GRAPH NEURAL NETWORKS ARE INHERENTLY GOOD GENERALIZERS: INSIGHTS BY BRIDGING GNNS AND MLPS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs), as the de-facto model class for representation learning on graphs, are built upon the multi-layer perceptrons (MLP) architecture with additional message passing layers to allow features to flow across nodes. While conventional wisdom largely attributes the success of GNNs to their advanced expressivity for learning desired functions on nodes' ego-graphs, we conjecture that this is *not* the main cause of GNNs' superiority in node prediction tasks. This paper pinpoints the major source of GNNs' performance gain to their intrinsic generalization capabilities, by introducing an intermediate model class dubbed as P(ropagational)MLP, which is identical to standard MLP in training, and then adopt GNN's architecture in testing. Intriguingly, we observe that PMLPs consistently perform on par with (or even exceed) their GNN counterparts across ten benchmarks and different experimental settings, despite the fact that PMLPs share the same (trained) weights with poorly-performed MLP. This critical finding opens a door to a brand new perspective for understanding the power of GNNs, and allow bridging GNNs and MLPs for dissecting their generalization behaviors. As an initial step to analyze PMLP, we show its essential difference with MLP at infinite-width limit lies in the NTK feature map in the post-training stage. Moreover, though MLP and PMLP cannot extrapolate non-linear functions for extreme OOD data, PMLP has more freedom to generalize near the training support. Codes are available at `https://github.com/chr26195/PMLP`.

1 INTRODUCTION

In the past decades, *Neural Networks* (NNs) have achieved great success in many areas. As a classic NN architecture, *Multi-Layer Perceptrons* (MLPs) (Rumelhart et al., 1986) stack multiple *Feed-Forward* (FF) layers with nonlinearity to universally approximate functions. Later, *graph neural networks* (GNNs) (Scarselli et al., 2008; Bruna et al., 2014; Gilmer et al., 2017; Kipf & Welling, 2017; Veličković et al., 2017; Hamilton et al., 2017; Klicpera et al., 2019; Wu et al., 2019) build themselves upon the MLP architecture, e.g., by inserting additional *Message Passing* (MP) operations amid FF layers (Kipf & Welling, 2017) to propagate features across instance nodes.

Two cornerstone concepts lying in the basis of deep learning research are model's *representation* and *generalization* power. While the former is concerned with what function class NNs can approximate and to what extent they can minimize the *empirical risk* $\widehat{\mathcal{R}}(\cdot)$, the latter instead focuses on the inductive bias in the learning procedure, asking how well the learned function can generalize to unseen in- and out-of distribution samples, reflected by the *generalization gap* $\mathcal{R}(\cdot) - \widehat{\mathcal{R}}(\cdot)$. There exist a number of works trying to dissect GNNs' representational power (Xu et al., 2018a; Maron et al., 2019; Oono & Suzuki, 2019) and optimization properties (Xu et al., 2021a), while their generalizability and theoretical connections with MLP are far less well-understood.

In this work, we show that GNNs are intrinsically good generalizers that naturally encourage low generalization gap in node-level prediction, by probing into the key component that indeed contributes to the superiority of GNNs over MLP. In specific, we bridge these two types of NNs by introducing a new class of MLP variants, named as *Propagational MLPs* (PMLPs). During training, PMLPs are exactly the same as a standard MLP (e.g., same architecture, training data, initialization, loss function, and optimization algorithm), and thus have the same training dynamics as MLP (e.g., values of

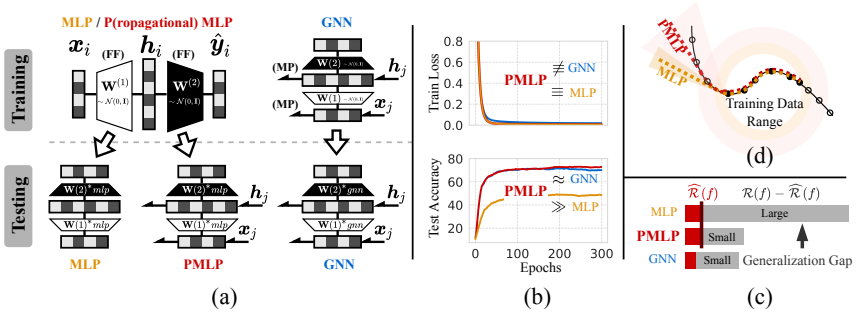


Figure 1: ***(a) Model illustration*** for MLP, GNN (in GCN-style) and PMLP. ***(b) Learning curves*** for node classification on Cora that depicts a typical empirical phenomenon. ***(c) Intrinsic generalizability*** of GNN reflected by close generalization performance of GNN and PMLP. ***(d) Extrapolation illustration***: both MLP and PMLP linearize outside the training data support (•: train sample, ○: test sample), while PMLP transits more smoothly and exhibits larger tolerance for OOD testing sample.

weights, empirical risk, and loss landscape). The only difference is that in the testing phase, PMLPs additionally insert non-parametric message passing layers between (corresp. GCN (Kipf & Welling, 2017)), before (corresp. SGC (Wu et al., 2019)) or after (corresp. APPNP (Klicpera et al., 2019)) FF layers, as shown in Fig. 1(a). We have the following *results and empirical contributions:*

• **An intriguing phenomenon revealing GNNs' inherent generalizability.** This performance of the proposed Propagational MLP (PMLP) in inductive node prediction tasks is remarkably close or even better than its GNN counterpart and significantly outperforms MLP despite being the same model (see example in Fig. 1(b)), which is verified on ten benchmark datasets including three large-scale ones. This indicates that the GNN architecture, naturally (i.e., irrelevant to training) encourages lower generalization gap (Fig. 1(c)), and is also the main source of performance gain from MLP to GNN, providing a new perspective for understanding GNNs and in broader sense enabling us to leverage well-established theoretical frameworks for MLPs to enrich those for GNNs.

• **Consistency across different experimental settings.** The observed phenomenon is consistent across different architecture choices (i.e., number of layers, size of hidden states), model instantiations (i.e., activation function, message passing implementation) and data characteristics (i.e., data split, graph sparsity, noisy edges). In some cases (i.e., removing self-loops and adding noisy edges), PMLP can even outperform its GNN counterpart by a large margin.

• **Practical value.** PMLPs, though originally motivated from theoretical aspects, they also possess practical significance such as better training efficiency (6.82 times more efficient than same-sized GNNs on large datasets), robustness to noisy edges, and serving as a new paradigm for model development.

As mentioned above, our empirical finding narrows down many difference factors between MLPs and GNNs to a key one that attributes to their performance gap, i.e., improvement in generalizability due to the change in network architecture. Then, a natural question arises: 'Why this is the case and what kind of inductive bias is injected when extending MLP to PMLP?'. In this work, we take an initial step towards answering this question with the *following insights and theoretical contributions:*

○ **Comparing MLP, PMLP and GNN at infinite width limit.** We compare MLP, PMLP, and GNN in the *Neural Tangent Kernel* (NTK) regime (Jacot et al., 2018), where models are over-parameterized and gradient descent finds the global optima. From this perspective, the distinction of PMLP and MLP is rooted in the change of NTK *feature map* determined by model architecture while fixing their minimum RKHS-norm solutions (Proposition 1). We also derive the explicit formula for computing the feature map for PMLP/GNN by extending the definition of *Graph Neural Tangent Kernel* (GNTK) (Du et al., 2019) to the node regression setting (Lemma 2).

○ **PMLPs are more likely to generalize OOD nodes.** We consider an important (yet overlooked) aspect of generalization analysis, i.e., *extrapolation* behavior for Out-of-Distribution (OOD) testing samples (Xu et al., 2021b), and reveal that alike MLP, PMLP eventually converges to a linear function as the testing sample increasingly departs from the training data support (Theorem 4). Nevertheless, its convergence rate is smaller than that of MLP by a factor related to node degrees and features'

cosine similarity (Theorem 5). That is, PMLP is more tolerant to OOD samples and thus has larger potential to generalize near the training data support, which is illustrated in in Fig. 1(d).

1.1 Related Works

Generalization, especially for feed-forward NNs (i.e., MLPs), has been extensively studied in the general ML field (Arora et al., 2019a; Allen-Zhu et al., 2019; Cao & Gu, 2019). For GNNs, the large body of existing theoretical works focus on their representational power (Xu et al., 2018a; Maron et al., 2019; Oono & Suzuki, 2019), while their generalization capability is less well-understood. For node-level prediction setting, those works in generalization analysis (Scarselli et al., 2018; Verma & Zhang, 2019; Baranwal et al., 2021; Ma et al., 2021) mainly derive generalization bounds based on, e.g., classic learning theory and algorithmic stability, which, however, could not directly lead to comparison with MLP. For theoretical analysis, the most related work is (Xu et al., 2021b) that studies the extrapolation behavior of MLP, whose results are later used in this work. The authors also shed lights on the extrapolation power of GNNs, but for graph-level prediction with max/min propagation from the perspective of algorithmic alignment, which cannot apply to GNNs with average/sum propagation in node-level prediction that are more commonly used.

Regarding the relation between MLPs and GNNs, there are some recent attempts to boost the performance of MLP to approach that of GNN, by either using contrastive learning (Hu et al., 2021) or knowledge distillation technique (Zhang et al., 2022). However, it is unclear whether these graph-enhanced MLPs can be used to explain the success of GNNs since it is still an open research question to understand these training techniques themselves. There are also few works probing into similar model architectures as PMLP, e.g. Klicpera et al. (2019), which can generally be seen as special cases of PMLP (in Sec. 2). Beyond the algorithmic aspect, we are the first to leverage PMLP as a tool to identify a pervasive phenomenon for bridging MLP and GNNs (in Sec. 3), with in-depth explorations into its theoretical implications (in Sec. 4 and 5).

2 Background and Model Formulation

Assume a graph dataset $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the node set $\mathcal{V}$ contains n nodes instances $\{(\boldsymbol{x}_u, y_u)\}_{u\in\mathcal{V}}$, where $\boldsymbol{x}_u \in \mathbb{R}^d$ denotes node features and y_u is the label. Without loss of generality, y_u can be a categorical variable or a continuous one depending on specific prediction tasks (classification or regression). Instance relations are described by the edge set $\mathcal{E}$ and an associated adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$. In general, the problem is to learn a predictor model with $\hat{y} = f(\boldsymbol{x}; \theta, \mathcal{G}_{\boldsymbol{x}}^k)$ for node-level prediction, where $\mathcal{G}_{\boldsymbol{x}}^k$ denotes the k-hop ego-graph around $\boldsymbol{x}$ over $\mathcal{G}$.

Graph Neural Networks and Multi-Layer Perceptrons. The core design of GNNs lies in the message passing paradigm that recursively aggregates layer-wise neighbored information to encode structural features into node-level representations for downstream prediction. To probe into the connection between mainstream GNNs and MLP from the architectural view, we re-write the GNN formulation in a general form that explicitly disentangles each layer into two operations, namely a *Message-Passing* (MP) operation and then a *Feed-Forwarding* (FF) operation:

$$\text{(MP):} \quad \tilde{\mathbf{h}}_u^{(l-1)} = \sum\nolimits_{v\in\mathcal{N}_u\cup\{u\}} a_G(u,v)\cdot \mathbf{h}_u^{(l-1)}, \quad \text{(FF):} \quad \mathbf{h}_u^{(l)} = \psi^{(l)}\left(\tilde{\mathbf{h}}_u^{(l-1)}\right), \tag{1}$$

where $\mathcal{N}_u$ is the set of neighbored nodes centered at u, $a_{\mathcal{G}}(u,v)$ is the affinity function dependent on graph structure $\mathcal{G}$, $\psi^{(l)}$ denotes a feature transformation mapping at the l-th layer, and $\mathbf{h}_u^{(0)} = \boldsymbol{x}_u$ is the initial node feature. For example, in Graph Convolution Network (GCN) (Kipf & Welling, 2017), $a_{\mathcal{G}}(u,v) = \mathbf{A}_{uv}/\sqrt{\tilde{d}_u\tilde{d}_v}$, where $\tilde{d}_u$ denotes the degree of node u (with self-loop), and $\psi^{(l)}$ is a fully-connected layer with non-linearity. For an L-layer GNN, the prediction is given by $\hat{y}_u = \psi^{(L)}(\mathbf{h}_u^{(L-1)})$, where $\psi^{(L)}$ is often set as linear transformation for regression tasks or with Softmax for classification tasks. Note that GNN models in forms of Eq. 1 degrade to an MLP with a series of FF layers after removing all the MP operations:

$$\hat{y}_u = \psi^{(L)}(\cdots(\psi^{(1)}(\mathbf{x}_u)) = \psi(\mathbf{x}_u). \tag{2}$$

Typical Types of GNN Architectures. Besides GCN, many other mainstream GNN models can be written as the architectural form defined by Eq. 1 whose layer-wise updating rule involves MP and FF operations, e.g., GAT (Veličković et al., 2017) and GraphSAINT (Zeng et al., 2019).

Table 1: Head-to-head comparison of three proposed PMLP models (PMLP$_{GCN}$, PMLP$_{SGC}$ and PMLP$_{APP}$) and the standard MLP.

Model	Train and Valid	Inference	GNN Counterpart
MLP	$\hat{y}_u = \psi(\mathbf{x}_u)$	$\hat{y}_u = \psi(\mathbf{x}_u)$	N/A
PMLP$_{GCN}$		$\mathbf{h}_u^{(l)} = \psi^{(l)}(\text{MP}(\{\mathbf{h}_v^{(l-1)}\}_{v\in\mathcal{N}_u\cup\{u\}}))$	GCN (Kipf & Welling, 2017)
PMLP$_{SGC}$		$\hat{y}_u = \psi(\text{Multi-MP}(\{\mathbf{x}_v\}_{v\in\mathcal{V}}))$	SGC (Wu et al., 2019)
PMLP$_{APP}$		$\hat{y}_u = \text{Multi-MP}(\psi(\{\mathbf{x}_v\}_{v\in\mathcal{V}}))$	APPNP (Klicpera et al., 2019)

Furthermore, there are also other types of GNN architecture represented by SGC (Wu et al., 2019) and APPNP (Klicpera et al., 2019) where the former adopts multiple MP operations on the initial node features, and the later stacks a series of MP operations at the end of FF layers. These two classes of GNNs are also widely explored and studied. For example, SIGN (Rossi et al., 2020), S^{2}GC (Zhu & Koniusz, 2021) and GBP (Chen et al., 2020a) follow the SGC-style, and DAGNN (Liu et al., 2020c), AP-GCN (Spinelli et al., 2020) and GPR-GNN (Chien et al., 2020) follow the APPNP-style.

