



(a) Squirrel (b) Citeseer

(c) Chameleon (d) Computer

Fig. 9: Learnt Frequency Responses

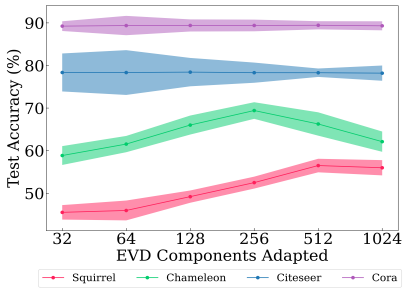
This phenomenon is also observed in GPR-GNN. To illustrate this, we can compare GPR-GNN with LGC (linear version of GPR-GNN). We can observe in Table 8 that GPR-GNN and LGC are comparable in performance.

A.5 Comparison against general FIR filters

Instead of using a polynomial filter, we can use a general FIR filter (GFIR) which is described by the following equation:

$$Z = \sum_{k=0}^{K} S^k X H_k$$

where S is the graph shift operator (which in our case is $\widetilde{A}$), X is the node feature matrix and H_k's are learnable filter matrices. One can see GCN, SGCN, GPR-GNN as special cases of this GFIR filter, which constrain the H_k in different ways.



(a) Varying No. of EVs

Fig. 10: Analyzing varying number of eigenvalues on performance

Table 8: Comparision of Linear GPR-GNN and Linear PP-GNN with respect to other pertinent baselines.

	Computers	**Chameleon**	**Citeseer**	**Cora**	**Squirrel**	**Texas**	**Wisconsin**
GPR-GNN	82.38 (1.60)	62.59 (2.04)	76.84 (1.69)	87.77 (1.31)	46.31 (2.46)	81.35 (5.32)	82.55 (6.23)
LGC	83.44 (1.77)	61.14 (2.07)	76.96 (1.73)	88.02 (1.44)	44.26 (1.49)	80.20 (4.28)	81.89 (5.98)
PP-GNN (Original)	85.23 (1.36)	69.10 (1.37)	78.25 (1.76)	89.52 (0.85)	59.15 (1.91)	89.73 (4.90)	88.24 (3.33)
PP-GNN (Linear)	84.27 (1.19)	67.88 (1.62)	77.86 (1.74)	88.43 (0.69)	55.11 (1.72)	85.58 (4.70)	86.24 (3.23)

We first demonstrate that constraint on the GFIR filter is necessary for getting improvement in performance, particularly on heterophilic datasets. Towards this, we build two versions of GFIR: one with regularization (constrained), and the other without regularization (unconstrained). We ensure that that the number of trainable parameters in these models are comparable to those used in PP-GNN. We provide further details of the versions of the GFIR models below and report the results in Table 9 below:

- **Unconstrained Setting**: In this setting, we do not impose any regularization constraints such as dropout and L2 regularization.
- **Constrained Setting**: In this setting, we impose dropouts as well as L2 regularization on the GFIR model. Both dropouts and L2 regularization were applied on the H_k's (the learnable filter matrices from the above equation).

We also compare PP-GNN (the proposed model) as well as GPR-GNN to the General FIR filter model (GFIR).

We can make the following observation from the results reported in Table 9:

- Firstly, constrained GFIR performs better than the unconstrained version, with performance lifts of up to $\sim$10%. This suggests that regularization is important for GFIR models.
- GPR-GNN outperforms the constrained GFIR version. It is to be noted that GPR-GNN further restricts the space of graphs explored as compared to GFIR. This

Table 9: Comparing PP-GNN and GPR-GNN against the GFIR filter models.

