ICLR

Graph Neural Networks are Inherently Good Generalizers: Insights by Bridging GNNs and MLPs

Presenter: Chenxiao Yang

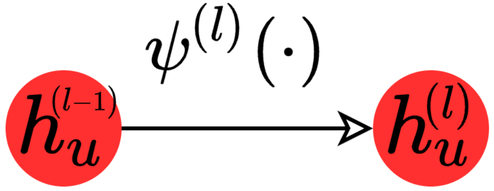
Outline

- Architectural Connection between GNN and MLP
- Bridging GNN and MLP : Propagational MLP (PMLP)
- Experiments
- Quick Guide of Implementation
- GNNs are Inherently Good Generalizers
- Theoretical Analysis

Architectural Connection between GNN and MLP

- **Each layer of MLP (multi-layer perceptrons)** consists of linear transformation and non-linear activation :

(Feed-Forward / FF) : $\mathbf{h}_u^{(l)} = \psi^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), \quad \psi^{(l)}(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W}^{(l)})$

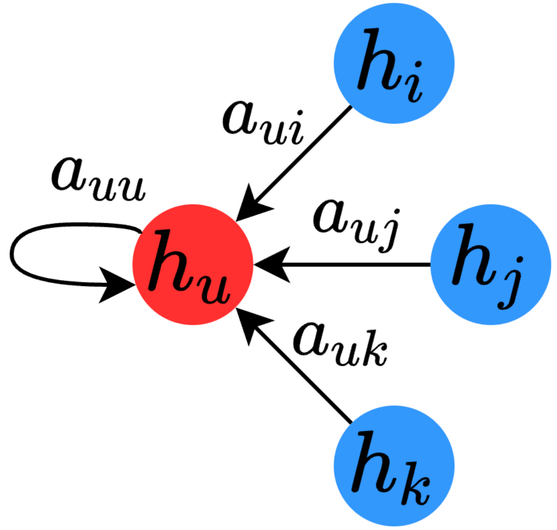


- **Each layer of GNN (graph neural networks)** additionally incorporates a message passing operation to propagate node features :

(Message Passing / MP) : $\tilde{\mathbf{h}}_u^{(l-1)} = \sum_{v \in \mathcal{N}_u \cup \{u\}} a_{uv} \cdot \mathbf{h}_v^{(l-1)}$

e.g. $a_{uv} = \mathbf{A}_{uv} / \sqrt{\tilde{d}_u \tilde{d}_v}$ for GCN.

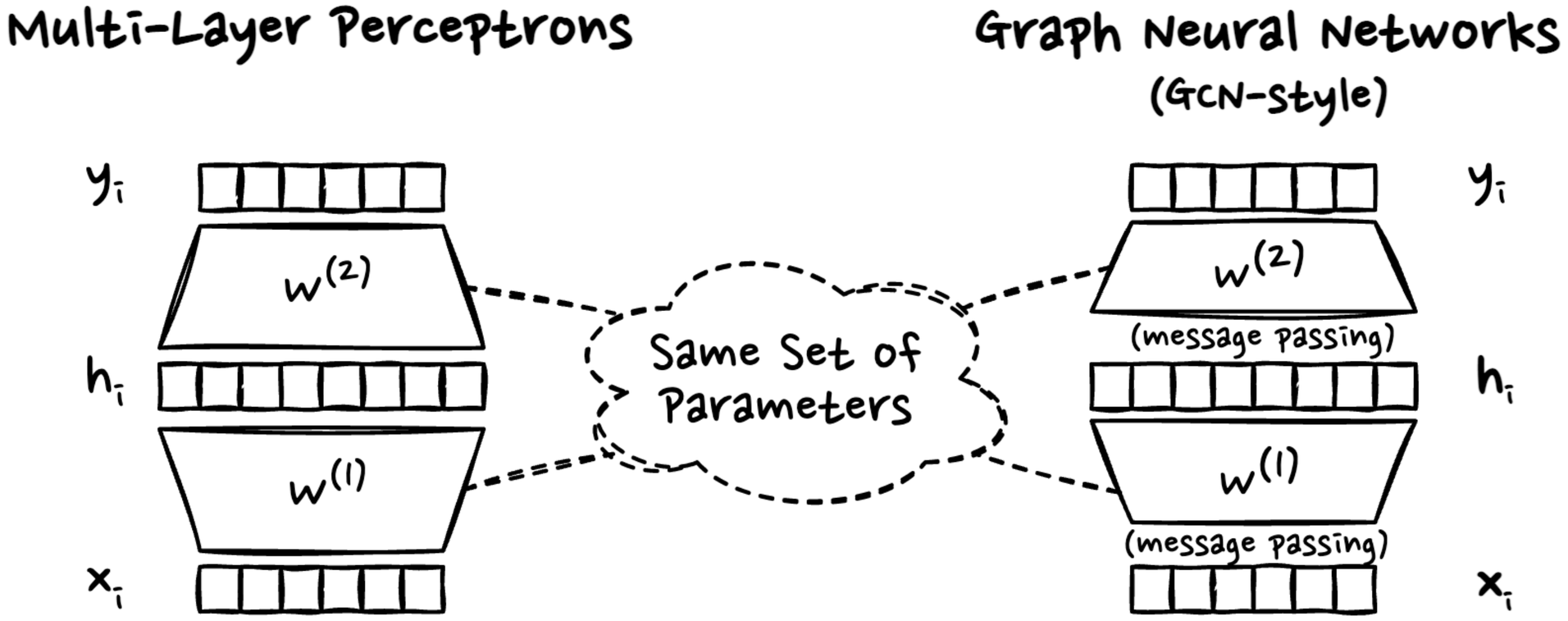
(Feed-Forward / FF) : $\mathbf{h}_u^{(l)} = \psi^{(l)}\left(\tilde{\mathbf{h}}_u^{(l-1)}\right), \quad \psi^{(l)}(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W}^{(l)})$