Bridging GNNs and MLPs: Propagational MLP. After we decouple the MP and FF operations from GNNs' layer-wise updating, we notice that the unique and critical difference of GNNs and MLP lies in whether to adopt MP (somewhere between the input node features and output prediction). To connect two families, we introduce a new model class, dubbed as *Propagational MLP* (PMLP), which has exactly the same architecture as conventional MLP, namely, the same feed-forwarding network. During the inference/testing stage, PMLP$_{GCN}$ incorporates a message passing layer into each layer's feed-forwarding, PMLP$_{SGC}$ adds multiple MP layers in the first layer, and PMLP$_{APP}$ adds them in the last layer. For clear head-to-head comparison, Table 1 summarizes these models with the models adopted in training and testing stages.

3 Empirical Evaluation

In this section, we conduct extensive experiments on a variety of node-level prediction benchmarks. **Section 3.1** shows that the proposed PMLPs can significantly outperform the original MLP though they share the same weights, and approach or even exceed their GNN counterparts. **Section 3.1** shows this phenomenon holds across different experimental settings. Due to space limit, we present the key experimental results in the main text and defer extra results and discussions to Appendix D and F.

Datasets. We use ten widely adopted node classification benchmarks involving different types of networks: three citations networks (Cora, Citeseer and Pubmed), two product co-occurrency networks (Amazon-Computer and Amazon-Photo), two coauthor-ship networks (Coauthor-CS, Coauthor-Computer), and three large-scale networks (OGBN-Arxiv, OGBN-Products and Flickr). More detailed descriptions for dataset statistics and splits are deferred to Appendix D.

Competitors. For fair comparison, we set the layer number and hidden size to the same values for GNN, PMLP and MLP in the same dataset, namely all models can be viewed as sharing the same 'backbone'. Furthermore, for variants PMLP$_{SGC}$ and PMLP$_{APP}$, we consider comparison with SGC and APPNP, respectively, and slightly modify their original architectures: for SGC, we use a multi-layer neural network instead of one-layer after linear message passing, and for APPNP, we remove the residual connection (i.e., $\alpha = 0$) such that the message passing scheme is aligned with other models. We basically use GCN convolution for MP layer and ReLU activation for FF layer for all models unless otherwise stated. PMLPs use the MLP architecture for validation. Detailed descriptions on hyperparameter settings are deferred to Appendix D.

Protocol. We adopt inductive learning setting as the evaluation protocol, which is a commonly used benchmark setting by the community and close to the real scenarios where testing nodes are from new observations and unknown during training. More specifically, for node set $\mathcal{V} = \mathcal{V}_{tr} \cup \mathcal{V}_{te}$ where $\mathcal{V}_{tr}$ (resp. $\mathcal{V}_{te}$) denotes the set of training (resp. testing) nodes, the training process is only exposed to $\mathcal{G}_{tr} = \{\mathcal{V}_{tr}, \mathcal{E}_{tr}\}$, where $\mathcal{E}_{tr} \subset \mathcal{V}_{tr} \times \mathcal{V}_{tr}$ only contains edges for nodes in $\mathcal{V}_{tr}$ and the trained model is tested with the whole graph $\mathcal{G}$ for prediction on $\mathcal{V}_{te}$. Another important reason why we adopt the inductive learning setting is to guarantee a fair comparison between PMLP and its GNN counterpart as the information of validation/testing nodes are not used for training MLP/PMLP.

Table 2: Mean and STD of testing accuracy on node-level prediction benchmark datasets.

	Dataset #Nodes	Cora 2,708	Citeseer 3,327	Pubmed 19,717	A-Photo 7,650	A-Computer 13,752	Coauthor-CS 18,333	Coauthor-Physics 34,493
GNNs	GCN	74.82 ± 1.09	67.60 ± 0.96	76.56 ± 0.85	89.69 ± 0.87	78.79 ± 1.62	91.79 ± 0.35	91.22 ± 0.18
	SGC	73.96 ± 0.59	67.34 ± 0.54	76.00 ± 0.59	83.42 ± 2.47	77.10 ± 2.54	91.24 ± 0.59	89.18 ± 0.46
	APPNP	75.02 ± 2.17	66.58 ± 0.77	76.48 ± 0.49	89.51 ± 0.86	78.29 ± 0.55	91.64 ± 0.34	91.80 ± 0.77
MLPs	MLP	55.30 ± 0.58	56.20 ± 1.27	70.76 ± 0.78	75.61 ± 0.63	63.07 ± 1.67	87.51 ± 0.51	85.09 ± 4.11
	PMLP$_{GCN}$ Δ_{GNN} Δ_{MLP}	$\mathbf{75.86 \pm 0.93}$ $+1.39\%$ $+37.18\%$	$\mathbf{68.00 \pm 0.70}$ $+0.59\%$ $+21.00\%$	$\mathbf{76.06 \pm 0.55}$ -0.65% $+7.49\%$	$\mathbf{89.10 \pm 0.88}$ -0.66% $+17.84\%$	$\mathbf{78.05 \pm 1.21}$ -0.94% $+23.75\%$	$\mathbf{91.76 \pm 0.27}$ -0.03% $+4.86\%$	$\mathbf{91.35 \pm 0.82}$ $+0.14\%$ $+7.36\%$
	PMLP$_{SGC}$ Δ_{GNN} Δ_{MLP}	$\mathbf{75.04 \pm 0.95}$ $+1.46\%$ $+35.70\%$	$\mathbf{67.66 \pm 0.64}$ $+0.48\%$ $+20.39\%$	$\mathbf{76.02 \pm 0.57}$ $+0.03\%$ $+7.43\%$	$\mathbf{86.50 \pm 1.40}$ $+3.69\%$ $+14.40\%$	$\mathbf{74.72 \pm 3.86}$ -3.09% $+18.47\%$	$\mathbf{91.09 \pm 0.50}$ -0.16% $+4.09\%$	$\mathbf{89.34 \pm 1.40}$ $+0.18\%$ $+4.99\%$
	PMLP$_{APP}$ Δ_{GNN} Δ_{MLP}	$\mathbf{75.84 \pm 1.36}$ $+1.09\%$ $+37.14\%$	$\mathbf{67.52 \pm 0.82}$ $+1.41\%$ $+20.14\%$	$\mathbf{76.30 \pm 1.44}$ -0.24% $+7.83\%$	$\mathbf{88.47 \pm 1.64}$ -1.16% $+17.01\%$	$\mathbf{78.07 \pm 2.10}$ -0.28% $+23.78\%$	$\mathbf{91.64 \pm 0.46}$ $+0.00\%$ $+4.72\%$	$\mathbf{91.96 \pm 0.51}$ $+0.17\%$ $+8.07\%$

Table 3: Mean and STD of testing accuracy on three large-scale datasets.

	GCN	SGC	APPNP	MLP	**PMLP**$_{GCN}$	**PMLP**$_{SGC}$	**PMLP**$_{APP}$
OGBN-Arxiv $(\Delta_{GNN}/\Delta_{MLP})$	69.04 ± 0.18	68.56 ± 0.14	69.19 ± 0.12	53.86 ± 0.28	$\mathbf{63.74 \pm 2.28}$ $(-7.68\%/+18.34\%)$	$\mathbf{62.65 \pm 0.35}$ $(-8.62\%/+16.32\%)$	$\mathbf{63.30 \pm 0.17}$ $(-8.51\%/+17.53\%)$
Train loss Train time	1.0480 63.48 ms	1.1124 90.50 ms	1.0396 87.24 ms	1.5917 7.78 ms	**1.5917** **7.78 ms**	**1.5917** **7.78 ms**	**1.5917** **7.78 ms**
OGBN-Products $(\Delta_{GNN}/\Delta_{MLP})$	71.35 ± 0.19	71.17 ± 0.29	70.41 ± 0.07	56.24 ± 0.10	$\mathbf{69.71 \pm 0.13}$ $(-2.30\%/+23.95\%)$	$\mathbf{70.09 \pm 0.13}$ $(-1.52\%/+24.63\%)$	$\mathbf{65.72 \pm 0.09}$ $(-6.66\%/+16.86\%)$
Train loss Train time	0.4013 288.61 ms	0.4018 665.06 ms	0.4311 527.26 ms	1.0841 39.73 ms	**1.0841** **39.73 ms**	**1.0841** **39.73 ms**	**1.0841** **39.73 ms**
Flickr $(\Delta_{GNN}/\Delta_{MLP})$	49.66 ± 0.57	50.93 ± 0.16	45.31 ± 0.28	46.44 ± 0.14	$\mathbf{49.55 \pm 1.30}$ $(-0.22\%/+6.70\%)$	$\mathbf{50.99 \pm 0.39}$ $(+0.12\%/+9.80\%)$	$\mathbf{44.31 \pm 0.24}$ $(-2.21\%/-4.59\%)$
Train loss Train time	1.1462 36.82 ms	1.4030 50.42 ms	0.7449 163.82 ms	1.3344 7.44 ms	**1.3344** **7.44 ms**	**1.3344** **7.44 ms**	**1.3344** **7.44 ms**

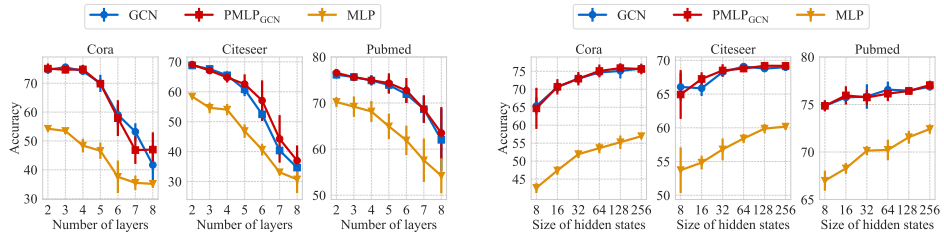
3.1 Main Results

How do PMLPs perform compared with GNNs and MLP on common benchmarks? The main results for comparing the testing accuracy of MLP, PMLPs and GNNs on seven benchmark datasets are shown in Table. 2. We found that, intriguingly, three variants of PMLPs consistently outperform MLP by a large margin on all the datasets despite using the same model with the same set of trainable parameters (furthermore, even the same parameter weights are shared by the trained MLP and PMLPs since they adopt the same implementation during training). Moreover, PMLPs are as effective as their GNN counterparts and can even exceed GNNs in some cases. These results suggest two implications. First, the performance improvement brought by GNNs (or more specifically, the MP operation) over MLP may not purely stem from the more advanced representational power, but the generalization ability. Second, the message passing can indeed contribute to better generalization ability of MLP, though it currently remains unclear how it helps MLP to generalize on unseen testing data. We will later try to shed some light on this in-depth question via theoretical analysis in Section 4.

How do PMLPs perform on larger datasets? We next apply PMLPs to larger graphs which can be harder for extracting informative features from observed data. The results on three large-scale datasets are shown in Table 3 where PMLP still considerably outperforms MLP. Yet differently, there is a certain gap between PMLP and GNN. These results are also valuable, and we conjecture that this is because in such large graphs the relations between inputs and target labels can be more complex, which requires more expressive architectures for learning desired node-level representations. This hypothesis is further validated by the results of training losses in Table 3, which indeed shows that GNNs can yield lower fitting error on the training data, presumably contributing to better accuracy on testing data. Critically though, these results also imply that the MP design works for learning on graph data due to two-fold aspects: better representational power as well as better generalization.

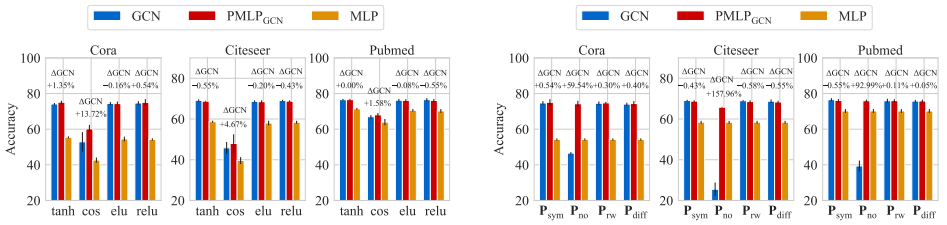
3.2 Further Discussions

We next conduct more experiments and comparison as further investigation for verifying the consistency of the observed phenomenon across different settings regarding model implementation and graph property. We also try to reveal how PMLPs work for representation learning through a close look at the produced embeddings by different models, and the results are deferred to Appendix F.



(a) Number of layers (b) Size of hidden states

Figure 2: Performance variation with increasing layer number and size of hidden states. (See complete results in Appendix F).



(a) Activation functions (b) Transition matrix

Figure 3: Performance variation with different activation functions in FF layer and different transition matrices in MP layer. (See complete results in Appendix F)

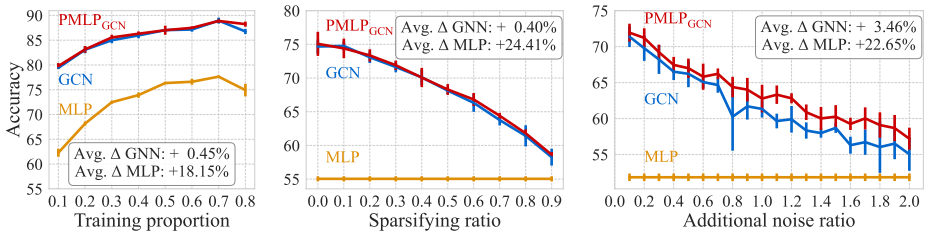


Figure 4: Impact of graph structural information by changing data split, sparsifying the graph, adding random structural noise on Cora. (See complete results in Appendix F)

Q1: What is the impact of model layers and hidden sizes? In Fig. 2(a) and (b), we plot the testing accuracy of GCN, PMLP$_{GCN}$ and MLP w.r.t. different layer numbers and hidden sizes. The results show that the observed phenomenon in Section 3.1 consistently holds with different settings of model depth and width, which suggests that the generalization effect of the MP operation is insensitive to model architectural hyperparameters. The increase of layer numbers cause performance degradation for all three models, presumably because of over-fitting.

Q2: What is the impact of different activation functions in FF layer? As shown in Fig. 3(a), the relative performance rankings among GCN, PMLP and MLP stay consistent across four different activation functions (tanh, cos, ELU, ReLU) and in particular, in some cases the performance gain of PMLP over GCN is further amplified (e.g., with cos activation).

