A PIECE-WISE POLYNOMIAL FILTERING APPROACH FOR GRAPH NEURAL NETWORKS

Anonymous authors

Paper under double-blind review

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, in several practical instances, 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 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. Our code is available at `https://tinyurl.com/PPGNN2022`.

1 INTRODUCTION

Graph Neural Networks (GNNs) have shown impressive performance on a wide range of graph learning tasks, including node classification (Kipf & Welling, 2017; Wu et al., 2019; Veličković et al., 2018). Several popular GNNs rely on the homophily assumption that connected nodes share same labels. However, these approaches perform poorly on heterophilic graphs, where connected nodes share different labels. To address this, a multitude of approaches were proposed that use modified aggregation scheme (Pei et al., 2020; Kim & Oh, 2021; Zhu et al., 2020), or the label-label compatibility matrix (Zhu et al., 2021).

Recent methods (Klicpera et al., 2019; Bo et al., 2021; Bianchi et al., 2021; Chien et al., 2021; He et al., 2021; Dong et al., 2021; Zheng et al., 2021) tackle Heterophily from a graph filter learning perspective. These approaches employ different filter designs to implicitly adapt the eigenspectrum to exploit signals from different frequencies, thereby leading to improved task performance. Among these, (Chien et al., 2021) proposed to learn polynomial filters utilizing monomial basis; (He et al., 2021) proposed to learn polynomial filters with positive frequency response, which they achieve by using Bernstein basis. Though these models can learn better filters and give good performance gains, we find them ineffective at learning more rich and complex frequency responses, which require higher order polynomials. One key reason for their inability to learn effective higher-order polynomials is that they only *mitigate* the over-smoothing problem. There have been some efforts in addressing this issue by using infinite impulse response (IIR) filters using auto-regressive moving average (Bianchi et al., 2021).

Our goal is to learn an effective polynomial filter for the node classification task in a transductive setting that can both mitigate the oversmoothing problem and also be capable of learning complex filter responses. Towards this, we propose PP-GNN, a novel piece-wise polynomial filtering approach to learn a filter bank, where each filter (low-order polynomial) acts on a subset of the eigenspectrum. The proposed approach relies on computing few extremal eigenpairs', making it computationally feasible and offers significant performance improvements of up to $\sim 6\%$ over previous state-of-the-art

approaches. We also show that PP-GNN's solution is more expressive and is capable of modeling richer and complex frequency responses.

2 PIECE-WISE POLYNOMIAL FILTERS

We focus on the problem of semi-supervised node classification (transductive setting) on a simple graph of n nodes, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ is the set of vertices and edges respectively. $\mathbf{A} \in \{0,1\}^{n \times n}$ denotes the adjacency matrix of $\mathcal{G}$. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the matrix of d-dimensional features for all the nodes and $\mathcal{Y}$ is the set of all possible class labels. Given a set of training nodes $\mathcal{D} \subset \mathcal{V}$ with known labels, 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 eigendecomposition of $\widetilde{\mathbf{A}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$.

Polynomial Filters: The spectral convolution of $\mathbf{X}$ on the graph $\mathbf{A}$ can be defined via the reference operator $\widetilde{\mathbf{A}}$ and a filter function h operating on the eigenvalues, in the Fourier domain (Tremblay et al., 2017; Chien et al., 2021) as,

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

where $H(\mathbf{\Lambda}) = \text{diag}(h(\mathbf{\Lambda}))$. The equality follows when the filter h takes a polynomial form, i.e., $h(\lambda) := \sum_{i=0}^{k} \alpha_i \lambda^i$ with $[\alpha_i]$ as coefficients and k as the order. It is well-known that polynomial filters can approximate any graph filter (Shuman et al., 2013; Tremblay et al., 2017), however, several practical challenges arise while learning a good polynomial filter. It can be observed in Figure 1 and Appendix A.4.5 that several datasets often require a complex spectral filter to obtain a superior performance, which in turn requires the order k of the polynomial to be very large. A large order results in a uniform convergence of the term $\widetilde{\mathbf{A}}^k\mathbf{X}$, making the node features indistinguishable (aka oversmoothing). Empirical demonstration of oversmoothing and its effect on performance can be found in the Appendix in figures 2,3 and figure 4 respectively. Chien et al. (2021) show that higher order coefficients in (1) converge to zero during training. While this phenomenon diminishes contributions of corresponding terms and helps to mitigate oversmoothing, it limits the model's capability to learn complex filters that require higher order polynomials. To address this ineffectiveness, we propose a novel piece-wise polynomial filtering approach.

Piece-wise Polynomial Filters: Instead of using a single polynomial for learning a filter across the entire spectrum, we propose to partition the eigen spectrum to several subsets, and use different (low degree) polynomials over each partition to learn a bank of filters. Let $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ denote a partition set with m contiguous intervals and $h_{i,k_i}(\lambda; \gamma_i)$ denote a k_i-degree polynomial filter function of the interval σ_i with polynomial coefficients γ_i (The number and size of partitions; polynomial orders are hyperparameters). Note k_i is a low-degree. We define piece-wise polynomial GNN (PP-GNN) filter function as:

$$h(\lambda) = \sum_{\sigma_i \in \mathcal{S}} h_{i,k_i}(\lambda; \gamma_i) \tag{2}$$

and learn a smooth filter function by imposing additional constraints to maintain continuity between polynomials of contiguous intervals at different endpoints (*aka* knots). Given the filter function, we compute the PP-GNN node embedding matrix as:

$$\mathbf{Z} = \sum_{\sigma_i \in \mathcal{S}} \mathbf{U}_i H_i(\gamma_\mathbf{i}) \mathbf{U}_i^T \mathbf{Z}_x(\mathbf{X}; \mathbf{\Theta}) \tag{3}$$

where $\mathbf{U}_i$ is a matrix with eigenvectors corresponding to eigenvalues that lie in σ_i and $H_i(\gamma_i)$ is the diagonal matrix with diagonals containing the h_i evaluated at the eigenvalues; $\mathbf{Z}_x(\mathbf{X}; \mathbf{\Theta})$ is an MLP network with parameters $\mathbf{\Theta}$. Obtaining this filter's response requires eigendecomposition which is prohibitive for large graphs. However, if the above partition only consists of a few subsets containing the spectrum's extremes, then one can resort to several efficient algorithms to obtain extremal eigenpairs for large matrices (Davidson & Thompson, 1993). This also follows from recent