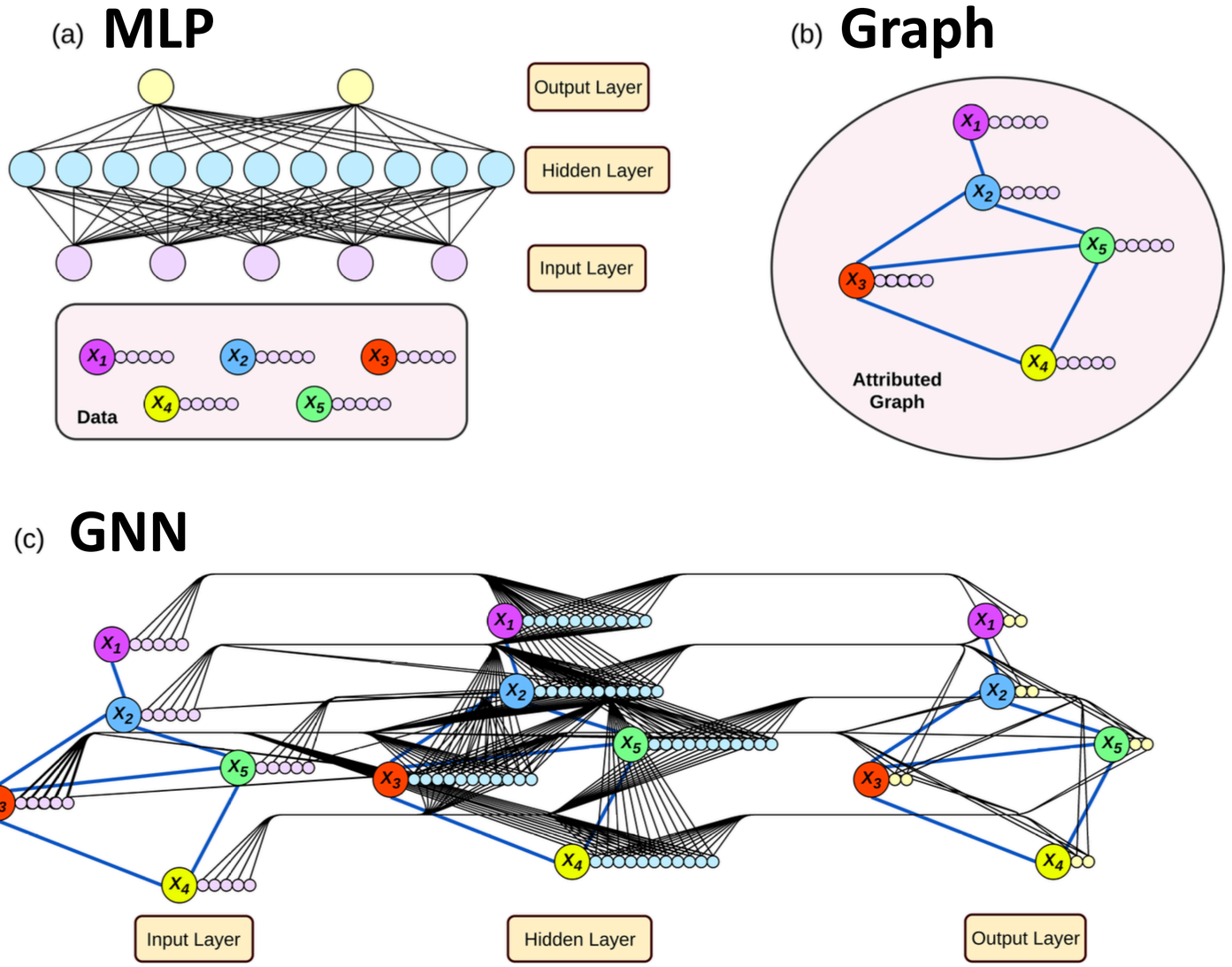
Architectural Connection between GNN and MLP