Train Acc /Test Acc	Computers	Chameleon	Citeseer	Cora	Squirrel	Texas	Wisconsin
GFIR (Unconstrained)	78.39 (1.09)	51.71 (3.11)	75.83 (1.94)	87.93 (0.90)	36.50 (1.12)	73.24 (6.91)	77.84 (3.21)
GFIR (Constrained)	79.57 (2.12)	61.27 (2.42)	76.24 (1.43)	87.46 (1.26)	41.12 (1.17)	74.59 (4.45)	79.41 (3.10)
GPR-GNN	82.38 (1.60)	62.59 (2.04)	76.84 (1.69)	87.77 (1.31)	46.31 (2.46)	81.35 (5.32)	82.55 (6.23)
PP-GNN	**85.23** (1.36)	**67.74** (2.31)	**78.25** (1.76)	**89.52** (0.85)	**56.86** (1.20)	**89.73** (4.90)	**88.24** (3.33)

suggests that regularization beyond simple L2/dropout kind of regularization (polynomial filter) is beneficial.

– PP-GNN performs better than GPR-GNN. Our model slightly expands the space of graphs explored (as compared to GPR-GNN, but lesser than GFIR), while retaining good performance. This suggests that there is still room for improvement on how regularization is done.

PP-GNN has shown one possible way to constrain the space of graphs while improving performance on several datasets, however, it remains to be seen whether there are alternative methods that can do even better. We hope to study and analyze this aspect in the future.

A.6 More details on comparison against polynomial filtering methods

More details on section 4.4 are given below:

APPNP: The node embeddings are learnt by APPNP as described below:

$$Z = \sum_{k=0}^{K} \gamma_k \tilde{A}_{sym}^k \mathbf{Z}_x(\mathbf{X}, \boldsymbol{\Theta})$$

APPNP uses fixed $\gamma_k = \alpha(1-\alpha)^k$ with $\gamma_K = (1-\alpha)^K$ where α is a hyperparameter, $\mathbf{Z}_x(\mathbf{X}, \boldsymbol{\Theta})$ are the node features transformed by MLP with parameter $\boldsymbol{\Theta}$.

GPR-GNN: The node embeddings are learnt by GPR-GNN as described below:

$$Z = \sum_{k=0}^{K} \gamma_k \tilde{A}_{sym}^k \mathbf{Z}_x(\mathbf{X}, \boldsymbol{\Theta})$$

Where $\gamma_k(\forall k)$ are now learnable parameters, $\mathbf{Z}_x(\mathbf{X}, \boldsymbol{\Theta})$ are the node features transformed by MLP with parameter $\boldsymbol{\Theta}$.

BERNNET: The node embeddings for BERNNET are learnt as described below:

$$
\begin{aligned}
Z &= \sum_{k=0}^{K} \frac{\theta_k}{2^K} \binom{K}{k} (2I - L)^{K-k} L^k \mathbf{Z}_x(\mathbf{X}, \mathbf{\Theta}) \\
&= \sum_{k=0}^{K} \frac{\theta_k}{2^K} \binom{K}{k} (A_{sym} + I)^{K-k} (I - A_{sym})^k \mathbf{Z}_x(\mathbf{X}, \mathbf{\Theta}) \\
&= \sum_{q=0}^{K} [\sum_{r=0}^{K} \frac{\theta_k}{2^K} \binom{K}{r} \sum_{p=0}^{q} \binom{K-r}{q-p} \binom{r}{p} (-1)^p] A_{sym}^q \mathbf{Z}_x(\mathbf{X}, \mathbf{\Theta}) \\
&= \sum_{q=0}^{K} [\sum_{r=0}^{K} \theta_r \alpha_{rq}] A_{sym}^q \mathbf{Z}_x(\mathbf{X}, \mathbf{\Theta}) \\
&= \sum_{q=0}^{K} w_q A_{sym}^q \mathbf{Z}_x(\mathbf{X}, \mathbf{\Theta})
\end{aligned}
$$

The following table summarizes recent key polynomial filtering based methods and a short description of the constraints/variant they employ.

Table 10: Different polynomial filtering based methods. Note that the coefficients of APPNP are fixed (not learnable) PPR coefficients ($\gamma_k \, \forall \, k$) and the coefficients of GPR-GNN ($\gamma_k \, \forall k$)and BERNNET ($\theta_k \, \forall k$) are learnable.