investigations (eg. Chien et al. (2021)) that show that graph filters that amplify/attenuate low and high-frequency components of signals (i.e., low-pass and high-pass filters) are critical in improving performance on several benchmark datasets of varying homophily. To extract signals from the remaining (middle) portion of the spectrum, we also define a polynomial filter acting on the entire spectrum (like GPR-GNN). Thus an efficient variant of (2), which we henceforth denote as PP-GNN becomes (see Section A.2.4 of appendix for more implementation details):

$$\tilde{h}(\lambda) = \eta_l \sum_{\sigma_i \in \mathcal{S}^l} h_i^{(l)}(\lambda; \gamma_i^{(l)}) + \eta_h \sum_{\sigma_i \in \mathcal{S}^h} h_i^{(h)}(\lambda; \gamma_i^{(h)}) + \eta_{gpr} h_{gpr}(\lambda; \gamma) \tag{4}$$

where $\mathcal{S}^l$ consists of partitions over low-frequency components, $\mathcal{S}^h$ consists of partitions over high-frequency components. The third term is the filter function used by GPR-GNN which is efficiently computed. $\eta_l, \eta_h, \eta_{gpr}$ are hyperparameters to control contributions of each term. Substituting (4) in (3) results in a linear combination of the outputs of the bank of filters. The extremal eigendecomposition step is one-time cost per dataset and is amortized over several rounds of hyperparameter optimization. Such piece-wise filter allows us to approximate a complex frequency response by using small-degree polynomials, hence we restrict the polynomial order to be a lower value (see A.4.4), which in turn prevents the model from running into over-smoothing problem. Thus, our proposed model offers richer capability and flexibility to learn complex frequency response and balance computation costs over GPR-GNN.

Model Training: Like GPR-GNN, we apply SOFTMAX activation function on (3) and use the standard cross-entropy loss to learn the sets of polynomial coefficients (γ) and MLP parameters ($\mathbf{\Theta}$) using labeled data. We use validation accuracy for model selection.

Analysis: We present formal proofs to demonstrate: (a) Superior capabilities of our model at approximating arbitrary filters compared to a standard polynomial filter; (b) The new space of filters that our model learns from induces a controllable bias towards certain parts of the spectrum. This new space has a dimension of the same order as the other polynomial filtering approaches. These theorems along with their proofs can be found in sections A.3.2, A.3.3, A.3.4.

Our model formulation is a generalization of the formulation by Chien et al. (2021). We show in Section A.3.4 that our model inherits their property of mitigating oversmoothing effects when using a high degree polynomial. Our experiments show that we can obtain superior performance without depending on the higher-order polynomials.

3 EXPERIMENTS

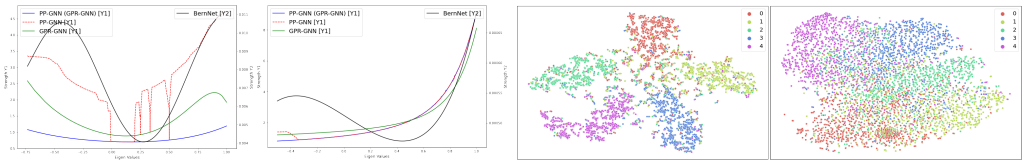
We conducted several experiments to answer the following questions: **RQ1:** How well does PP-GNN perform in comparison to other SOTA polynomial filtering methods? **RQ2:** How does the frequency response of PP-GNN look like? **RQ3:** How does the training time of PP-GNN compare with other methods? We first provide details on datasets and baselines.

Datasets: We evaluate our model on several real-world heterophilic and homophilic datasets. The heterophilic datasets include **Texas**[1], **Wisconsin**[1], **Chameleon**, **Squirrel** (Rozemberczki et al., 2021) and **Flickr**. The homophilic datasets include **Ogbn-Arxiv**, **Wiki-CS**, **Citeseer**, **Pubmed**, **Cora**, **Computer**, and **Photos** borrowed from Kim & Oh (2021). Please refer to A.4.1 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 against the following baselines: FAGCN (Bo et al., 2021), APPNP (Klicpera et al., 2019), GPR-GNN (Chien et al., 2021), LGC (Navarin et al., 2020b), BernNet (He et al., 2021), ARMA (Bianchi et al., 2021). More details regarding the baselines as well as a comprehensive comparison against other baselines (including recent state-of-the-art models) is resorted to A.4.2 and A.4.3 respectively.

RQ1: From Table 1, we can observe that while PP-GNN outperforms other polynomial filtering models on almost all datasets, we observe most improvements on heterophilic datasets ($\sim 6-7\%$

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



(a) Squirrel (b) Citeseer (c) PP-GNNon Squirrel (d) BERNNET on Squirrel

Figure 1: 1a and 1b show the learned filter responses of PP-GNN,GPR-GNN, and BERNNET. 1c and 1d show the t-SNE plots of learned embeddings.

improvements on Texas, Squirrel and Chameleon). These improvements can be attributed due to the effective polynomial filter that PP-GNN learns, which will be further discussed in RQ2.

	Texas	Wisconsin	Squirrel	Chameleon	Flickr	OGBN-ArXiv	Wiki-CS	Citeseer	Pubmed	Cora	Computer	Photos
FAGCN	82.43 (6.89)	82.94 (7.95)	42.59 (0.79)	55.22 (3.19)	OOM	OOM	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	81.89 (5.85)	**85.49 (4.45)**	39.15 (1.88)	47.79 (2.35)	50.33	69.20	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	80.20 (4.28)	81.89 (5.98)	44.26 (1.49)	61.14 (2.07)	51.67	**69.64**	**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	81.35 (5.32)	82.55 (6.23)	46.31 (2.46)	**62.59 (2.04)**	52.74	68.44	79.68 (0.50)	76.84 (1.69)	89.08 (0.39)	87.77 (1.31)	82.38 (1.60)	91.43 (0.89)
BernNET	**83.24 (6.47)**	84.90 (4.53)	**52.56 (1.69)**	62.02 (2.28)	52.35	69.21	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	79.46 (3.65)	82.75 (3.56)	47.37 (1.63)	60.24 (2.19)	**53.79**	69.49	78.94 (0.32)	**78.15 (0.74)**	88.73 (0.52)	87.37 (1.14)	78.55 (2.62)	90.26 (0.48)
PP-GNN	**89.73 (4.90)**	**88.24 (3.33)**	**59.15 (1.91)**	**69.10 (1.37)**	**55.30**	**69.28**	**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 1: Results on multiple datasets. For additional baselines see A.4.5

RQ2: 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.4.5. We also qualitatively assess the difference in the learned embedding of PP-GNN and BERNNET. Towards this, we generated t-SNE plots of the learned node embeddings and visually inspected them. From Figure 1c and 1d, we observe that PP-GNN discovers more discriminative features resulting in discernible clusters on the Squirrel dataset compared to BERNNET, enabling PP-GNN to achieve significantly improved performance.