Q3: What is the impact of different propagation schemes in MP functions? We replace the original symmetric normalized transition matrix $\mathbf{P}_{\text{sym}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ used in the MP layer by other commonly used transition matrices: 1) $\mathbf{P}_{\text{no-loop}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, i.e., removing self-loop; 2) $\mathbf{P}_{\text{rw}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$, i.e., random walk matrix; 3) $\mathbf{P}_{\text{diff}} = \sum_{k=0}^{\infty} \frac{1}{e \cdot k!}(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}})^k$, i.e., heat kernel diffusion matrix. The results are presented in Fig. 3(b) where we found that the relative performance rankings of three models keep nearly unchanged when replacing the original MP with random walk and diffusion matrices. And, intriguingly, the performance of GNNs degrade dramatically after removing the self-loop, while the accuracy of PMLPs stays at almost the same level. The possible reason is that the self-loop connection plays an important role in GCN's training stage for preserving enough centered nodes' information, but does not affect PMLP.

Q4: What is the impact of training proportion? We show in Fig. 4(left) that the labeled portion for training data has negligible impacts on the relative performance of three models, indicating the phenomenon is consistent regardless of the amount of graph structure information used in training.

Q5: What is the impact of graph sparsity? As suggested by Fig. 4(middle), when the graph goes sparser, the absolute performance of GCN and PMLP degrades yet their performance gap remains unchanged. This shows that the quality of input graphs indeed impacts the testing performance and tends to control the performance upper bound of the models. Critically though, the generalization effect brought by PMLP is insensitive to the graph completeness.

Q6: What is the impact of noisy structure? As shown in Fig. 4(right), the performances of both PMLP and GCN tend to decrease as we gradually add random connections to the graph, whose amount is controlled by the noise ratio (defined as # noisy edges$/|\mathcal{E}|$), while PMLP shows better robustness to such noise presumably because, again, noisy structures could have negative effects on GNNs in training but not PMLP.

4 Theoretical Insights on the Success of GNNs

Towards theoretically answering why "GNNs are inherently good generalizers" and explaining the superior generalization performance of PMLP and GNN, we next compare MLP, PMLP and GNN from the *Neural Tangent Kernel* (NTK) perspective, and examine their extrapolation behaviors.

4.1 NTK Perspective on MLP, PMLP and GNN

Linearization of Neural Networks. For a neural network $f(\boldsymbol{x};\boldsymbol{\theta}) : \mathcal{X} \to \mathbb{R}$ with initial parameters $\boldsymbol{\theta}_0$ and a fixed input sample $\boldsymbol{x}$, performing first-order Taylor expansion around $\boldsymbol{\theta}_0$ yields the linearized form of NNs (Lee et al., 2019) as follows:

$$f^{lin}(\boldsymbol{x};\boldsymbol{\theta}) = f(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}_0)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0), \tag{3}$$

where the gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}_0)$ could be thought of as a feature map $\phi(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}^{|\boldsymbol{\theta}|}$, depending on the specific initialization. As such, when $\boldsymbol{\theta}_0$ is initialized by Gaussian distribution with certain scaling and the network width tends to infinity (i.e., $m \to \infty$, where m denotes the layer width), the feature map becomes constant and is determined by the model architecture (e.g., MLP, GNN and CNN), inducing a kernel called the *neural tangent kernel* (NTK) (Jacot et al., 2018):

$$\mathrm{NTK}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi_{ntk}(\boldsymbol{x}_i)^\top \phi_{ntk}(\boldsymbol{x}_j) = \langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i;\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_j;\boldsymbol{\theta}) \rangle \tag{4}$$

Kernel Regression with NTK. Recent works (Liu et al., 2020b;a) show that the spectral norm of Hessian matrix in the Taylor series tends to zero with increasing width by $\Theta(1/\sqrt{m})$, and hence the linearization becomes almost exact. Therefore, training an over-parameterized NN using gradient descent with infinitesimal step size is equivalent to kernel regression with NTK (Arora et al., 2019b):

$$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^\top \phi_{ntk}(\boldsymbol{x}), \quad \mathcal{L}(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \boldsymbol{w}^\top \phi_{ntk}(\boldsymbol{x}_i) \right)^2. \tag{5}$$

We next show the equivalence of MLP and PMLP in training at infinite width limit (NTK regime).

Proposition 1. *MLP and its corresponding PMLP have the same minimum RKHS-norm NTK kernel regression solution, but differ from that of GNNs, i.e.,* $\boldsymbol{w}^*_{mlp} = \boldsymbol{w}^*_{pmlp} \neq \boldsymbol{w}^*_{gnn}$.

Implications. From the NTK perspective, stacking additional message passing layers in the testing phase implies transforming the fixed feature map from that of MLP $\phi_{mlp}(\boldsymbol{x})$ to that of GNN $\phi_{gnn}(\boldsymbol{x})$, while fixing $\boldsymbol{w}$. Given $\boldsymbol{w}^*_{mlp} = \boldsymbol{w}^*_{pmlp}$, the superior generalization performance of PMLP (i.e., the key factor of performance surge from MLP to GNN) can be explained by such transformation of feature map to $\phi_{gnn}(\boldsymbol{x})$ in testing:

$$f_{mlp}(\boldsymbol{x}) = \boldsymbol{w}^{*\top}_{mlp}\, \phi_{mlp}(\boldsymbol{x}), \quad f_{pmlp}(\boldsymbol{x}) = \boldsymbol{w}^{*\top}_{mlp}\, \phi_{gnn}(\boldsymbol{x}), \quad f_{gnn}(\boldsymbol{x}) = \boldsymbol{w}^{*\top}_{gnn}\, \phi_{gnn}(\boldsymbol{x}). \tag{6}$$

This perspective significantly simplifies the subsequent theoretical analysis and might inspire more works in this direction. In the following, we first derive the NTK feature map for PMLP and GNN in node regression, and will use the result for analysing the extrapolation behavior of PMLP.

Graph NTK in Node-Level Regression. Following existing work that analyse shallow and wide NNs (Arora et al., 2019a; Chizat & Bach, 2020; Xu et al., 2021b), we focus on a two-layer GNN using average aggregation with self-connection. By extending the original definition of graph neural tangent kernel (GNTK) (Du et al., 2019) from graph-level regression to node-level regression as specified in Appendix. B, we have the following explicit form of GNTK feature map for a two-layer GNN denoted as $\phi_{gnn}(\boldsymbol{x})$.

Lemma 2. *The explicit form of GNTK feature map for a two-layer GNN $\phi_{gnn}(x)$ with average aggregation and ReLU activation in node regression is*

$$\phi_{gnn}(\boldsymbol{x}_i) = c \sum_{j \in \mathcal{N}_i \cup \{i\}} \left[\mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)\top} \mathbf{X}^\top \boldsymbol{a}_j \right), \boldsymbol{w}^{(k)\top} \mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)\top} \mathbf{X}^\top \boldsymbol{a}_j \right), \ldots \right], \quad (7)$$

where $c = O(\tilde{d}^{-1})$ is a constant proportional to the inverse of node degree, $\mathbf{X} \in \mathbb{R}^{n \times d}$ is node features, $\boldsymbol{a}_i \in \mathbb{R}^n$ denotes adjacency vector of node i, $\boldsymbol{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$ is a random Gaussian vector in $\mathbb{R}^d$, two components in the brackets repeat infinitely many times with k ranging from 1 to ∞, and $\mathbb{I}_+$ is an indicator function that outputs 1 if the input is positive otherwise 0.

4.2 MLP v.s. PMLP in Extrapolation

As indicated by Proposition 1, the fundamental difference between MLP and PMLP at infinite width limit stems from the difference of feature map in the testing phase. This reduces the problem of explaining the success of GNN to the question that why this change is significant for generalizability.

Extrapolation Behavior of MLP. One important aspect of generalization analysis is regarding model's behavior when confronted with OOD testing samples (i.e., testing nodes that are considerably outside the training support), a.k.a. *extrapolation analysis*. A previous study on this direction (Xu et al., 2021b) reveal that a standard MLP with ReLU activation quickly converges to a linear function as the testing sample escapes the training support, which is formalized by the following theorem.

Theorem 3. ***(Xu et al., 2021b)**. Suppose $f_{mlp}(\boldsymbol{x})$ is an infinitely-wide two-layer MLP with ReLU trained by square loss. For any direction $\boldsymbol{v} \in \mathbb{R}^d$ and step size $\Delta t > 0$, let $\boldsymbol{x}_0 = t\boldsymbol{v}$, we have*

$$\left| \frac{\left(f_{mlp}(\boldsymbol{x}_0 + \Delta t \boldsymbol{v}) - f_{mlp}(\boldsymbol{x}_0)\right)/\Delta t}{c_{\boldsymbol{v}}} - 1 \right| = O(\frac{1}{t}). \quad (8)$$

where $c_{\boldsymbol{v}}$ is a constant linear coefficient. That is, as $t \to \infty$, $f_{mlp}(\boldsymbol{x}_0)$ converges to a linear function.

The intuition behind this phenomenon is the fact that ReLU MLPs learn piece-wise linear functions with finitely many linear regions and thus eventually becomes linear outside training data support. Now a naturally arising question is how does PMLP compare with MLP regarding extrapolation?

Extrapolation Behavior of PMLP. Based on the explicit formula for $\phi_{gnn}(\boldsymbol{x})$ in Lemma 2, we extend the theoretical result of extrapolation analysis from MLP to PMLP. Our first finding is that, as the testing node feature becomes increasingly outside the range of training data, alike MLP, PMLP (as well as GNN) with average aggregation also converges to a linear function, yet the corresponding linear coefficient reflects its ego-graph property rather than being a fixed constant.

Theorem 4. *Suppose $f_{pmlp}(\boldsymbol{x})$ is an infinitely-wide two-layer MLP with ReLU activation trained using squared loss, and adds average message passing layer before each feed-forward layer in the testing phase. For any direction $\boldsymbol{v} \in \mathbb{R}^d$ and step size $\Delta t > 0$, let $\boldsymbol{x}_0 = t\boldsymbol{v}$, and as $t \to \infty$, we have*

$$\left(f_{pmlp}(\boldsymbol{x}_0 + \Delta t \boldsymbol{v}) - f_{pmlp}(\boldsymbol{x}_0)\right)/\Delta t \quad \to \quad c_{\boldsymbol{v}} \sum_{i \in \mathcal{N}_0 \cup \{0\}} (\tilde{d} \cdot \tilde{d}_i)^{-1}. \quad (9)$$

where $c_{\boldsymbol{v}}$ is the same constant as in Theorem 3, $\tilde{d}_0 = \tilde{d}$ is the node degree (with self-connection) of $\boldsymbol{x}_0$, and $\tilde{d}_i$ is the node degree of its neighbors.

Remark. This result also applies to infinitely-wide two-layer GNN with ReLU activation and average aggregation in node regression settings, except the constant $c_{\boldsymbol{v}}$ is different from that of MLP. The theoretical results presented here and in the following are applicable to both inductive and transductive setting since the variable $\boldsymbol{w}^*_{gnn}$ is not of interest in the analysis.

Convergence Comparison. Though both MLP and PMLP tend to linearize for outlier testing samples, indicating that they have common difficulty to extrapolate non-linear functions, we find

that PMLP in general has more freedom to deviate from the convergent linear coefficient, implying smoother transition from in-distribution (non-linear) to out-of-distribution (linear) regime and thus could potentially generalize to out-of-distribution samples near the range of training data.

Theorem 5. *Suppose all node features are normalized, and the cosine similarity of node $\boldsymbol{x}_i$ and the average of its neighbors is deonoted as $\alpha_i \in [0, 1]$. Then, the convergence rate for $f_{pmlp}(\boldsymbol{x})$ is*

$$\left| \frac{\left(f_{pmlp}(\boldsymbol{x}_0 + \Delta t \boldsymbol{v}) - f_{pmlp}(\boldsymbol{x}_0)\right)/\Delta t}{c_{\boldsymbol{v}} \sum_{i \in \mathcal{N}_0 \cup \{0\}} (\tilde{d} \cdot \tilde{d}_i)^{-1}} - 1 \right| = O\left(\frac{1 + (\tilde{d}_{max} - 1)\sqrt{1 - \alpha_{min}^2}}{t} \right). \tag{10}$$

where $\alpha_{min} = \min\{\alpha_i\}_{i \in \mathcal{N}_0 \cup \{0\}} \in [0, 1]$, and $\tilde{d}_{max} \geq 1$ denotes the maximum node degree in the testing node $\boldsymbol{x}_0$'s neighbors (including itself).

This result indicates larger node degree and feature dissimilarity imply smoother transition and better compatibility with OOD samples. Specifically, when the testing node's degree is 1 (connection to itself), PMLP becomes equivalent to MLP. As reflection in Eq. 10, $\tilde{d}_{max} = 1$, and the bound degrades to that of MLP in Eq. 8. Moreover, when all node features are equal, message passing will become meaningless. Correspondingly, $\tilde{\alpha}_{min} = 1$, and the bound also degrades to that of MLP.

5 More Discussions and Conclusion

Other sources of performance gap between MLP and GNN / When PMLP fails? Besides the intrinsic generalizability of GNNs that is revealed by the performance gain from MLP to PMLP in this work, we note that there are some other less significant but non-negligible sources that attributes to the performance gap between GNN and MLP in node prediction tasks:

- **Expressiveness:** While our experiments find that GNNs and PMLPs can perform similarly in most cases, showing great advantage over MLPs in generalization, in practice, there still exists a certain gap between their expressiveness, which can be amplified in large datasets and cause certain degrees of performance difference. This is reflected by our experiments on three large-scale datasets. Despite that, we can see from Table 3 that the intrinsic generalizability of GNN (corresponding to Δ_{mlp}) is still the major source of performance gain from MLP, and requires further in-depth investigation.
- **Semi-Supervised / Transductive Learning:** As our default experimental setting, inductive learning ensures that testing samples are unobserved during training and keeps the comparison among models fair and reasonable. However in practice, the ability to leverage the information of unlabeled nodes in training is a well-known advantage of GNN (but not the advantage of PMLP). Indeed, we also empirically find that PMLP, combined with some classic semi-supervised learning methods such as manifold regularization (Belkin et al., 2006) and label propagation (Zhu et al., 2003) can again approach GNN in semi-supervised learning setting. Therefore, it would also be valuable to theoretically study the efficacy of GNN in terms of how it benefits from unlabeled nodes in training in comparison with other semi-supervised learning methods.

Current Limitations and Outlooks. From modeling perspective, the current design of PMLP does not take into account parametrized MP layers such as the one used in GAT (Veličković et al., 2017), but could be further extended, e.g., by training parametrized MP layers as independent modules, which will be left as future work. For theoretical analysis, the result in Theorem 5 provides a bound to show PMLP's better potential in OOD generalization, rather than guaranteeing its superior generalization capability. More theoretical works are expected to be done in this direction for more in-depth understanding of the reported findings especially when and why PMLPs perform closely to their GNN counterparts. Moreover, following most theoretical works on NTK, we consider the regression task with squared loss for analysis instead of classification. However, as evidences (Janocha & Czarnecki, 2017; Hui & Belkin, 2020) show squared loss can be as competitive as softmax cross-entropy loss, the insights obtained from regression tasks could also adapt to classification tasks.