Method	Polynomial Basis	Filter Response	Constraints
APPNP	Monomial	$h(\lambda) = \sum_{k=0}^{K} \gamma_k \lambda^k$	$\gamma_k = \alpha(1-\alpha)^k; \gamma_K = (1-\alpha)^K$; α is a hyper-parameter
GPR-GNN	Monomial	$h(\lambda) = \sum_{k=0}^{K} \gamma_k \lambda^k$	γ_k are unconstrained
BERNNET	Bernstein	$h(\lambda) = \sum_{k=0}^{K} \frac{\theta_k}{2^K} \binom{K}{k} (2-\lambda)^{K-k} \lambda^k$	$\theta_k \geq 0$

A.7 Training Time Analysis

In the following subsections, we provide comprehensive timing analysis.

Computational Complexity: Listed below is the computational complexity for each piece in our model for a single forward pass. Notation n: number of nodes, $|E|$: the number of edges, A: symmetric normalized adjacency matrix, F: features dimensions, d: hidden layer dimension, C: number of classes, e^* denotes the cost of EVD, K: polynomial/hop order, l: number of eigenvalues/vectors in a single partition of spectrum (for implementation, we keep l same for all such intervals), m: number of partitions of a spectrum.

- MLP: $O(nFd + ndC)$
- GPR-term: $O(K|E|C) + O(nKC)$. The first term is the cost for computing $A^K f(X)$ for sparse A. The second term is the cost of summation $\sum_k A^k f(X)$.

- Excess terms for PP-GNN: $O(mnlC)$. This is obtained by the optimal matrix multiplication present in Equation 5 of the main paper ($\mathbf{U}_i$ is $n \times l$, $H_i(\gamma_i)$ is $l \times l$, $\mathbf{Z}_0()$ is $n \times C$). The additional factor m is because we have m different contiguous intervals/different polynomials. Typically n is much larger than l.
- EVD-term: e^*, the complexity for obtaining the eigenvalues/vectors of the adjacency matrix, which is usually very sparse for the observed graphs. Most publicly available solvers for this task utilize Lanczos' algorithm (which is a specific case of a more general Arnoldi iteration). However, the convergence bound of this iterative procedure depends upon the starting vectors and the underlying spectrum (particularly the ratio of the absolute difference of two largest eigenvalues to the diameter of the spectrum) [[34], [35], [36]]. Lanczos' algorithm is shown to be a practically efficient way for obtaining extreme eigenpairs for a similar and even very large systems. We use ARPACK's built-in implementation to precompute the eigenvalues/vectors for all datasets before training, thus amortizing this cost across training with different hyper-parameters configuration.

Per Component Timing Breakup: In Table 11, we provide a breakdown of cost incurred in seconds for different components of our model. Since the eigenpairs' computation is a one time cost, we amortize this cost over the total hyper-parameters configurations and report the effective training time in the last column on of Table 11.

Average Training Time: In Table 12, we report the training time averaged over 20 hyper-parameter configurations for several models. To understand the relative performance of our model with respect to GCN, we compute the relative time taken and report it in Table 13. We can observe in Table 13 that PP-GNN is $\sim 4x$ slower than GCN, $\sim 2X$ slower than GPR-GNN and BernNET, and $\sim 2X$ faster than AdaGNN. However, it is important to note that in our average training time, the time taken to compute K top and bottom eigenvalues/vectors is amortized across the number of trials

Table 11: **PP-GNN's per component timing cost**. Training Time refers to the end to end training time (without eigen decomposition) averaged across 20 trials. EVD cost refers to the time taken to obtain **x** top and bottom eigenvalues. This **x** can be found in the 'Number of EV's obtained' column. Since EVD is a one time cost, we average this cost over the total number of trials and add it to the training time. We refer to this cost as the Effective Training Time.