RQ3: We conducted a comprehensive training time evaluation study to compare the running-time performance of various models on diverse datasets. 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-to-end training time comparison results show that PP-GNN is $\sim$2x slower than GPR-GNN and BERNNET. Please refer to A.4.7 for more details.

4 CONCLUSION

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 spectrum using multiple polynomial filters. Our experiments demonstrate that the proposed approach can learn effective filter functions that improve node classification accuracy significantly across diverse graphs.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *ArXiv*, abs/1907.10902, 2019.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. ISSN 1939-3539. doi: 10.1109/tpami.2021.3054830. URL http://dx.doi.org/10.1109/TPAMI.2021.3054830.

Deyu Bo, X. Wang, Chuan Shi, and Hua-Wei Shen. Beyond low-frequency information in graph convolutional networks. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations (ICLR)*, 2021.

Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009.

Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Society for Industrial and Applied Mathematics, 2002. doi: 10.1137/1.9780898719192. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898719192.

Ernest R. Davidson and William J. Thompson. Monster matrices: Their eigenvalues and eigenvectors. *Computers in Physics*, 7(5):519–522, 1993. doi: 10.1063/1.4823212. URL https://aip.scitation.org/doi/abs/10.1063/1.4823212.

Yushun Dong, Kaize Ding, Brian Jalaian, Shuiwang Ji, and Jundong Li. Graph neural networks with adaptive frequency response filter, 2021.

Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation, 2021.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pp. 271–279, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136803.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations (ICLR)*, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations (ICLR)*, 2019.

REN-CANG LI. Sharpness in rates of convergence for the symmetric lanczos method. *Mathematics of Computation*, 79(269):419–435, 2010. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/40590409.

Nicolò Navarin, Wolfgang Erb, Luca Pasa, and Alessandro Sperduti. Linear graph convolutional networks. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pp. 151–156, 2020a. URL https://www.esann.org/sites/default/files/proceedings/2020/ES2020-96.pdf.

Nicolò Navarin, Wolfgang Erb, Luca Pasa, and Alessandro Sperduti. Linear graph convolutional networks. In *ESANN*, pp. 151–156, 2020b. URL https://www.esann.org/sites/default/files/proceedings/2020/ES2020-96.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 2021.

Y. Saad. On the rates of convergence of the lanczos and the block-lanczos methods. *SIAM Journal on Numerical Analysis*, 17(5):687–706, 1980. doi: 10.1137/0717059. URL `https://doi.org/10.1137/0717059`.

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

Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. Design of graph filters and filterbanks, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Yu Wang and Tyler Derr. Tree decomposed graph neural network. In *Conference on Information and Knowledge Management*, 2021.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2019.

Xuebin Zheng, Bingxin Zhou, Junbin Gao, Yuguang Wang, Pietro Lió, Ming Li, and Guido Montufar. How framelets enhance graph neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12761–12771. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/zheng21c.html`.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Jiong Zhu, Ryan A. Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K. Ahmed, and Danai Koutra. Graph Neural Networks with Heterophily. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021.

A Appendix

The appendix is structured as follows. In Section A.1, we provide a URL for our code and data, along with some information about the contents in the URL. In Section A.2, we present additional evidence of the limitations of GPR-GNN and a representative experiment to further motivate Section 2. In Section A.3, we provide proofs for theorems, propositions and corollaries defined in Section 2. In Section A.4, we provide more details regarding the baselines, datasets and their respective splits.

We also provide implementation details and rank PP-GNN against current SoTA. We additionally provide details and results of additional experiments. In Section A.4.7, we provide a comprehensive timing analysis.

A.1 Code and Dataset

Our code, along with the datasets and their respective splits, is available at `https://tinyurl.com/PPGNN2022`. A `README` file is available explaining how the code can be executed on various datasets. The `requirements.txt` file lists all the necessary python packages required to execute the code. The `src` folder also contains bash scripts to run the code on various datasets.