Conclusion. In this work, we bridge MLP and GNN by introducing an intermediate model class called PMLP, which is equivalent to MLP in training, but shows significantly better generalization performance after adding non-parametric message passing layers in testing and can rival with its GNN counterpart in most cases. This phenomenon is consistent across different datasets and experimental settings. To shed some lights on this phenomenon, we show despite that both MLP and PMLP cannot extrapolate non-linear functions, PMLP converges slower, indicating smoother transition and better tolerance for out-of-distribution samples.

REFERENCES

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32, 2019.

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332. PMLR, 2019a.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems*, 32, 2019b.

Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. Graph convolution for semi-supervised classification: Improved linear separability and out-of-distribution generalization. *arXiv preprint arXiv:2102.06966*, 2021.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7 (11), 2006.

Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. *Advances in neural information processing systems*, 32, 2019.

Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020b.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Conference on Learning Theory*, pp. 1305–1338. PMLR, 2020.

Weilin Cong, Morteza Ramezani, and Mehrdad Mahdavi. On provable benefits of depth in training graph convolutional networks. *Advances in Neural Information Processing Systems*, 34:9936–9949, 2021.

Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

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. *NeurIPS*, 33:22118–22133, 2020.

Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. Graph-mlp: node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*, 2021.

Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. *arXiv preprint arXiv:2006.07322*, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *ICLR*, 2019.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*, 2020a.

Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33: 15954–15964, 2020b.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020c.

Jiaqi Ma, Junwei Deng, and Qiaozhu Mei. Subgroup generalization and fairness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:1048–1061, 2021.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *International Workshop on Mining and Learning with Graphs*, 2012.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 7:15, 2020.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pp. 243–246, 2015.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4755–4760, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Saurabh Verma and Zhi-Li Zhang. Stability and generalization of graph convolutional neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1539–1548, 2019.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018b.

Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *International Conference on Machine Learning*, pp. 11592–11602. PMLR, 2021a.

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *ICLR*, 2021b.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *International Conference on Learning Representations*, 2022.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.

A PROOF FOR PROPOSITION 1

To analyse the extrapolation behavior of PMLP and compare it with MLP, As mentioned in the main text, training an infinitely wide neural network using gradient descent with infinitesimal step size is equivalent to solving kernel regression with the so-called NTK by minimizing the following squared loss function:

$$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^\top \phi_{ntk}(\boldsymbol{x}), \quad \mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^\top \phi_{ntk}(\boldsymbol{x}_i)\right)^2. \tag{11}$$

Let us now consider an arbitrary minimizer $\boldsymbol{w}^* \in \mathcal{H}$ in NTK's reproducing kernel Hilbert space. The minimizer could be further decomposed as $\boldsymbol{w}^* = \hat{\boldsymbol{w}}^* + \boldsymbol{w}_\perp$, where $\hat{\boldsymbol{w}}^*$ lies in the linear span of feature mappings for training data and $\boldsymbol{w}_\perp$ is orthogonal to $\hat{\boldsymbol{w}}^*$, i.e.,

$$\hat{\boldsymbol{w}}^* = \sum_{i=1}^{n} \lambda_i \cdot \phi_{ntk}(\boldsymbol{x}_i), \quad \langle \hat{\boldsymbol{w}}^*, \boldsymbol{w}_\perp \rangle_{\mathcal{H}} = 0. \tag{12}$$

One observation is that the loss function is unchanged after removing the orthogonal component $\boldsymbol{w}_\perp$:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}^*) &= \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \langle \sum_{i=1}^{n} \lambda_i \cdot \phi_{ntk}(\boldsymbol{x}_i) + \boldsymbol{w}_\perp, \phi_{ntk}(\boldsymbol{x}_i)\rangle_{\mathcal{H}}\right)^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \langle \sum_{i=1}^{n} \lambda_i \cdot \phi_{ntk}(\boldsymbol{x}_i), \phi_{ntk}(\boldsymbol{x}_i)\rangle_{\mathcal{H}}\right)^2 = \mathcal{L}(\hat{\boldsymbol{w}}^*).\end{aligned} \tag{13}$$

This indicates that $\hat{\boldsymbol{w}}^*$ is also a minimizer whose $\mathcal{H}$-norm is smaller than that of $\boldsymbol{w}^*$, i.e., $\|\hat{\boldsymbol{w}}^*\|_{\mathcal{H}} \leq \|\boldsymbol{w}^*\|_{\mathcal{H}}$. It follows that the minimum $\mathcal{H}$-norm solution for Eq. 11 can be expressed as a linear combination of feature mappings for training data. Therefore, solving Eq. 11 boils down to solving a linear system with coefficients $\boldsymbol{\lambda} = [\lambda_i]_{i=1}^n$. Resultingly, the minimum $\mathcal{H}$-norm solution is

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \left[y_i \mathbf{K}^{-1}\right]_i \phi_{ntk}(\boldsymbol{x}_i), \tag{14}$$

where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is the kernel matrix for training data. We see that the final solution is only dependent on training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ and the model architecture used in training (since it determines the form of $\phi_{ntk}(\boldsymbol{x}_i)$). It follows immediately that the min-norm NTK kernel regression solution is equivalent for MLP and PMLP (i.e., $\boldsymbol{w}^*_{mlp} = \boldsymbol{w}^*_{pmlp}$) given that they are the same model trained on the same set of data. In contrast, the architecture of GNN is different from that of MLP, implying different form of feature map, and hence they have different solutions in their respective NTK kernel regression problems (i.e., $\boldsymbol{w}^*_{mlp} \neq \boldsymbol{w}^*_{gnn}$).

B PROOF FOR LEMMA 3

B.1 GNTK FOR GRAPH-LEVEL REGRESSION

Graph Neural Tangent Kernel (GNTK) (Du et al., 2019) is a natural extension of NTK to graph neural networks. Originally, the squared loss function is defined for graph-level regression and the kernel function is defined over a pair of graphs:

$$\mathrm{GNTK}(\mathcal{G}_i, \mathcal{G}_j) = \phi_{gnn}(\mathcal{G}_i)^\top \phi_{gnn}(\mathcal{G}_j) = \langle \nabla_{\boldsymbol{\theta}} f(\mathcal{G}_i;\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}} f(\mathcal{G}_j;\boldsymbol{\theta})\rangle, \tag{15}$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f(\mathcal{G}_i;\boldsymbol{\theta})\right)^2, \tag{16}$$

where $f(\mathcal{G}_i;\boldsymbol{\theta})$ yields prediction for a graph such as the property of a molecule. The formula for calculating GNTK is given by the following (where we modify the original notation for clarity and alignment with our definition of GNTK in node-level regression setting):

$$\begin{gathered}\left[\mathrm{GNTK}^{(0)}(\mathcal{G}_i, \mathcal{G}_j)\right]_{uu'} = \left[\boldsymbol{\Sigma}^{(0)}(\mathcal{G}_i, \mathcal{G}_j)\right]_{uu'} = \boldsymbol{x}_u^\top \boldsymbol{x}_{u'}, \\ \text{where} \quad \boldsymbol{x}_u \in \mathcal{V}_i, \boldsymbol{x}_{u'} \in \mathcal{V}_j.\end{gathered} \tag{17}$$

The message passing operation in each layer corresponds to:

$$\begin{aligned}\left[\mathbf{\Sigma}_{mp}^{(\ell)}\left(\mathcal{G}_i,\mathcal{G}_j\right)\right]_{uu'} &= c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \left[\mathbf{\Sigma}^{(\ell)}\left(\mathcal{G}_i,\mathcal{G}_j\right)\right]_{vv'} \\ \left[\mathrm{GNTK}_{mp}^{(\ell)}\left(\mathcal{G}_i,\mathcal{G}_j\right)\right]_{uu'} &= c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \left[\mathrm{GNTK}^{(\ell)}\left(\mathcal{G}_i,\mathcal{G}_j\right)\right]_{vv'},\end{aligned} \tag{18}$$

where c_u denotes a scaling factor. The calculation formula of feed-forward operation (from $\mathrm{GNTK}_{mp}^{(\ell-1)}\left(\mathcal{G}_i,\mathcal{G}_j\right)$ to $\mathrm{GNTK}^{(\ell)}\left(\mathcal{G}_i,\mathcal{G}_j\right)$) is similar to that for NTKs of MLP (Jacot et al., 2018). The final output of GNTK (without jumping knowledge) is calculated by

$$\mathrm{GNTK}\left(\mathcal{G}_i,\mathcal{G}_j\right) = \sum_{u\in\mathcal{V}_i, u'\in\mathcal{V}_j} \left[\mathrm{GNTK}^{(L-1)}\left(\mathcal{G}_i,\mathcal{G}_j\right)\right]_{uu'}. \tag{19}$$

B.2 GNTK FOR NODE-LEVEL REGRESSION

We next extend the above definition of GNTK to the node-level regression setting, where the model $f(\boldsymbol{x};\boldsymbol{\theta},\mathcal{G})$ outputs prediction of a node, and the kernel function is defined over a pair of nodes in a single graph:

$$\mathrm{GNTK}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \phi_{gnn}(\boldsymbol{x}_i)^\top \phi_{gnn}(\boldsymbol{x}_j) = \left\langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i;\boldsymbol{\theta},\mathcal{G}), \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_j;\boldsymbol{\theta},\mathcal{G})\right\rangle, \tag{20}$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta},\mathcal{G})\right)^2. \tag{21}$$

Then, the explicit formula for GNTK in node-level regression is as follows.

$$\mathrm{GNTK}^{(0)}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \mathbf{\Sigma}^{(0)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) = \boldsymbol{x}_i^\top \boldsymbol{x}_j, \tag{22}$$

Without loss of generality, we consider using random walk matrix as implementation of message passing. Then, the **message passing operation** in each layer corresponds to:

$$\begin{aligned}\mathbf{\Sigma}_{mp}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) &= \frac{1}{(|\mathcal{N}_i|+1)(|\mathcal{N}_j|+1)} \sum_{i'\in\mathcal{N}_i\cup\{i\}} \sum_{j'\in\mathcal{N}_j\cup\{j\}} \mathbf{\Sigma}^{(\ell)}\left(\boldsymbol{x}_{i'},\boldsymbol{x}_{j'}\right) \\ \mathrm{GNTK}_{mp}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) &= \frac{1}{(|\mathcal{N}_i|+1)(|\mathcal{N}_j|+1)} \sum_{i'\in\mathcal{N}_i\cup\{i\}} \sum_{j'\in\mathcal{N}_j\cup\{j\}} \mathrm{GNTK}^{(\ell)}\left(\boldsymbol{x}_{i'},\boldsymbol{x}_{j'}\right),\end{aligned} \tag{23}$$

Moreover, the **feed-forward operation** in each layer corresponds to:

$$\begin{aligned}\mathbf{\Sigma}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) &= c\cdot \mathop{\mathbb{E}}_{u,v\sim\mathcal{N}\left(\mathbf{0},\mathbf{\Lambda}^{(\ell)}\right)}[\sigma(u)\sigma(v)], \\ \dot{\mathbf{\Sigma}}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) &= c\cdot \mathop{\mathbb{E}}_{u,v\sim\mathcal{N}\left(\mathbf{0},\mathbf{\Lambda}^{(\ell)}\right)}[\dot{\sigma}(u)\dot{\sigma}(v)], \\ \mathrm{GNTK}^{(\ell)}(\boldsymbol{x}_i,\boldsymbol{x}_j) &= \mathrm{GNTK}_{mp}^{(\ell-1)}(\boldsymbol{x}_i,\boldsymbol{x}_j)\cdot \dot{\mathbf{\Sigma}}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) + \mathbf{\Sigma}^{(\ell)}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right), \\ \text{where}\qquad \mathbf{\Lambda}^{(\ell)} &= \begin{bmatrix} \mathbf{\Sigma}_{mp}^{(\ell-1)}(\boldsymbol{x}_i,\boldsymbol{x}_i) & \mathbf{\Sigma}_{mp}^{(\ell-1)}(\boldsymbol{x}_i,\boldsymbol{x}_j) \\ \mathbf{\Sigma}_{mp}^{(\ell-1)}(\boldsymbol{x}_j,\boldsymbol{x}_i) & \mathbf{\Sigma}_{mp}^{(\ell-1)}(\boldsymbol{x}_j,\boldsymbol{x}_j) \end{bmatrix}\end{aligned} \tag{24}$$

Suppose the GNN has L layers and the last layer uses linear transformation that is akin to MLP, the final GNTK in node-level regression is defined as

$$\mathrm{GNTK}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \mathrm{GNTK}_{mp}^{(L-1)}(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{25}$$

B.3 GNTK AND FEATURE MAP FOR A TWO-LAYER GNN

We next derive the explicit NTK formula for a two-layer graph neural network in node-level regression setting. For notational convenience, we use $\boldsymbol{a}_i \in \mathbb{R}^n$ to denote adjacency, i.e.,

$$(\boldsymbol{a}_i)_j = \begin{cases} 1/(|\mathcal{N}_i|+1) & \text{if } (i,j)\in\mathcal{E} \\ 0 & \text{if } (i,j)\notin\mathcal{E} \end{cases}, \tag{26}$$

and $\mathbf{G} = \mathbf{X}\mathbf{X}^\top \in \mathbb{R}^{n\times n}$ to denote the Gram matrix of all nodes. Then we have

(First message passing layer)

$$\begin{aligned}\mathrm{GNTK}^{(0)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \boldsymbol{\Sigma}^{(0)}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j, \\ \mathrm{GNTK}^{(0)}_{mp}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \boldsymbol{\Sigma}^{(0)}_{mp}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_j,\end{aligned} \tag{27}$$