PP-GNN	Training Time	EVD Cost	Number of EV's obtained	Effective Training Time
Texas	11.89	0.00747	183 (All EVs)	11.89
Cornell	11.63	0.03271	183 (All EVs)	11.63
Wisconsin	12.08	0.01225	251 (All EVs)	12.08
Chameleon	21.44	3.71883	2048	21.63
Squirrel	31.38	15.8152	2048	32.17
Cora	22.46	54.3684	2048	25.18
Citeseer	20.51	56.9744	2048	23.36
Cora-Full	63.98	155.304	2048	71.75
Pubmed	52.54	256.71	2048	65.38
Computers	28.63	76.2738	2048	32.44
Photo	19.3	48.3683	2048	21.72
Flickr	161.16	304.114	2048	176.37
ArXiv	189.94	412.504	1024	210.57
WikiCS	27.92	65.4376	2048	31.19

Table 12: Training Time (in seconds) across Models

Dataset	GPR-GNN	PP-GNN	MLP	GCN	BernNet	ARMA	AdaGNN
Texas	9.27	11.89	1.08	3.46	5.59	6.00	13.97
Cornell	9.41	11.63	1.06	3.69	5.37	5.51	12.56
Wisconsin	9.67	12.08	1.07	3.42	5.69	5.36	13.57
Chameleon	14.69	21.63	2.60	6.42	12.46	7.84	28.77
Squirrel	18.94	32.17	5.04	7.52	17.82	28.87	90.36
Cora	12.90	25.18	1.95	5.94	12.25	10.67	22.15
Citeseer	10.62	23.36	3.72	4.56	9.52	19.5	35.34
Cora-Full	24.98	71.75	7.77	8.01	31.26	40.21	175.58
Pubmed	14.00	65.38	6.21	11.73	12.64	27.76	162.01
Computers	7.67	32.44	2.24	6.68	7.48	27.76	118.43
Photo	8.58	21.72	1.68	5.1	7.95	14.34	45.46
Flickr	42.64	176.37	21.00	30.4	62.11	119.3	178.7371
ArXiv	118.35	210.57	78.9	102.88	693.92	771.59	307.84
WikiCS	14.37	31.19	3.34	10.8	11.43	30.79	73.63

Table 13: Training Time of models relative to the training time of GCN

Dataset	GPR-GNN	PP-GNN	MLP	GCN	BernNet	ARMA	AdaGNN
Texas	2.68	3.44	0.31	1.00	1.62	1.73	4.04
Cornell	2.55	3.15	0.29	1.00	1.46	1.49	3.40
Wisconsin	2.83	3.53	0.31	1.00	1.66	1.57	3.97
Chameleon	2.29	3.37	0.40	1.00	1.94	1.22	4.48
Squirrel	2.52	4.28	0.67	1.00	2.37	3.84	12.02
Cora	2.17	4.24	0.33	1.00	2.06	1.80	3.73
Citeseer	2.33	5.12	0.82	1.00	2.09	4.28	7.75
Cora-Full	3.12	8.96	0.97	1.00	3.90	5.02	21.92
Pubmed	1.19	5.57	0.53	1.00	1.08	2.37	13.81
Computers	1.15	4.86	0.34	1.00	1.12	4.16	17.73
Photo	1.68	4.26	0.33	1.00	1.56	2.81	8.91
Flickr	1.40	5.80	0.69	1.00	2.04	3.92	5.88
ArXiv	1.15	2.05	0.77	1.00	6.74	7.50	2.99
WikiCS	1.33	2.89	0.31	1.00	1.06	2.85	6.82
Average	**2.03**	**4.39**	**0.50**	**1.00**	**2.19**	**3.18**	**8.39**

Table 14: End to end training time (in HH:MM:SS) for optimizing over 20 hyper-parameter configurations

Dataset	Chameleon	Citeseer	Computers	Cora	Cora-Full	Photo	Pubmed	Squirrel	Texas	Wisconsin	OGBN-ArXiv
Time	00:03:46	00:10:17	00:34:37	00:05:24	00:59:29	00:10:31	00:57:40	00:10:38	00:02:27	00:02:33	01:03:20