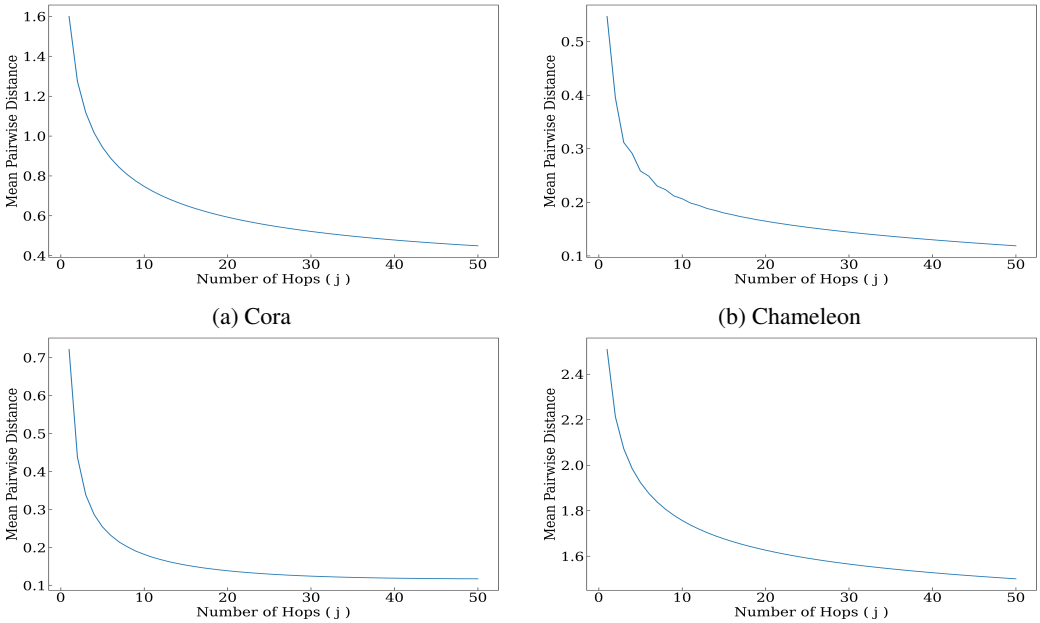
A.2 PP-GNN Motivation

Several proposed polynomial based filtering methods (e.g. APPNP, GPR-GNN, BernNet) run into the over-smoothing issue when using high-order terms. While some polynomial filtering methods (e.g. GPR-GNN and BernNet) mitigate it through slowly zeroing out the coefficients of higher-order terms while training, this makes them unable to model complex filter response which would require high-degree polynomials in order to be well approximated. The presence of the over-smoothing phenomenon is evidenced via experiments demonstrated in Section A.2.1. We further illustrate the effects of the phenomenon over the performance of GPR-GNN in Section A.2.2.

PP-GNN approximates the filter responses via fitting piece-wise low-order polynomials in different subsets of the spectrum. To demonstrate the effectiveness of this way of approximation, we conduct a toy experiment in Section A.2.3. The formulation to obtain final node embeddings for PP-GNN is illustrated in section A.2.4 along with a different perspective of the model.

A.2.1 Node Feature Indistinguishably Plots

In Figure 2, we plot the average of pairwise distances between node features for four datasets: Cora, Citeseer, Chameleon and Squirrel, after computing $\widetilde{\mathbf{A}}^j X$ for increasing j values, and observe that the mean pairwise node feature distance decreases as j increases.

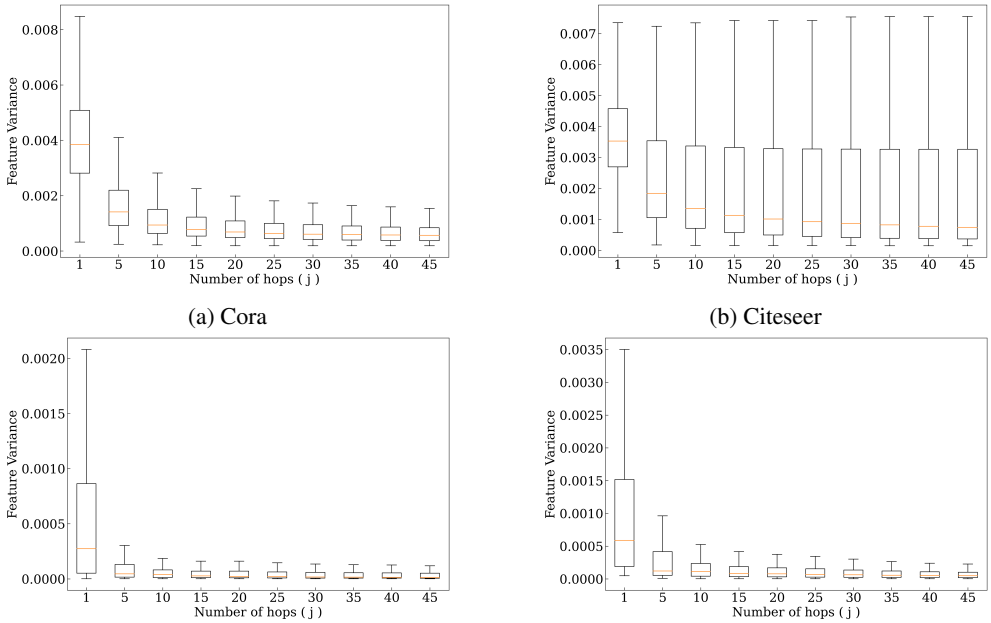


(a) Cora (b) Chameleon

(c) Squirrel (d) Citeseer

Figure 2: 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 3.



(a) Cora (b) Citeseer

(c) Squirrel (d) Chameleon

Figure 3: Variance of each dimension of node features

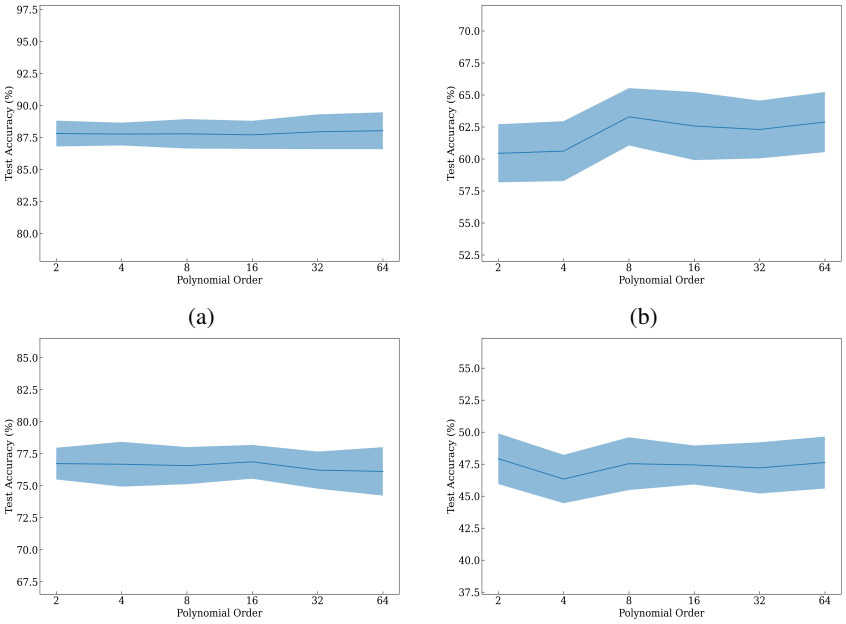
A.2.2 Effect of varying the order of the GPR-GNN Polynomial

In Figure 4, we plot the test accuracies of the GPR-GNN model while increasing the order of the polynomials for four datasets: Cora, Chameleon, Squirrel and Citeseer. We observe that on increasing the polynomial order, the accuracies do not increase any further.