(First feed-forward layer)

$$\begin{aligned}\boldsymbol{\Sigma}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= c \cdot \mathop{\mathbb{E}}_{u,v\sim\mathcal{N}\left(\mathbf{0},\boldsymbol{\Lambda}^{(1)}\right)}[\sigma(u)\sigma(v)], \\ \dot{\boldsymbol{\Sigma}}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= c \cdot \mathop{\mathbb{E}}_{u,v\sim\mathcal{N}\left(\mathbf{0},\boldsymbol{\Lambda}^{(1)}\right)}[\dot{\sigma}(u)\dot{\sigma}(v)],\end{aligned} \tag{28}$$

$$\boldsymbol{\Lambda}^{(1)} = \begin{bmatrix} \boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_i & \boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_j \\ \boldsymbol{a}_j^\top \mathbf{G} \boldsymbol{a}_i & \boldsymbol{a}_j^\top \mathbf{G} \boldsymbol{a}_j \end{bmatrix} = \begin{bmatrix} \mathbf{X}^\top \boldsymbol{a}_i \\ \mathbf{X}^\top \boldsymbol{a}_j \end{bmatrix} \cdot \begin{bmatrix} \mathbf{X}^\top \boldsymbol{a}_i & \mathbf{X}^\top \boldsymbol{a}_j \end{bmatrix}$$

By noting that $\boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_j = (\mathbf{X}^\top \boldsymbol{a}_i)^\top \mathbf{X}^\top \boldsymbol{a}_j$ and substituting $\sigma(k) = k \cdot \mathbb{I}_+(k)$, $\dot{\sigma}(k) = \mathbb{I}_+(k)$, where $\mathbb{I}_+(k)$ is an indicator function that outputs 1 if k is positive otherwise 0, we have the following equivalent form for the covariance

$$\begin{aligned}\boldsymbol{\Sigma}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= c \cdot \mathop{\mathbb{E}}_{\boldsymbol{w}\sim\mathcal{N}(\mathbf{0},I_d)} \left[\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j)\right], \\ \dot{\boldsymbol{\Sigma}}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= c \cdot \mathop{\mathbb{E}}_{\boldsymbol{w}\sim\mathcal{N}(\mathbf{0},I_d)} \left[\mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j)\right].\end{aligned} \tag{29}$$

Hence, we have

$$\begin{aligned}\mathrm{GNTK}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \mathrm{GNTK}^{(0)}_{mp}(\boldsymbol{x}_i, \boldsymbol{x}_j) \cdot \dot{\boldsymbol{\Sigma}}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) + \boldsymbol{\Sigma}^{(1)}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= c \cdot \mathop{\mathbb{E}}_{\boldsymbol{w}\sim\mathcal{N}(\mathbf{0},I_d)} \left[\boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_j \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j)\right] \\ &\quad + c \cdot \mathop{\mathbb{E}}_{\boldsymbol{w}\sim\mathcal{N}(\mathbf{0},I_d)} \left[\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j)\right]\end{aligned} \tag{30}$$

(Second message passing layer / The last layer)

Since the GNN uses a linear transformation on top of the (second) message passing layer for output, the neural tangent kernel for a two-layer GNN is given by

$$\begin{aligned}\mathrm{GNTK}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \mathrm{GNTK}^{(1)}_{mp}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= \frac{1}{(|\mathcal{N}_i|+1)(|\mathcal{N}_j|+1)} \sum_{i'\in\mathcal{N}_i\cup\{i\}} \sum_{j'\in\mathcal{N}_j\cup\{j\}} \mathrm{GNTK}^{(1)}(\boldsymbol{x}_{i'}, \boldsymbol{x}_{j'}) \\ &= \left\langle \left[\phi^{(1)}(\boldsymbol{x}_1), \cdots, \phi^{(1)}(\boldsymbol{x}_n)\right]^\top \boldsymbol{a}_i, \left[\phi^{(1)}(\boldsymbol{x}_1), \cdots, \phi^{(1)}(\boldsymbol{x}_n)\right]^\top \boldsymbol{a}_j \right\rangle_{\mathcal{H}}\end{aligned} \tag{31}$$

where $\mathbf{K}^{(1)} \in \mathbb{R}^{n\times n}$ and $\phi^{(1)} : \mathbb{R}^d \to \mathcal{H}$ is the kernel matrix and feature map induced by $\mathrm{GNTK}^{(1)}$. By Eq. 31, the final feature map $\phi_{gnn}(\boldsymbol{x})$ is

$$\phi_{gnn}(\boldsymbol{x}_i) = \left[\phi^{(1)}(\boldsymbol{x}_1), \cdots, \phi^{(1)}(\boldsymbol{x}_n)\right]^\top \boldsymbol{a}_i. \tag{32}$$

Also, notice that in Eq. 30,

$$\begin{gathered}\boldsymbol{a}_i^\top \mathbf{G} \boldsymbol{a}_j \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j) = \phi_i^\top \phi_j, \\ \boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i) \cdot \boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_j) = (\boldsymbol{w}^\top \phi_i)^\top \boldsymbol{w}^\top \phi_j, \\ \text{where } \phi_i = \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+(\boldsymbol{w}^\top \mathbf{X}^\top \boldsymbol{a}_i).\end{gathered} \tag{33}$$

Then, the feature map $\phi^{(1)}$ can be written as

$$\begin{aligned}\phi^{(1)}(\boldsymbol{x}_i) = c' \cdot \Big[& \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(1)\top} \mathbf{X}^\top \boldsymbol{a}_i\right), \boldsymbol{w}^{(1)\top} \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(1)\top} \mathbf{X}^\top \boldsymbol{a}_i\right), \\ & \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(2)\top} \mathbf{X}^\top \boldsymbol{a}_i\right), \boldsymbol{w}^{(2)\top} \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(2)\top} \mathbf{X}^\top \boldsymbol{a}_i\right), \\ & \cdots \\ & \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(\infty)\top} \mathbf{X}^\top \boldsymbol{a}_i\right), \boldsymbol{w}^{(\infty)\top} \mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(\infty)\top} \mathbf{X}^\top \boldsymbol{a}_i\right) \Big]\end{aligned} \tag{34}$$

where $\boldsymbol{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, I_d)$ is random Gaussian vector in $\mathbb{R}^d$, with the superscript $^{(k)}$ denoting that it is the k-th sample among infinitely many i.i.d. sampled ones, c' is a constant. We write Eq. 34 in short as

$$\phi^{(1)}(\boldsymbol{x}_i) = c' \cdot \left[\mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}\mathbf{X}^\top \boldsymbol{a}_i\right), \boldsymbol{w}^{(k)^\top}\mathbf{X}^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}\mathbf{X}^\top \boldsymbol{a}_i\right), \cdots\right]. \tag{35}$$

Finally, substituting Eq. 35 into Eq. 32 completes the proof.

C Proof for Theorem 4 and Theorem 5

To analyse the extrapolation behavior of PMLP along a certain direction $\boldsymbol{v}$ in the testing phase and compare it to MLP, we consider a newly arrived testing node $\boldsymbol{x}_0 = t\boldsymbol{v}$, whose degree (with self-connection) is $\tilde{d}$ and its corresponding adjacency vector is $\boldsymbol{a} \in \mathbb{R}^{n+1}$, where $(\boldsymbol{a})_i = 1/\tilde{d}$ if $(i, 0) \in \mathcal{E}$ otherwise 0. Following (Xu et al., 2021b; Bietti & Mairal, 2019), we consider a constant bias term and denote the data $\boldsymbol{x}$ plus this term as $\hat{\boldsymbol{x}} = [\boldsymbol{x}|1]$. Then, the asymptotic behavior of $f(\cdot)$ at large distances from the training data range can be characterized by the change of network output with a fixed-length step $\Delta t \cdot \boldsymbol{v}$ along the direction $\boldsymbol{v}$, which is given by the following in the NTK regime

$$\frac{1}{\Delta t}\left(f_{mlp}\left(\hat{\boldsymbol{x}}\right) - f_{mlp}\left(\hat{\boldsymbol{x}}_0\right)\right) = \frac{1}{\Delta t}\boldsymbol{w}_{mlp}^{*^\top}(\phi_{mlp}(\hat{\boldsymbol{x}}) - \phi_{mlp}(\hat{\boldsymbol{x}}_0)) \tag{36}$$

$$\frac{1}{\Delta t}\left(f_{pmlp}\left(\hat{\boldsymbol{x}}\right) - f_{pmlp}\left(\hat{\boldsymbol{x}}_0\right)\right) = \frac{1}{\Delta t}\boldsymbol{w}_{mlp}^{*^\top}(\phi_{gnn}(\hat{\boldsymbol{x}}) - \phi_{gnn}(\hat{\boldsymbol{x}}_0)) \tag{37}$$

where $\boldsymbol{x} = \boldsymbol{x}_0 + \Delta t \cdot \boldsymbol{v} = (t + \Delta t)\boldsymbol{v}$, $\hat{\boldsymbol{x}} = [\hat{\boldsymbol{x}}|1]$ and $\hat{\boldsymbol{x}}_0 = [\hat{\boldsymbol{x}}_0|1]$. As our interest is in how PMLP extrapolation, we use $f(\cdot)$ to refer to $f_{pmlp}(\cdot)$ in the rest of the proof. By Lemma. 2, the explicit formula for computing this node's feature map is given by

$$\phi_{gnn}(\hat{\boldsymbol{x}}_0) = \left[\phi^{(1)}(\hat{\boldsymbol{x}}_0), \phi^{(1)}(\boldsymbol{x}_1; \hat{\boldsymbol{x}}_0), \cdots, \phi^{(1)}(\boldsymbol{x}_n; \hat{\boldsymbol{x}}_0)\right]^\top \boldsymbol{a}, \tag{38}$$

where

$$\begin{aligned}\phi^{(1)}(\hat{\boldsymbol{x}}_0) = c' \cdot \Big[&[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}\right), \\ &\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}\right), \ldots\Big],\end{aligned} \tag{39}$$

and similarly

$$\begin{aligned}\phi^{(1)}(\boldsymbol{x}_i; \hat{\boldsymbol{x}}_0) = c' \cdot \Big[&[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}_i\right), \\ &\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}_i \cdot \mathbb{I}_+\left(\boldsymbol{w}^{(k)^\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}_i\right), \ldots\Big],\end{aligned} \tag{40}$$

$\boldsymbol{w}^{(k)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$, with k going to infinity, c' is a constant, $\mathbb{I}_+(k)$ is an indicator function that outputs 1 if k is positive otherwise 0, and $(\boldsymbol{a}_i)_1 = 1/(|\mathcal{N}_i| + 1)$ if i is connected to the new testing node. It follows from Eq. 37 that

$$\begin{aligned}&\frac{1}{\Delta t}\left(f\left(\hat{\boldsymbol{x}}\right) - f\left(\hat{\boldsymbol{x}}_0\right)\right) \\ =&\frac{1}{\Delta t}\boldsymbol{w}_{mlp}^{*^\top}\left[\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0), \cdots, \phi^{(1)}(\boldsymbol{x}_n; \hat{\boldsymbol{x}}) - \phi^{(1)}(\boldsymbol{x}_n; \hat{\boldsymbol{x}}_0)\right]^\top \boldsymbol{a} \\ =&\frac{1}{\tilde{d}\Delta t}\boldsymbol{w}_{mlp}^{*^\top}\left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0)\right) + \frac{1}{\tilde{d}\Delta t}\sum_{i \in \mathcal{N}_0}\boldsymbol{w}_{mlp}^{*^\top}\left(\phi^{(1)}(\boldsymbol{x}_i; \hat{\boldsymbol{x}}) - \phi^{(1)}(\boldsymbol{x}_i; \hat{\boldsymbol{x}}_0)\right)\end{aligned} \tag{41}$$

Now, let us consider $\boldsymbol{w}_{mlp}^{*^\top}\left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0)\right)$. Recall that $\boldsymbol{w}_{mlp}^{*^\top}$ is from infinite dimensional Hilbert space, and $\boldsymbol{w}^{(k)}$ is drawn from Gaussian with k going to infinity in Eq. 39 and Eq. 40, where each $\boldsymbol{w}^{(k)}$ corresponds to some certain dimensions of $\boldsymbol{w}_{mlp}^*$. Let us denote the part that corresponds to the first line in Eq. 39 as $\boldsymbol{\beta}_{\boldsymbol{w}^{(k)}}$ and the second line in Eq. 39 as $\boldsymbol{\gamma}_{\boldsymbol{w}^{(k)}}$. Consider the following way

of rearrangement for (the first element in) $\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0)$

$$\begin{aligned}
&[\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)^\top} [\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \right) - [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)^\top} [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \\
=&[\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^{(k)^\top} [\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \right) - \mathbb{I}_+ \left(\boldsymbol{w}^{(k)^\top} [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \right) \\
&\qquad + \frac{1}{\tilde{d}} \left[\Delta t \boldsymbol{v} \mid 0 \right] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)^\top} [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right),
\end{aligned} \tag{42}$$

where $\frac{1}{\tilde{d}} \left[\Delta t \boldsymbol{v} \mid 0 \right]$ is obtained by subtracting $[\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a}$ from $[\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a}$. Then, we can re-write $\boldsymbol{w}_{mlp}^{*\top} \left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0) \right)$ into a more convenient form:

$$\frac{1}{\Delta t} \boldsymbol{w}_{mlp}^{*\top} \left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0) \right) \tag{43}$$

$$= \int \frac{1}{\tilde{d}} \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\boldsymbol{v} \mid 0] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \tag{44}$$

$$+ \int \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \right) - \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \tag{45}$$

$$+ \int \frac{1}{\tilde{d}} \gamma_{\boldsymbol{w}} \boldsymbol{w}^\top [\boldsymbol{v} \mid 0] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \tag{46}$$

$$+ \int \gamma_{\boldsymbol{w}} \boldsymbol{w}^\top [\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \right) - \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \tag{47}$$

Remark. For other components in Eq. 41, i.e., $\frac{1}{\Delta t} \boldsymbol{w}_{mlp}^{*\top} \left(\phi^{(1)}(\boldsymbol{x}_i; \hat{\boldsymbol{x}}) - \phi^{(1)}(\boldsymbol{x}_i; \hat{\boldsymbol{x}}_0) \right)$ where $i \in \mathcal{N}_0$, the corresponding result of the expansion in Eq. 43 is similar, which only differs by a scaling factor (since the first element in both $\boldsymbol{a}$ and $\boldsymbol{a}_i$ indicating whether the current node is connected to the new testing node is non-zero) and replacing $\boldsymbol{a}$ with $\boldsymbol{a}_i$. Therefore, in the following proof we focus on Eq. 43 and then generalize the result to other components in Eq. 41.