Q: *What if we remove all message passing operations / layers in GNNs?*

- **Observation** : GNNs without message passing operations become MLPs, and they share the same set of learnable parameters.



Architectural Connection between GNN and MLP

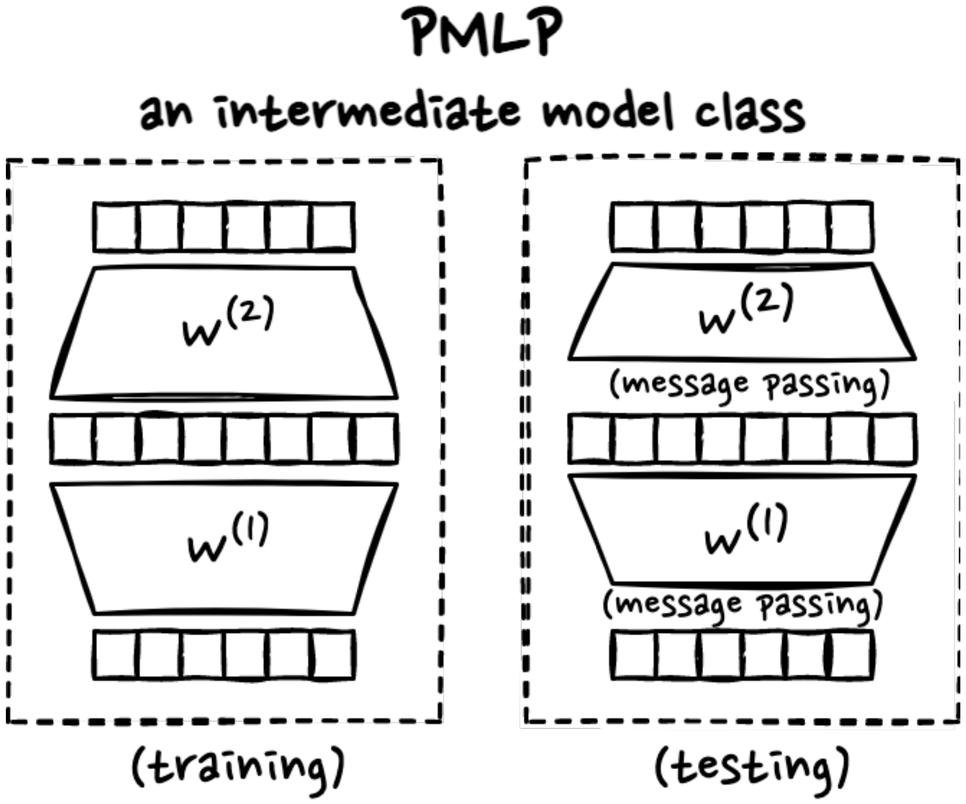


Hartnett, Gavin S., et al. "Deep Generative Modeling in Network Science with Applications to Public Policy Research." arXiv 2020.

Bridging GNN and MLP : Propagational MLP (PMLP)

Q: *Does there exist any intermediate models between MLPs and GNNs ?*

PMLP = MLP (in training) + GNN (for inference)



- **Training** : PMLPs are exactly the same as a standard MLP (e.g. same architecture, data, loss function, optimization algorithm)

- **Testing / Inference** : PMLPs additionally insert message passing (MP) layers amid feed-forward (FF) layers, to align with various GNN architectures.