In Section 2 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-GNN polynomial, neither does the test accuracy increase nor does the waveform become as complicated as PP-GNN. See Figure 5. Note: We perform these experiments with GPR-GNN as all other constrained polynomial filters can be shown as special cases of GPR-GNN.

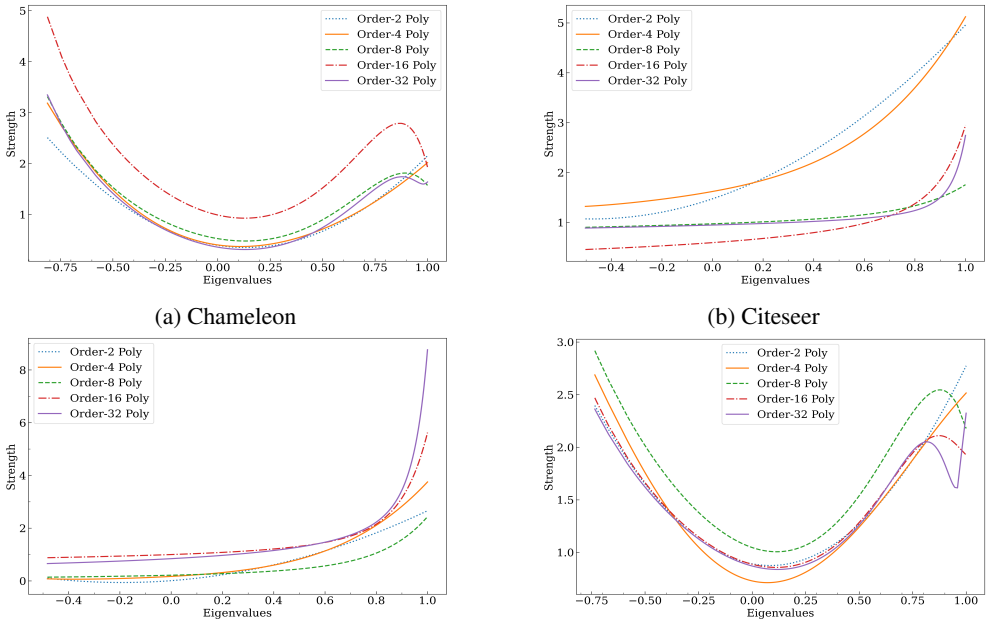
A.2.3 Fictitious Polynomial

In Section 2, 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 6. 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) (b) (c) Citeseer (d) Squirrel

Figure 4: Accuracy of the GPR-GNN model on increasing the order of the polynomial

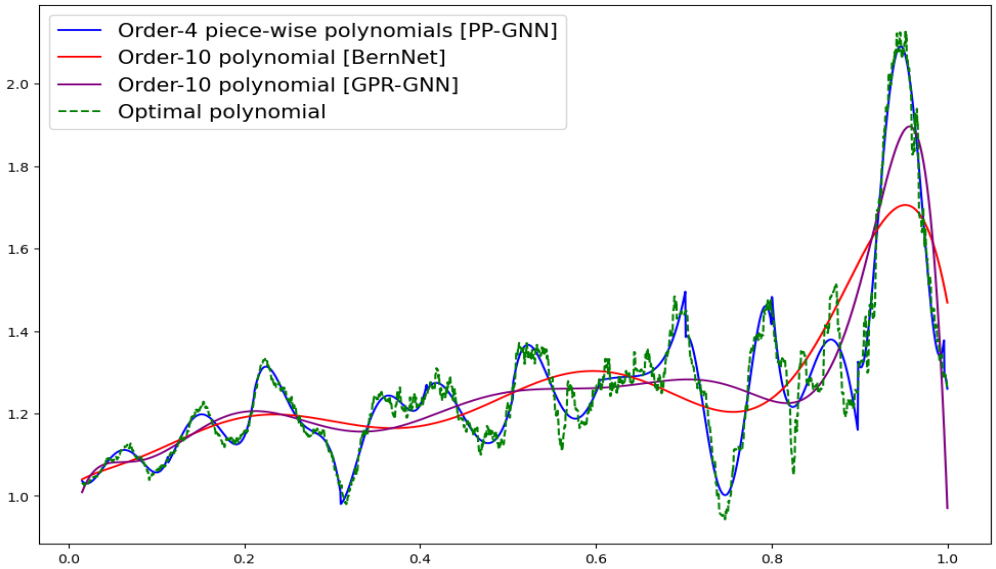


(a) Chameleon (b) Citeseer (c) Cora (d) Squirrel

Figure 5: Varying Polynomial Order in GPR-GNN

A.2.4 Implementing the Filter in Practice

We provide more details to explain the filtering operation. An Equation similar to Equation 1 can be derived for our model by substituting Equation 4 from the paper into Equation 3. On substitution, we get:



(a) Fitting single 10-degree polynomial

Figure 6: 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

$$Z = \eta_l \sum_{\sigma_i \in S^l} \sum_{j=1}^{k_1} \gamma_{ij}^{(l)} U_{\sigma_i}^{(l)} \left(\Lambda_{\sigma_i}^{(l)}\right)^j U_{\sigma_i}^{(l)T} Z_0(X;\theta) + \eta_h \sum_{\sigma_i \in S^h} \sum_{j=1}^{k_2} \gamma_{ij}^{(h)} U_{\sigma_i}^{(h)} \left(\Lambda_{\sigma_i}^{(h)}\right)^j U_{\sigma_i}^{(h)T} Z_0(X;\theta) + \eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta) \quad (5)$$

where

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

$U_{(\sigma_i)}^{(l)}$, $\Lambda_{(\sigma_i)}^{(l)}$ are the matrices containing eigenvectors and eigenvalues corresponding to the partition σ_i of the low frequency components and $U_{(\sigma_i)}^{(h)}$, $\Lambda_{(\sigma_i)}^{(h)}$ 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 5 can be

rewritten as:

$$
\begin{aligned}
Z = \eta_l \sum_{\sigma_i \in S^l} U^{(l)}_{\sigma_i} H^{(l)}_{\sigma_i} U^{(l)T}_{\sigma_i} Z_0(X;\theta) \\
+ \eta_h \sum_{\sigma_i \in S^h} U^{(h)}_{\sigma_i} H^{(h)}_{\sigma_i} U^{(h)T}_{\sigma_i} Z_0(X;\theta) \\
+ \eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta)
\end{aligned} \tag{6}
$$

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 6.