C.1 Proof for Theorem 4 (Convergence to a Linear Function)

We first analyse the convergence of Eq. 43. Specifically, for Eq. 44:

$$\begin{aligned}
&\int \frac{1}{\tilde{d}} \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\boldsymbol{v} \mid 0] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=&\int \frac{1}{\tilde{d}} \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\boldsymbol{v} \mid 0] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0/t, \mathbf{X}/t]^\top \boldsymbol{a} \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
\to&\int \frac{1}{\tilde{d}} \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\boldsymbol{v} \mid 0] \cdot \mathbb{I}_+ \left(\boldsymbol{w}^\top [\boldsymbol{v} \mid 0] \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) = \frac{c'_{\boldsymbol{v}}}{\tilde{d}}, \quad \text{as } t \to \infty
\end{aligned} \tag{48}$$

where the final result is a constant that depends on training data, direction $\boldsymbol{v}$ and node degree $\tilde{d}$. Moreover, the convergence of Eq. 45 is given by

$$\begin{aligned}
&\int \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}, \mathbf{X}]^\top \boldsymbol{a} \right) - \mathbb{I}_+ \left(\boldsymbol{w}^\top [\hat{\boldsymbol{x}}_0, \mathbf{X}]^\top \boldsymbol{a} \right) \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=&\int \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^\top [[\boldsymbol{v} \mid \frac{1}{t + \Delta t}], \frac{\mathbf{X}}{t + \Delta t}]^\top \boldsymbol{a} \right) - \mathbb{I}_+ \left(\boldsymbol{w}^\top [[\boldsymbol{v} \mid \frac{1}{t}], \frac{\mathbf{X}}{t}]^\top \boldsymbol{a} \right) \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
\to&\int \boldsymbol{\beta}_{\boldsymbol{w}}^\top [\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^\top \boldsymbol{a} \left(\mathbb{I}_+ \left(\boldsymbol{w}^\top [\boldsymbol{v} \mid 0] \right) - \mathbb{I}_+ \left(\boldsymbol{w}^\top [\boldsymbol{v} \mid 0] \right) \right) \mathrm{d}\mathbb{P}(\boldsymbol{w}) = 0, \quad \text{as } t \to \infty
\end{aligned} \tag{49}$$

The similar results in Eq. 48 and Eq. 49 also apply to analysis of Eq. 46 and Eq. 47, respectively. By combining these results, we conclude that

$$\frac{1}{\Delta t} \boldsymbol{w}_{mlp}^{*\top} \left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0) \right) \to \frac{c_{\boldsymbol{v}}}{\tilde{d}}, \quad \text{as } t \to \infty \tag{50}$$

It follows that

$$\frac{1}{\Delta t} \left(f(\hat{\boldsymbol{x}}) - f(\hat{\boldsymbol{x}}_0) \right) \to c_{\boldsymbol{v}} \tilde{d}^{-1} \sum_{i \in \mathcal{N}_0 \cup \{0\}} \tilde{d}_i^{-1}, \quad \text{as } t \to \infty. \tag{51}$$

In conclusion, both MLP and PMLP with ReLU activation will eventually converge to a linear function along directions away from the training data. In fact, this result also holds true for two-layer GNNs with weighted-sum style message passing layers by simply replacing $\boldsymbol{w}_{mlp}^*$ by $\boldsymbol{w}_{gnn}^*$ in the proof.

However, a remarkable difference between MLP and PMLP is that the linear coefficient for MLP is a constant $c_{\boldsymbol{v}}$ that is fixed upon a specific direction $\boldsymbol{v}$ and not affected by the inter-connection between testing node $\boldsymbol{x}$ and training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. In contrast, the linear coefficient for PMLP (and GNN) is also dependent on testing node's degree and the degrees of adjacent nodes.

Moreover, by Proposition. 1, MLP and PMLP share the same $\boldsymbol{w}_{mlp}^*$ (including $\boldsymbol{\beta}_{\boldsymbol{w}}$ and $\boldsymbol{\gamma}_{\boldsymbol{w}}$), and thus the constant $c_{\boldsymbol{v}}$ in Eq. 51 is exactly the linear coefficient of MLP. This can also be verified by setting $\boldsymbol{x}$ to be an isolated node, in which case $\tilde{d}^{-1}\sum_{i\in\mathcal{N}_0\cup\{0\}}\tilde{d}_i^{-1}=1$ and PMLP is equivalent to MLP. Therefore, we can directly compare the linear coefficients for MLP and PMLP. As an immediate consequence, if all node degrees of adjacent nodes are larger than the node degree of the testing node, the linear coefficient will become smaller, vice versa.

C.2 Proof for Theorem 5

We next analyse the convergence rate for Eq. 48 and Eq. 49 to see to what extent can PMLP deviate from the converged linear coefficient as an indication of its tolerance to out-of-distribution sample. For Eq. 48, we have

$$\begin{aligned}
&\left|\int \frac{1}{\tilde{d}}\,\boldsymbol{\beta}_{\boldsymbol{w}}^\top[\boldsymbol{v}\mid 0]\cdot\left(\mathbb{I}_+\left(\boldsymbol{w}^\top[\hat{\boldsymbol{x}}_0,\mathbf{X}]^\top\boldsymbol{a}\right)-\mathbb{I}_+\left(\boldsymbol{w}^\top[\boldsymbol{v}\mid 0]\right)\right)\mathrm{d}\mathbb{P}(\boldsymbol{w})\right| \\
\leq\ & \frac{c_1'}{\tilde{d}}\cdot\int\left|\mathbb{I}_+\left(\boldsymbol{w}^\top[\hat{\boldsymbol{x}}_0,\mathbf{X}]^\top\boldsymbol{a}\right)-\mathbb{I}_+\left(\boldsymbol{w}^\top[\boldsymbol{v}\mid 0]\right)\right|\mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=\ & \frac{c_1'}{\tilde{d}}\cdot\int\left|\mathbb{I}_+\left(\boldsymbol{w}^\top[[\boldsymbol{x}_0\mid 1],\mathbf{X}]^\top\boldsymbol{a}\right)-\mathbb{I}_+\left(\boldsymbol{w}^\top[\boldsymbol{x}_0\mid 0]\right)\right|\mathrm{d}\mathbb{P}(\boldsymbol{w})
\end{aligned} \tag{52}$$

Based on the observation that the integral of $|\mathbb{I}_+(\boldsymbol{w}^\top\boldsymbol{v}_1)-\mathbb{I}_+(\boldsymbol{w}^\top\boldsymbol{v}_2)|$ represents the volume of non-overlapping part of two half-balls that are orthogonal to $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, which grows linearly with the angle between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, denoted by $\measuredangle(\boldsymbol{v}_1,\boldsymbol{v}_2)$. Therefore, we have

$$\begin{aligned}
&\frac{c_1'}{\tilde{d}}\cdot\int\left|\mathbb{I}_+\left(\boldsymbol{w}^\top[[\boldsymbol{x}_0\mid 1],\mathbf{X}]^\top\boldsymbol{a}\right)-\mathbb{I}_+\left(\boldsymbol{w}^\top[\boldsymbol{x}_0\mid 0]\right)\right|\mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=\ & \frac{c_1}{\tilde{d}}\cdot\measuredangle\left([[\boldsymbol{x}_0\mid 1],\mathbf{X}]^\top\boldsymbol{a}\ ,\ [\boldsymbol{x}_0\mid 0]\right),
\end{aligned} \tag{53}$$

Note that the first term in the angle can be decomposed as

$$\tilde{d}\cdot[[\boldsymbol{x}_0\mid 1],\mathbf{X}]^\top\boldsymbol{a}=[\boldsymbol{x}_0\mid 0]+[\mathbf{0}\mid 1]+\sum_{i\in\mathcal{N}(0)}\boldsymbol{x}_i. \tag{54}$$

Suppose all node features are normalized, then we have

$$\begin{aligned}
\measuredangle\left([[\boldsymbol{x}_0,1],\mathbf{X}]^\top\boldsymbol{a}\ ,\ [\boldsymbol{x}_0,0]\right) &\leq \measuredangle\left([\boldsymbol{x}_0,0]\ ,\ [\boldsymbol{x}_0,1]\right)+\measuredangle(\hat{\boldsymbol{x}}_0\ ,\ \hat{\boldsymbol{x}}_0+\sum_{i\in\mathcal{N}(0)}\boldsymbol{x}_i) \\
&=\arctan(\frac{1}{t})+\arctan(\frac{(\tilde{d}-1)\sqrt{1-\alpha^2}}{(\tilde{d}-1)\alpha+\sqrt{t^2+1}}) \\
&=O(\frac{1+(\tilde{d}-1)\sqrt{1-\alpha^2}}{t})
\end{aligned} \tag{55}$$

where α denotes the cosine similarity of the testing node and the sum of its neighbors. The last step is obtained by noting $\arctan(x) < x$. Using the same reasoning, for Eq. 49, we have

$$\begin{aligned}
&\left|\int \boldsymbol{\beta}_{\boldsymbol{w}}^{\top}[\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^{\top}\boldsymbol{a}\left(\mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^{\top}\boldsymbol{a}\right) - \mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}, \mathbf{X}]^{\top}\boldsymbol{a}\right)\right)\mathrm{d}\mathbb{P}(\boldsymbol{w})\right| \\
\leq\ & \left|\boldsymbol{\beta}_{\boldsymbol{w}}^{\top}[\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^{\top}\boldsymbol{a}\right| \cdot \int \left|\mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^{\top}\boldsymbol{a}\right) - \mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}, \mathbf{X}]^{\top}\boldsymbol{a}\right)\right|\mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=\ & \left|\boldsymbol{\beta}_{\boldsymbol{w}}^{\top}[\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^{\top}\boldsymbol{a}\right| \cdot \int \left|\mathbb{I}_{+}\left(\boldsymbol{w}^{\top}\left[[\boldsymbol{x}_0 \mid 1], \mathbf{X}\right]^{\top}\boldsymbol{a}\right) - \mathbb{I}_{+}\left(\boldsymbol{w}^{\top}\left[[\boldsymbol{x}_0 \mid \frac{t}{t+\Delta t}], \frac{t}{t+\Delta t}\mathbf{X}\right]^{\top}\boldsymbol{a}\right)\right|\mathrm{d}\mathbb{P}(\boldsymbol{w}) \\
=\ & \left|\boldsymbol{\beta}_{\boldsymbol{w}}^{\top}[\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^{\top}\boldsymbol{a}\right| \cdot \measuredangle\left(\left[[\boldsymbol{x}_0 \mid 1], \mathbf{X}\right]^{\top}\boldsymbol{a}\ ,\ \left[[\boldsymbol{x}_0 \mid \frac{t}{t+\Delta t}], \frac{t}{t+\Delta t}\mathbf{X}\right]^{\top}\boldsymbol{a}\right),
\end{aligned} \tag{56}$$

Note that the second term in the angle can be re-written as

$$\left[[\boldsymbol{x}_0 \mid \frac{t}{t+\Delta t}], \frac{t}{t+\Delta t}\mathbf{X}\right]^{\top}\boldsymbol{a} = \frac{t}{t+\Delta t}\cdot\left[[\boldsymbol{x}_0 \mid 1], \mathbf{X}\right]^{\top}\boldsymbol{a} + \frac{\Delta t}{t+\Delta t}\cdot\left[[\boldsymbol{x}_0 \mid 0], \mathbf{0}\right]^{\top}\boldsymbol{a}, \tag{57}$$

and hence the angle is at most $\Delta t/(t+\Delta t)$ times of that in Eq. 53. It follows that

$$\begin{aligned}
&\left|\int \boldsymbol{\beta}_{\boldsymbol{w}}^{\top}[\hat{\boldsymbol{x}}/\Delta t, \mathbf{X}/\Delta t]^{\top}\boldsymbol{a}\left(\mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}_0, \mathbf{X}]^{\top}\boldsymbol{a}\right) - \mathbb{I}_{+}\left(\boldsymbol{w}^{\top}[\hat{\boldsymbol{x}}, \mathbf{X}]^{\top}\boldsymbol{a}\right)\right)\mathrm{d}\mathbb{P}(\boldsymbol{w})\right| \\
=\ & O(\frac{t+\Delta t}{\tilde{d}})\cdot O(\frac{1+(\tilde{d}-1)\sqrt{1-\alpha^2}}{t})\cdot O(\frac{\Delta t}{t+\Delta t}) \\
=\ & O(\frac{1+(\tilde{d}-1)\sqrt{1-\alpha^2}}{\tilde{d}t})
\end{aligned} \tag{58}$$

For Eq. 46 and Eq. 47, similar results can be derived by bounding $\boldsymbol{w}$ with standard concentration techniques. By substituting the above convergence rates to Eq. 43, and dividing it by the linear coefficient in Eq. 50, the convergence rate for Eq. 43 is

$$\left|\frac{\frac{1}{\Delta t}\boldsymbol{w}_{mlp}^{*\top}\left(\phi^{(1)}(\hat{\boldsymbol{x}}) - \phi^{(1)}(\hat{\boldsymbol{x}}_0)\right) - c_{\boldsymbol{v}}/\tilde{d}}{c_{\boldsymbol{v}}/\tilde{d}}\right| = O(\frac{1+(\tilde{d}-1)\sqrt{1-\alpha^2}}{t}) \tag{59}$$

It follows that the convergence rate for Eq. 37 is $O(\frac{1+(\tilde{d}_{max}-1)\sqrt{1-\alpha_{min}^2}}{t})$, where $\tilde{d}_{max}$ denotes the maximum node degree in the testing node' neighbors (including itself) and

$$\alpha_{min} = \min\{\alpha_i\}_{i\in\mathcal{N}_0\cup\{0\}} \tag{60}$$

denotes the minimum cosine similarity for the testing node' neighbors (including itself). This completes the proof.

D Implementation Details

We present implementation details for our experiments for reproducibility. We implement our model as well as the baselines with Python 3.7, Pytorch 1.9.0 and Pytorch Geometric 1.7.2. All parameters are initialized with Xavier initialization procedure. We train the model by Adam optimizer. Most of the experiments are running with a NVIDIA 2080Ti with 11GB memory, except that for large-scale datasets we use a NVIDIA 3090 with 24GB memory.

D.1 Dataset Description

We use ten widely adopted node classification benchmarks involving different types of networks: three citations networks (Cora, Citeseer and Pubmed), two product co-occurrency networks (Amazon-Computer and Amazon-Photo), two coauthor-ship networks (Coauthor-CS, Coauthor-Computer), and three large-scale networks (OGBN-Arxiv, OGBN-Products and Flickr). For Cora, Citeseer and Pubmed, we use the provided split in (Kipf & Welling, 2017). For Amazon-Computer, Amazon-Photo, Coauthor-CS and Coauthor-Computer, we randomly sample 20 nodes from each class as labeled nodes, 30 nodes for validation and all other nodes for test following (Shchur et al., 2018). For two three-scale datasets, we follow the original splitting Hu et al. (2020) for evaluation. The statistics of these datasets are summarized in Table. 4.

Table 4: Statistics of datasets.