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:

$$
\begin{aligned}
\eta_{gpr} \sum_{j=1}^{k_3} \gamma_j \widetilde{A}^j Z_0(X;\theta) = \eta_{gpr} (\sum_{\sigma_i \in S^l} U^{(l)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(l)T}_{\sigma_i} \\
+ \sum_{\sigma_i \in S^h} U^{(h)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(h)T}_{\sigma_i} \\
+ \sum_{\sigma_i \in S^{mid}} U^{(mid)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(mid)T}_{\sigma_i}) Z_0(X;\theta)
\end{aligned} \tag{7}
$$

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:

$$
\begin{aligned}
Z = \sum_{\sigma_i \in S^l} U^{(l)}_{\sigma_i} (\eta_l H^{(l)}_{\sigma_i} + \eta_{gpr} H^{(gpr)}_{\sigma_i}) U^{(l)T}_{\sigma_i} Z_0(X;\theta) \\
+ \sum_{\sigma_i \in S^h} U^{(h)}_{\sigma_i} (\eta_h H^{(h)}_{\sigma_i} + \eta_{gpr} H^{(gpr)}_{\sigma_i}) U^{(h)T}_{\sigma_i} Z_0(X;\theta) \\
+ \eta_{gpr} \sum_{\sigma_i \in S^{mid}} U^{(mid)}_{\sigma_i} H^{(gpr)}_{\sigma_i} U^{(mid)T}_{\sigma_i} Z_0(X;\theta)
\end{aligned} \tag{8}
$$

From equation 8, 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 8 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.

A.3 Theoretical Results

A.3.1 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 $f_1(g(\mathbf{v_1})) + \ldots + f_m(g(\mathbf{v_m}))$

A.3.2 Piecewise Polynomials are better Approximations

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. $|\sigma_{f_i}| > K, \ \ \forall i \in [m]$, 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:

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

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

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

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:

$$\gamma = \mathbf{V}^\dagger_L h^*_L = \gamma^*_g \qquad \gamma_i = \mathbf{V}^\dagger_i h^*_i - \mathbf{V}^\dagger_L h^*_L = \gamma^*_i, \ \ \forall i \in [m]$$

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,

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

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:

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

□

A.3.3 Description of the space of filters learnt by PP-GNN

Theorem. Define $\mathbb{H} := \{h(\cdot) \mid \forall \text{ possible K-degree polynomial parameterizations of } h\}$ 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 $\mathbb{H}' := \{\tilde{h}(\cdot) \mid \forall \text{ possible polynomial parameterizations of } \tilde{h}\}$ 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, $\mathbb{H}' \subset \mathrm{Im}(\phi)$, 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 $\dim(\mathbb{H}') = K+2K'+3$. □

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 $\mathbb{G} \subset \mathbb{G}'$. □

A.3.4 PP-GNN MITIGATES OVERSMOOTHING

For showing that our model mitigates oversmoothing for the higher orders, we extend a few results by Chien et al. (2021).

Lemma A.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 A.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 Chien et al. (2021).

Lemma A.2. Let $L = \sum_{i\in\mathcal{T}} L_i = \sum_{i\in\mathcal{T}} -\log\big(\langle \mathbf{P}_{i:}^T, \mathbf{Y}_{i:}^T\rangle\big)$ 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 Chien et al. (2021) from the above lemmas:

Theorem A.3. (**Extension of Theorem 4.2 of Chien et al. (2021)**) 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

A.4.1 DATASETS

We evaluate on multiple benchmark datasets to show the effectiveness of our approach. Detailed statistics of the datasets used are provided in Table 2. We borrowed **Texas**, **Cornell**, **Wisconsin** from WebKB[2], where nodes represent web pages and edges denote hyperlinks between them. **Actor** is a co-occurrence network borrowed from Tang et al. (2009), where nodes correspond to an actor, and and edge represents the co-occurrence on the same Wikipedia page. **Chameleon**, **Squirrel** are borrowed from Rozemberczki et al. (2021). Nodes correspond to web pages and edges capture mutual links between pages. For all benchmark datasets, we use feature vectors, class labels from Kim & Oh (2021). 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 Pei et al. (2020). We borrowed **Cora**, **Citeseer**, and **Pubmed** datasets and the corresponding train/val/test set splits from Pei et al. (2020). The remaining datasets were borrowed from Kim & Oh (2021). We follow the same dataset setup mentioned in Kim & Oh (2021) to create 10 random splits for each of these datasets. We also experiment with two slightly larger datasets Flickr Chua et al. (July 8-10, 2009) and OGBN-arXiv Hu et al. (2020). We use the publicly available splits for these datasets.

Table 2: 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

A.4.2 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.

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 Chien et al. (2021).

SGCN: SGCN (Wu et al., 2019) 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 (Kim & Oh, 2021) 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 $\text{SUPERGAT}_{\text{SD}}$. For other datasets, we use $\text{SUPERGAT}_{\text{MX}}$. We rely on authors code for our experiments.

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

GEOM-GCN: GEOM-GCN (Pei et al., 2020) 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_2GCN: H_2GCN (Zhu et al., 2020) 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 (Bo et al., 2021) 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 (Klicpera et al., 2019) 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) (Navarin et al., 2020a) is a spectrally grounded GCN that adapts the entire eigen spectrum of the graph to obtain better node feature representations.

GPR-GNN: GPR-GNN (Chien et al., 2021) 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 (Wang & Derr, 2021) 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 (Bianchi et al., 2021) 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 (Bianchi et al., 2021) do not specify a hyperparameter range in their work, following are the ranges we have followed: Graph Convolutional Skip (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 [3]

BernNet: BernNet (He et al., 2021) is a method that approximates any filter over the normalised Laplacian spectrum of a graph, by a Kth order Bernstein polynomial approximation. We use the model specific hyper-parameters prescribed by the authors of the paper. We vary the learning rate of the propagation layer as as follows: $\{0.001, 0.002, 0.01, 0.05\}$. We also vary the dropout of the propagation layer 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.

AdaGNN: AdaGNN (Dong et al., 2021) 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 number of layers are varied as follows: $\{2, 4, 8, 16, 32, 128\}$. We rely on the authors code for our experiments.

UFG: UFG (Zheng et al., 2021) 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.

Links to the authors' codebases can be found in Table 3.

A.4.3 PP-GNN VERSUS SOTA MODELS

In this section we compare PP-GNN with all the baselines described in A.4.2.

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

Table 3: 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

Heterophilic Datasets. We perform comprehensive experiments to show the effectiveness of PP-GNN on several Heterophilic graphs and tabulate the results in Table 4. 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 4. 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 5. 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 [4] numbers. These differences are because we turn off the optimization tricks like Batch Normalization.

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 (Pei et al., 2020). Baselines including BERNNET, GPR-GNN evaluate on random splits with 60/20/20 distribution for train/val/test labels. Additional information regarding our experimental setup can be found in A.4.4.

A.4.4 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 Kipf & Welling (2017), we use a symmetric normalized version ($\tilde{\mathbf{A}}$) of adjacency matrix $\mathbf{A}$ with self-loops: $\tilde{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\tilde{D}^{-\frac{1}{2}}$

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

Table 4: Results on Heterophilic Datasets. '*' indicates that the results were borrowed from the corresponding papers.

	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 (Wu et al., 2019)	62.43 (4.43)	55.69 (3.53)	45.72 (1.55)	60.77 (2.11)	62.43 (4.90)
GCN (Kipf & Welling, 2017)	61.62 (6.14)	58.82 (4.89)	47.78 (2.13)	62.83 (1.52)	62.97 (5.41)
SuperGAT (Kim & Oh, 2021)	61.08 (4.97)	56.47 (3.90)	31.84 (1.26)	43.22 (1.71)	57.30 (8.53)
Geom-GCN (Pei et al., 2020)	67.57*	64.12*	38.14*	60.90*	60.81*
H2GCN (Zhu et al., 2020)	**84.86 (6.77)***	**86.67 (4.69)***	37.90 (2.02)*	58.40 (2.77)	82.16 (4.80)*
TDGNN (Wang & Derr, 2021)	83.00 (4.50)*	85.57 (3.78)*	43.84 (2.16)	55.20 (2.30)	82.92 (6.61)*
FAGCN (Bo et al., 2021)	82.43 (6.89)	82.94 (7.95)	42.59 (0.79)	55.22 (3.19)	79.19 (9.79)
APPNP (Klicpera et al., 2019)	81.89 (5.85)	85.49 (4.45)	39.15 (1.88)	47.79 (2.35)	81.89 (6.25)
LGC (Navarin et al., 2020a)	80.20 (4.28)	81.89 (5.98)	44.26 (1.49)	61.14 (2.07)	74.59 (3.42)
GPR-GNN (Chien et al., 2021)	81.35 (5.32)	82.55 (6.23)	46.31 (2.46)	62.59 (2.04)	78.11 (6.55)
AdaGNN (Dong et al., 2021)	71.08 (8.55)	77.70 (4.91)	**53.50 (0.96)**	**65.45 (1.17)**	71.08 (8.36)
BernNET (He et al., 2021)	83.24 (6.47)	84.90 (4.53)	52.56 (1.69)	62.02 (2.28)	80.27 (5.41)
ARMA (Bianchi et al., 2021)	79.46 (3.65)	82.75 (3.56)	47.37 (1.63)	60.24 (2.19)	80.27 (7.76)
UFG-ConvR (Zheng et al., 2021)	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 5: 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)**
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 6: 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**

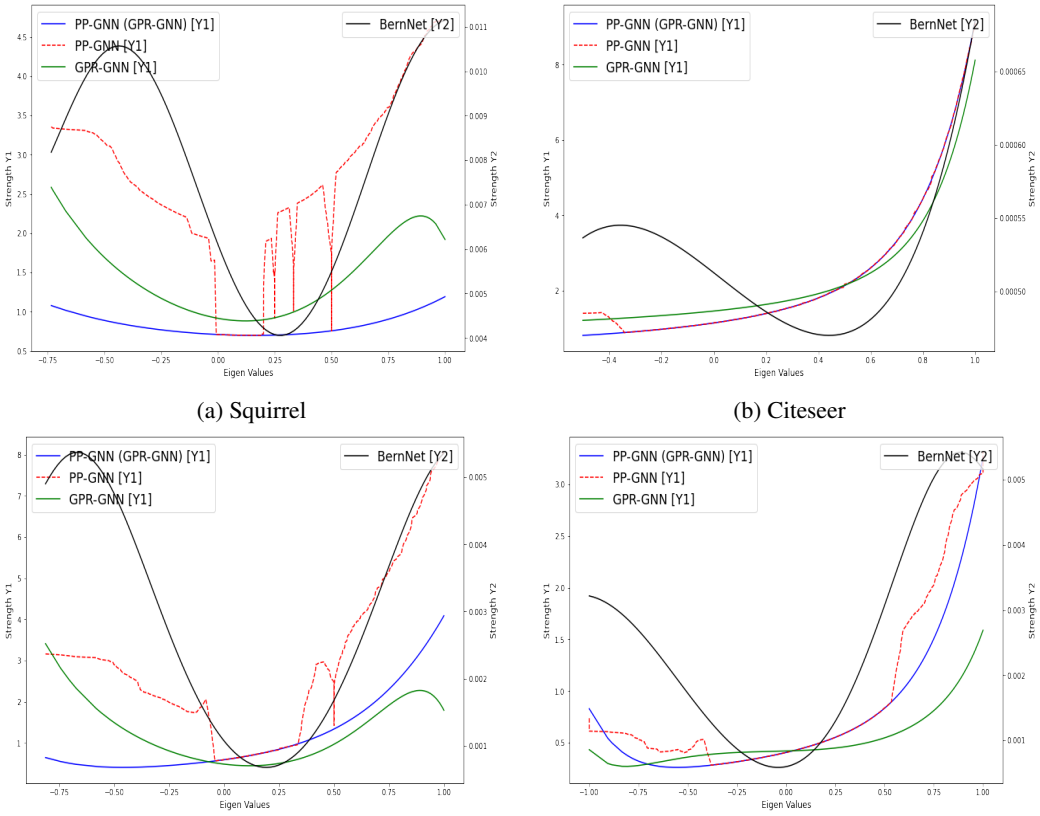
where $\tilde{D}_{ii} = 1 + D_{ii}$, $D_{ii} = \sum_j A_{ij}$ and $\tilde{D}_{ij} = 0, i \neq j$. We work with eigen matrix and eigen values of $\tilde{\mathbf{A}}$.