Dataset	# Classes	# Nodes	# Edges	# Node features
Cora (McCallum et al., 2000)	7	2,708	5,278	1,433
Citeseer (Sen et al., 2008)	6	3,327	4,552	3,703
Pubmed (Namata et al., 2012)	3	19,717	44,324	500
A-Photo (McAuley et al., 2015)	8	7,650	119,081	745
A-Computer	10	13,752	245,861	767
Coauthor-CS (Sinha et al., 2015)	15	18,333	81,894	6,805
Coauthor-Physics	5	34,493	247,962	8,415
OGBN-Arxiv (Hu et al., 2020)	40	169,343	1,157,799	128
OGBN-Products	47	2,449,029	61,859,076	100
Flickr (Zeng et al., 2019)	7	89,250	449,878	500

Table 5: Summary of model architectures.

		Cora	Citeseer	Pubmed	Photo	Computer	CS	Physics	Arxiv	Products	Flickr
GCN	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	2
	# FF Layer	2	2	2	2	2	2	2	2	2	2
PMLP_{gcn}	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	2
	# FF Layer	2	2	2	2	2	2	2	2	2	2
SGC	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	2
	# FF Layer	2	2	2	2	2	2	2	2	2	2
PMLP_{sgc}	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	2
	# FF Layer	2	2	2	2	2	2	2	2	2	2
APPNP	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	1
	# FF Layer	2	2	2	2	2	2	2	2	2	2
PMLP_{app}	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	2	2	2	2	2	2	2	2	2	1
	# FF Layer	2	2	2	2	2	2	2	2	2	2
MLP	Hidden Size	64	64	64	32	256	128	128	64	64	64
	# MP Layer	/	/	/	/	/	/	/	/	/	/
	# FF Layer	2	2	2	2	2	2	2	2	2	2

D.2 HYPERPARAMETER SEARCH

We use the same MLP architecture (i.e., number of FF layers and size of hidden states) as backbone for models in the same dataset, same GNN architecture (i.e., number of MP layers) for PMLP and its GNN counterpart, and finetune hyperparameters for each model including dropout rate (from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$), weight decay factor (from $\{0, 0.0001, 0.001, 0.01, 0.1\}$), and learning rate (from $\{0.0001, 0.001, 0.01, 0.1\}$) using grid search.

For model architecture hyperparameters (i.e., layer number, size of hidden states), we fix them as reported in Table. 5, instead of fine-tuning them in favor of GNN or PMLP for each dataset which might introduce bias into their comparison. By default, we set (FF and MP) layer number as 2, and hidden size as 64, but manually adjust in case the performance of GNN is far from the optimal. Furthermore, we have discussed different settings for these architecture hyperparameters and find the performance of PMLP is consistently close to its GNN counterpart.

Table 6: Mean and STD of testing accuracy on Cora dataset. For SGC and APPNP styles, layer number denotes the number of MP layers. The number of FF layers and hidden size are fixed as 2 and 64. 'Res' denotes using residual connection in the form of $\boldsymbol{X}^{(k)} = (1-\alpha)\,\mathrm{MP}(\boldsymbol{X}^{(k-1)}) + \alpha \boldsymbol{X}^{(0)}$, where α is 0.1, and 'ResInf' denotes only using residual connection for inference (or equivalently, training by GNN and testing by GNN+Res).

	# Layers	2	4	8	16	32	64	128
	MLP	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61
$\alpha = 0$	SGC	73.64 ± 0.75	76.12 ± 0.60	76.68 ± 0.99	75.64 ± 1.25	73.72 ± 1.45	66.02 ± 2.07	48.08 ± 2.24
$\alpha = 0$	PMLP_{SGC}	74.38 ± 0.59	76.44 ± 0.72	76.94 ± 0.51	76.10 ± 0.66	74.24 ± 0.65	69.68 ± 1.20	54.04 ± 3.20
$\alpha = 0.1$	SGC+Res	72.26 ± 0.67	74.70 ± 0.68	75.50 ± 0.85	75.82 ± 0.99	75.80 ± 0.89	75.80 ± 0.89 (Converged)	75.80 ± 0.89 (Converged)
$\alpha = 0.1$	SGC+ResInf	71.64 ± 0.55	74.24 ± 1.00	75.62 ± 0.76	75.76 ± 0.81	75.80 ± 1.01	75.42 ± 1.10	75.78 ± 1.01
$\alpha = 0.1$	$\text{PMLP}_{SGC+Res}$	73.20 ± 0.60	75.38 ± 0.83	76.46 ± 0.79	76.84 ± 0.67	76.86 ± 0.75	76.88 ± 0.68	76.88 ± 0.68 (Converged)
	MLP	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61	54.24 ± 0.61
$\alpha = 0$	APPNP	74.38 ± 0.59	76.96 ± 0.86	77.04 ± 0.98	76.22 ± 0.76	75.18 ± 0.61	69.54 ± 5.20	55.30 ± 5.45
$\alpha = 0$	PMLP_{APP}	75.08 ± 0.93	77.04 ± 0.93	77.50 ± 0.95	76.66 ± 0.84	75.42 ± 0.67	70.10 ± 5.05	56.98 ± 6.03
$\alpha = 0.1$	APPNP+Res	72.58 ± 0.73	75.44 ± 0.78	76.48 ± 0.97	76.62 ± 1.00	76.38 ± 0.72	76.38 ± 0.72 (Converged)	76.38 ± 0.72 (Converged)
$\alpha = 0.1$	APPNP+ResInf	72.68 ± 0.49	75.68 ± 0.30	76.46 ± 0.76	76.56 ± 0.97	76.46 ± 0.84	76.46 ± 0.84 (Converged)	76.46 ± 0.84 (Converged)
$\alpha = 0.1$	$\text{PMLP}_{APP+Res}$	73.84 ± 0.55	75.82 ± 1.06	76.84 ± 0.79	77.02 ± 0.82	76.96 ± 0.86	76.94 ± 0.88	76.94 ± 0.88 (Converged)

Table 7: Mean and STD of testing accuracy on Cora dataset. Layer number denotes the number of MP+FF layers, and the number of FF layers is fixed as 2. The hidden size is fixed as 64. For GCNII, $\mathbf{H}^{(\ell+1)} = \sigma(((1-\alpha_\ell)\,\mathrm{MP}(\mathbf{H}^{(\ell)}) + \alpha_\ell \mathbf{H}^{(0)})((1-\beta_\ell)\mathbf{I}_n + \beta_\ell \mathbf{W}^{(\ell)}))$, we set $\alpha_\ell = 0.1$, $\beta_\ell = 0.5/\ell$. ResNet here denotes MLP with residual connections (or equivalently, GCNII without MP operations). For JKNet, we use concatenation for layer aggregation. MLP+JK denotes MLP with jumping knowledge (or equivalently, JKNet without MP operations)

.

	# Layers	2	4	8	16	32	64	128
w/o res.	MLP	54.24 ± 0.61	55.66 ± 1.30	44.96 ± 2.01	29.56 ± 5.64	22.78 ± 3.30	20.32 ± 1.65	19.04 ± 1.30
w/o res.	GCN	72.94 ± 1.77	73.94 ± 2.50	69.82 ± 4.23	37.18 ± 4.41	32.32 ± 4.41	19.06 ± 4.41	24.48 ± 6.82
w/o res.	PMLP_{GCN}	73.58 ± 1.08	76.22 ± 1.30	69.30 ± 3.49	34.16 ± 3.27	29.94 ± 5.50	22.78 ± 5.35	18.22 ± 5.67
with res.	ResNet	53.00 ± 1.51	54.28 ± 1.23	53.88 ± 1.01	52.66 ± 1.67	50.46 ± 0.76	47.36 ± 1.87	40.90 ± 1.53
with res.	GCNII	67.42 ± 1.12	75.28 ± 1.02	76.60 ± 0.60	76.80 ± 0.60	75.70 ± 1.09	73.52 ± 1.50	70.78 ± 1.73
with res.	PMLP_{GCNII}	68.58 ± 1.84	74.98 ± 1.50	76.22 ± 0.43	76.30 ± 1.18	75.88 ± 1.79	73.60 ± 2.08	70.08 ± 3.10
with res.	MLP+JK	53.26 ± 1.97	55.02 ± 0.41	53.80 ± 1.02	55.08 ± 1.54	54.08 ± 0.81	48.66 ± 1.25	31.30 ± 2.10
with res.	JKNet	65.16 ± 1.46	71.98 ± 1.12	71.70 ± 0.99	72.54 ± 0.62	71.60 ± 1.69	64.08 ± 1.85	48.76 ± 8.00
with res.	PMLP_{JKNet}	67.82 ± 0.97	73.08 ± 0.76	73.70 ± 0.60	74.14 ± 1.09	72.24 ± 1.27	66.88 ± 1.65	45.24 ± 5.05

For other hyperparameters (i.e., learning rate, dropout rate, weight decay factor), we finetune them separately for each model on each dataset based on the performance on validation set. For PMLP, we use the MLP architecture for validation rather than using GNN since we find there is only slight difference in performance. In that sense, all PMLPs share the same training and validation process as the vanilla MLP, making them exactly the same model before inference.

E Model Depth, Over-Smoothing and Residual Connection

We have conducted additional experiments to shed lights on broader aspects of GNNs including model depth, over-smoothing and residual-connection. Table. 6 reports the results for SGC- and APPNP-style GNNs that use consecutive MP layers, and Table. 7 reports the results for SGC-style GNNs where MP layer and FF layer are coupled. Reported experiments are conducted on Cora

dataset, and the similar trend is also observed on other datasets. In summary, we have the following findings and insights:

- As we can see in Table. 6 and Table. 7, PMLP and its GNN counterpart (in GCN-, SGC- or APPNP-style) achieve close performance even for very deep (more than 100 layers) networks. Their performance drops as layer increases, indicating over-smoothing is an issue for both PMLP and GNN without residual connections.
- As we can see in Table. 6, for GNNs using consecutive MP layers (i.e. SGC-, APPNP-style), over-smoothing can be effectively addressed by 1) using MP layers with residual connections in inference (corresponding to PMLP$_{SGC+Res}$ and PMLP$_{APP+Res}$) or even 2) only adding residual connections in inference if message passing is already used in training (corresponding to SGC+ResInf and APPNP+ResInf).
- As we can see in Table. 7, for GCN-style GNNs whose MP layer and FF layer are coupled, training a MLP with residual connections alike ResNet (or equivalently, GCNII without message passing) and adding MP layers in inference yields a model called PMLP$_{GCNII}$ whose performance is consistently close to the vanilla GCNII (Chen et al., 2020b). Similar results are also observed between PMLP$_{JKNet}$ and JKNet (Xu et al., 2018b).

These results indicate that over-smoothing is more of a problem in inference and generalization, since over-smoothing occurs in training GNNs but not in training PMLPs, and still they achieve similar testing performance. This is somewhat in alignment with Cong et al. (2021) where the authors theoretically show very deep GNNs can still achieve high training accuracy but poor generalizability. Moreover, using residual connections (in inference for SGC- and APPNP-style GNNs, in training and inference for GCN-style GNNs) might be an effective way to improve model's generalization performance in case of over-smoothing.

F Additional Experimental Results

We supplement more experimental results in this section including extensions of the results in the main text and visualizations of the internal representations of nodes learned by 2-layer MLP, GNN, and PMLP on Cora and Citeseer datasets.

As we can see from Fig. 10-13, both PMLPs and GNNs show better capability for separating nodes of different classes than MLP in the internal layer, despite the fact that PMLPs share the same set of weights with MLP. Such results might indicate that the superior classification performance of GNNs mainly stem from the effects of message passing in inference, rather than GNNs' ability of learning better node representations.

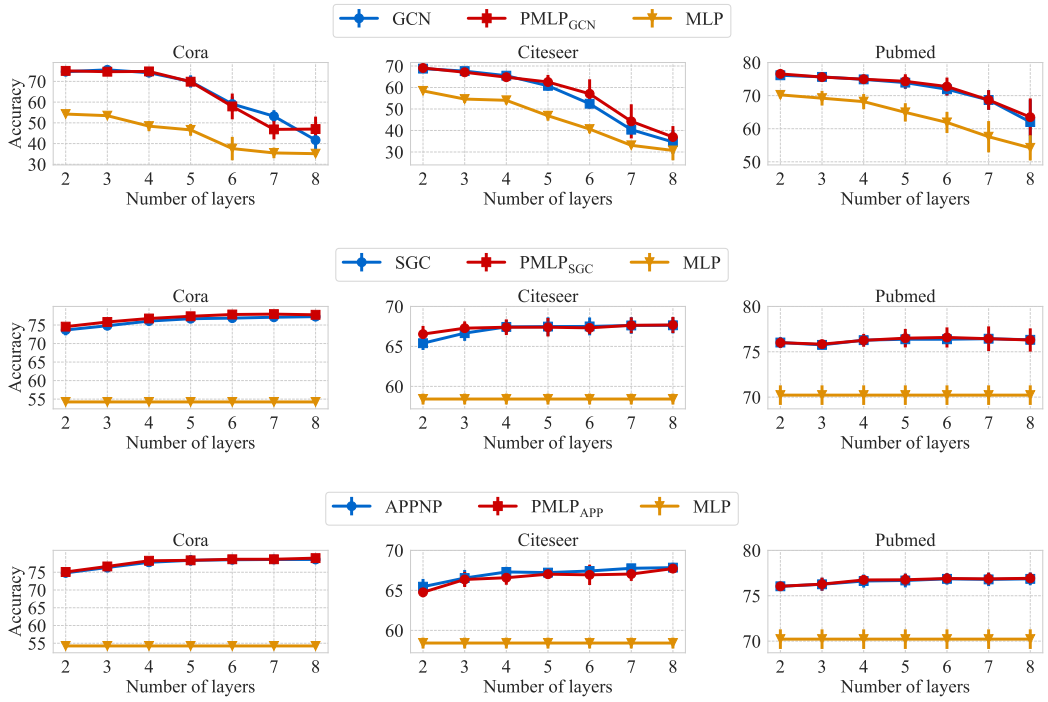


Figure 5: Performance variation with increasing layer number. Layer number here denotes number of FF and MP layers for GCN, and number of MP layers for SGC and APPNP.