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 set $\{2, 4, 5\}$. The orders of the polynomial filters are swept in the set $\{2, 4, 6\}$. The number of EVD components are swept in the set $\{256, 1024\}$. In our experiments, we set $\eta_l = \eta_h$ and we vary the η_l parameter in range $(0, 1)$ and $\eta_{gpr} = 1 - \eta_l$.

For optimization, we use the Adam optimizer (Kingma & Ba, 2015). 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 (Paszke et al., 2019). We used Optuna (Akiba et al., 2019) 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.

A.4.5 Adaptable Frequency Responses

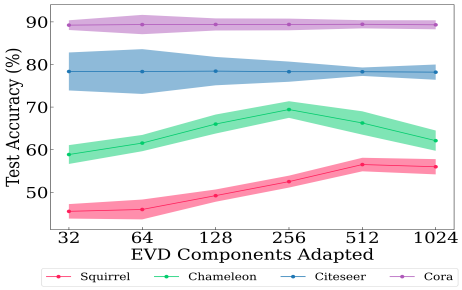
In Figure 1 of the main paper, we observe that PP-GNN learns a complicated frequency response for a heterophilic dataset (Squirrel) and a simpler frequency response for a homophilic dataset (Citeseer). We observe that this trend follows for two other datasets Chameleon (heterophilic) and Computer (homophilic). See Figure 7.



(a) Squirrel (b) Citeseer

(c) Chameleon (d) Computer

Figure 7: Learnt Frequency Responses



(a) Varying No. of EVs

Figure 8: Analyzing varying number of eigenvalues on performance

A.4.6 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 8a, 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.

A.4.7 Training Time Complexity 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 3 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) [Saad (1980), LI (2010), Cullum & Willoughby (2002)]. 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 7, 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

Table 7: **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

amortize this cost over the total hyper-parameters configurations and report the effective training time in the last column on of Table 7.

Average Training Time: In Table 8, 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 9. We can observe in Table 9 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 8: 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

A.5 Comparison with respect to other Polynomial Filtering Methods

Polynomial filters are a class of filters constructed and evaluated from polynomials. These filters can be constructed via multiple bases in the polynomial vector space. Examples of such bases are the canonical mononomial basis or more complicated basis such as the Chebyshev and Bernstein

Table 9: 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 10: 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

basis. Since all these bases (for a polynomial of order K) span the same vector space, it is possible to transform one vector space to another and vice versa. APPNP, GPR-GNN, and BERNNET are all instances of polynomial graph filters. APPNP and GPR-GNN use the monomial basis, while BERNNET uses the Bernstein basis. 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 (Klicpera et al., 2019), can be interpreted as a fixed polynomial graph filter that works with monomial basis. The polynomial coefficients correspond to Personalised PageRank (PPR) (Jeh & Widom, 2003). The node embeddings are learnt by APPNP as described below:

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

APPNP uses fixed γ_k values, where $\gamma_k = \alpha(1-\alpha)^k$, and $\gamma_K = (1-\alpha)^K$; α is a hyper-parameter, $Z_x(X, \Theta)$ are the node features transformed by MLP with parameter Θ. 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 filter coefficients learnable. (Chien et al., 2021) identified that negative coefficients allows the model to exploit high frequency signals required for better performance on heterophilic graphs. As discussed in the introductory paragraph, GPR-GNN uses the monomial basis. The node embeddings are learnt by GPR-GNN as described below:

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

Note that the $\gamma_k (\forall k)$ are learnable coefficients, $Z_x(X, \Theta)$ are the node features transformed by MLP with parameter Θ. 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 making the higher order terms' coefficients uniformly converge to zero. Mitigating the oversmoothing problem limits the complexity of the filter learnt, and therefore makes 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. He et al. (2021) 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. (He et al., 2021) 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 below:

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

Note that in the above expression, $\theta_k (\forall k)$ 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 11.

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	$h(\lambda) = \sum_{k=0}^{K} \frac{\theta_k}{2^K}\binom{K}{k}(2-\lambda)^{K-k}\lambda^k$	$\theta_k \geq 0$

Table 11: Different polynomial filtering based methods. Note that the coefficients of APPNP are fixed (not learnable) PPR coeffcients ($\gamma_k \,\forall\, k$) and the coefficients of GPR-GNN ($\gamma_k \,\forall k$)and BERNNET ($\theta_k \,\forall k$) are learnable.

Previously proposed polynomial filtering approaches rely on a single polynomial filter acting over the entire spectrum. Hence these models are prone to the oversmoothing problem when the degree of such a polynomial is increased (as shown in A.2). This phenomenon limits these models to effectively approximate complex filters. These shortcomings motivates the need for a method that can learn complex frequency responses, while avoiding the oversmoothing trap. PP-GNN, by relying on its piece-wise filtering formulation, can effectively approximate complex frequency response with low-order polynomials. The reliance on multiple polynomial filters, each acting on a subset of the eigenspectrum, enables PP-GNN's to effectively approximate complex responses (as described in Section - 2 of the main paper).

A.6 FUTURE WORK

In this work, we introduced PP-GNN, an effective polynomial filtering approach that has the capability to effectively approximate complex frequency responses. In our current implementation of PP-GNN, We used an unconstrained polynomial filter with monomial basis as the base filter. However, other polynomial filter variants can be used as the base filter. We leave this exploration as future work. One can observe that BERNNET is a special case of GPR-GNN. In other words, the solution space of BERNNET is a subset of that of GPR-GNN. This raises the first open question **Q1:** Why does BERNNET perform better than GPR-GNN? Could the performance gap just be an artifact of model optimization? The authors of BERNNET constrains the filter response to take on non-negative values. This decision is justified by (He et al., 2021) from signal processing perspective. However, the implications of such restriction is unclear for the node classification task. This leads us to the second open question **Q2:** Is it necessary for polynomial filters to have positive frequency response to achieve superior task performance? We intend to pursue these questions as future work.