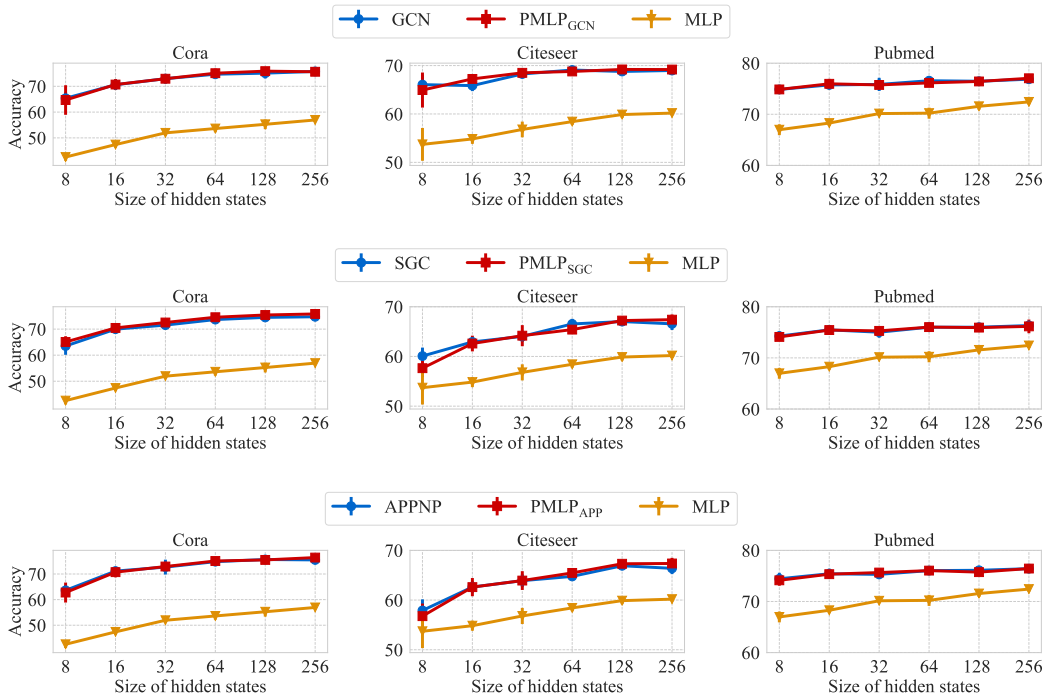
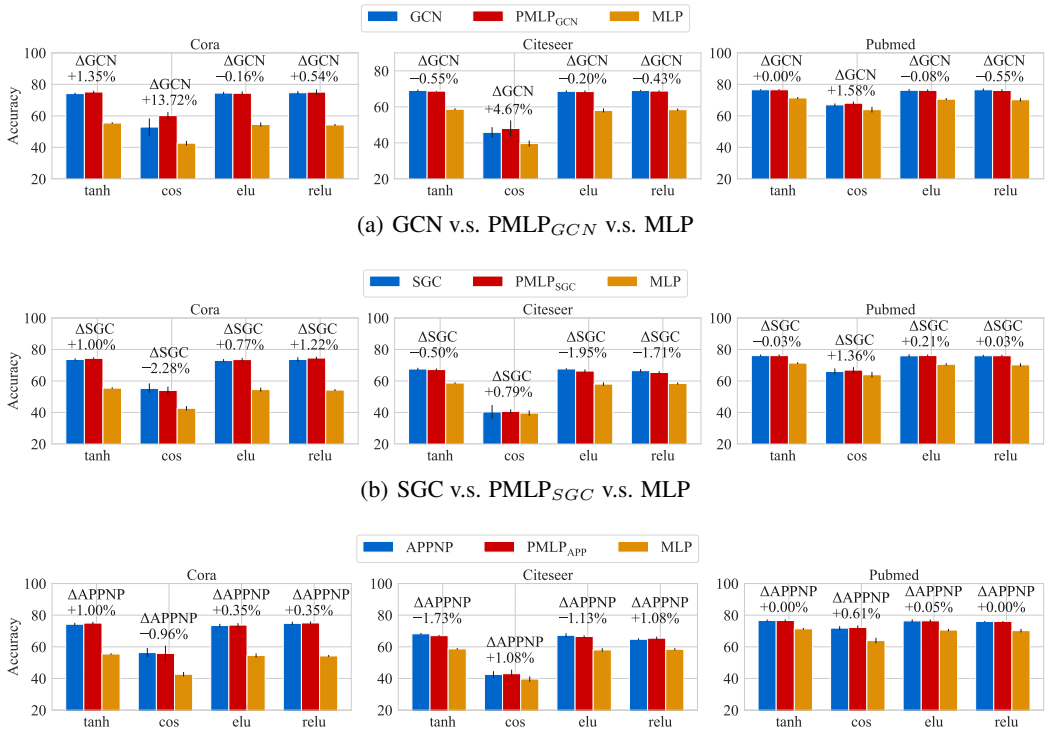


Figure 6: Performance variation with increasing size of hidden states.

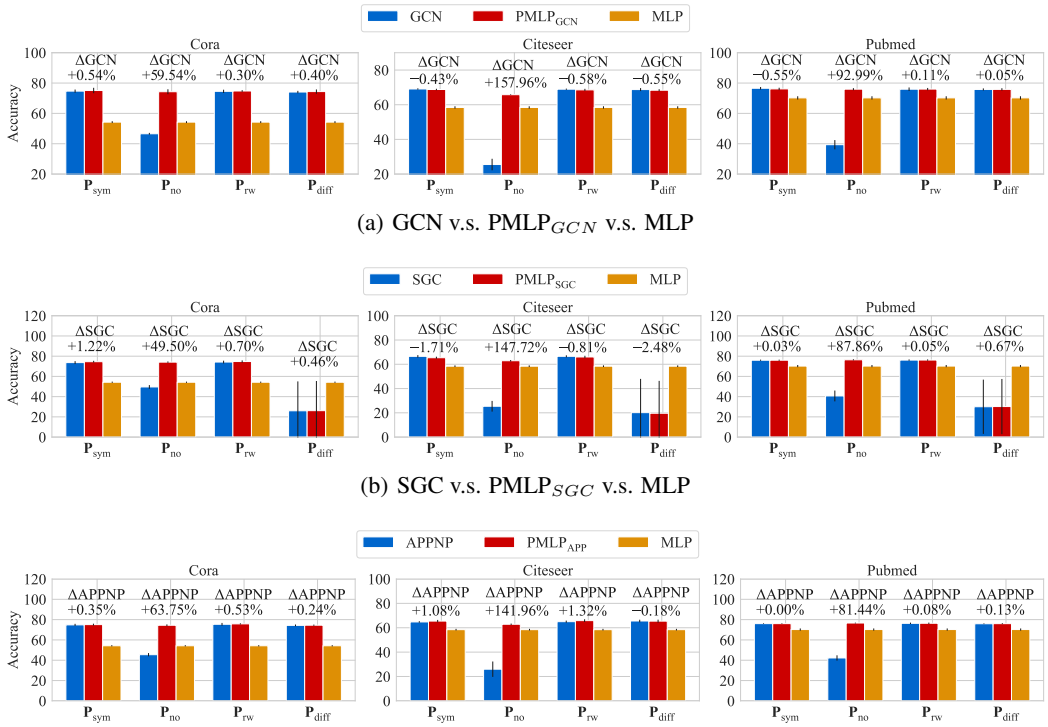


(a) GCN v.s. PMLP$_{GCN}$ v.s. MLP

(b) SGC v.s. PMLP$_{SGC}$ v.s. MLP

(c) APPNP v.s. PMLP$_{APP}$ v.s. MLP

Figure 7: Performance variation with different activation functions in FF layer.

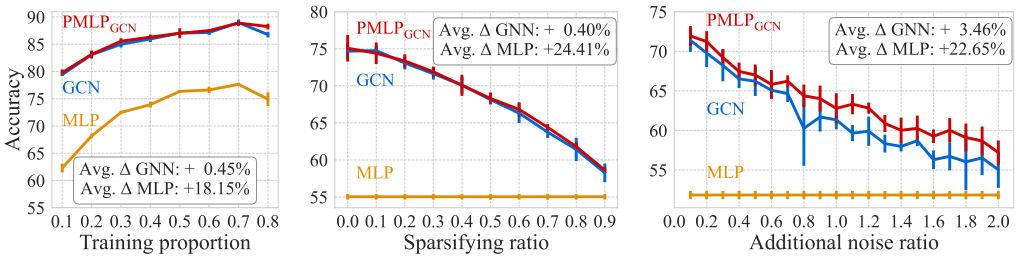


(a) GCN v.s. PMLP$_{GCN}$ v.s. MLP

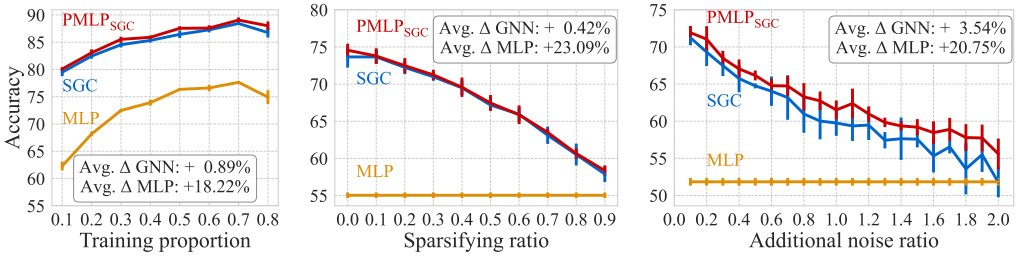
(b) SGC v.s. PMLP$_{SGC}$ v.s. MLP

(c) APPNP v.s. PMLP$_{APP}$ v.s. MLP

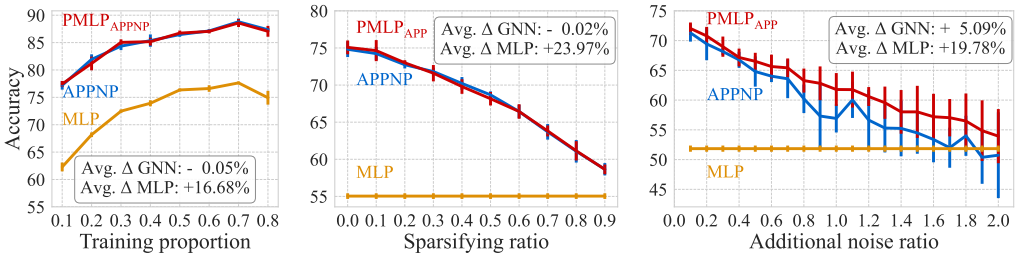
Figure 8: Performance variation with different transition matrices in MP layer.



(a) GCN v.s. PMLP$_{GCN}$ v.s. MLP

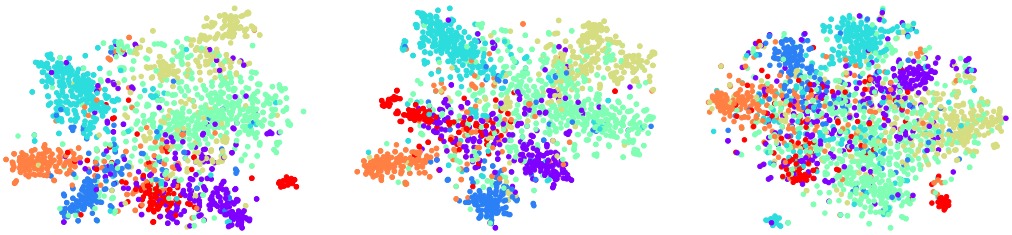


(b) SGC v.s. PMLP$_{SGC}$ v.s. MLP



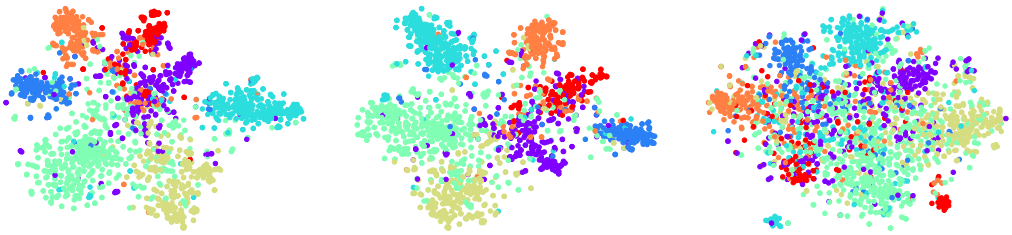
(c) APPNP v.s. PMLP$_{APP}$ v.s. MLP

Figure 9: Impact of graph structural information by changing data split, sparsifying the graph, adding random structural noise on Cora.



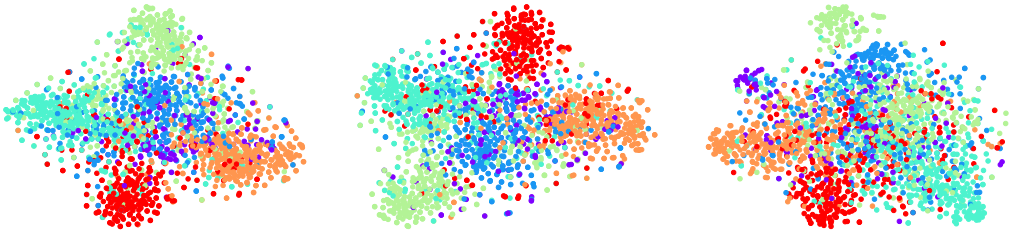
(a) GCN (b) PMLP_{GCN} (c) MLP

Figure 10: Visualization of node embeddings (2-D projection by t-SNE) in the internal layer for two-layer MLP, GCN and PMLP on Cora.



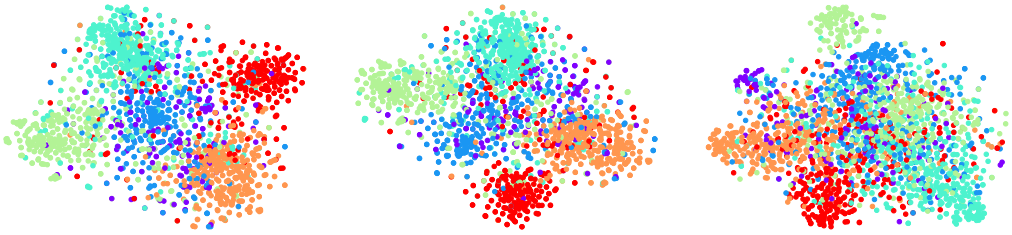
(a) SGC (b) PMLP_{SGC} (c) MLP

Figure 11: Visualization of node embeddings (2-D projection by t-SNE) in the internal layer for two-layer MLP, SGC and PMLP on Cora.



(a) GCN (b) PMLP_{GCN} (c) MLP

Figure 12: Visualization of node embeddings (2-D projection by t-SNE) in the internal layer for two-layer MLP, GCN and PMLP on Citeseer.



(a) SGC (b) PMLP_{SGC} (c) MLP

Figure 13: Visualization of node embeddings (2-D projection by t-SNE) in the internal layer for two-layer MLP, SGC and PMLP on Citeseer.