Bridging GNN and MLP : Propagational MLP (PMLP)

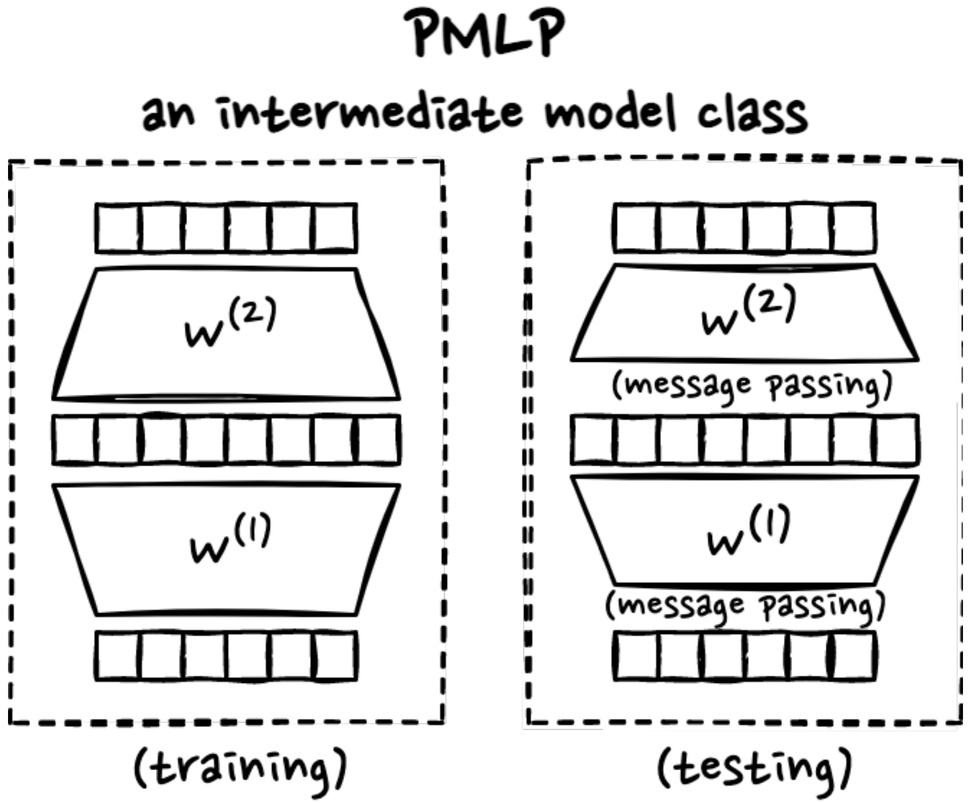
- **PMLP has different instantiations depending on the architecture of GNNs.**

Model	Train and Valid	Inference
MLP	MLP: $\hat{y}_u = \psi(\mathbf{x}_u)$	MLP
PMLP_{GCN}		GCN: $\psi^{(l)}(\text{MP}(\{\mathbf{h}_v^{(l-1)}\}_{v\in\mathcal{N}_u\cup\{u\}}))$
PMLP_{SGC}		SGC: $\psi(\text{Multi-MP}(\{\mathbf{x}_v\}_{v\in\mathcal{V}}))$
PMLP_{APPNP}		APPNP: $\text{Multi-MP}(\psi(\{\mathbf{x}_v\}_{v\in\mathcal{V}}))$
PMLP_{GCNII}	ResNet	GCNII
PMLP_{JKNet}	MLP+JK	JKNet

(and more ….)

Note : *We allow the model after removal of message passing layers to be other models, such as ResNet.*

Bridging GNN and MLP : Propagational MLP (PMLP)



Equivalent ways to view PMLP :

1. PMLP = MLP with test-time message passing operations.
2. PMLP = GNN without message passing layers in training.
3. PMLP = Dropping all edges while training GNN.
4. PMLP = Using pretrained weights of MLP on GNN.

Experiments

Q: *How does PMLP perform compared with MLP and GNN ?*

	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$_{GCN}$	$\mathbf{75.86 \pm 0.93}$	$\mathbf{68.00 \pm 0.70}$	$\mathbf{76.06 \pm 0.55}$	$\mathbf{89.10 \pm 0.88}$	$\mathbf{78.05 \pm 1.21}$	$\mathbf{91.76 \pm 0.27}$	$\mathbf{91.35 \pm 0.82}$
	Δ_{GNN}	$+1.39\%$	$+0.59\%$	-0.65%	-0.66%	-0.94%	-0.03%	$+0.14\%$
	Δ_{MLP}	$+37.18\%$	$+21.00\%$	$+7.49\%$	$+17.84\%$	$+23.75\%$	$+4.86\%$	$+7.36\%$
	PMLP$_{SGC}$	$\mathbf{75.04 \pm 0.95}$	$\mathbf{67.66 \pm 0.64}$	$\mathbf{76.02 \pm 0.57}$	$\mathbf{86.50 \pm 1.40}$	$\mathbf{74.72 \pm 3.86}$	$\mathbf{91.09 \pm 0.50}$	$\mathbf{89.34 \pm 1.40}$
	Δ_{GNN}	$+1.46\%$	$+0.48\%$	$+0.03\%$	$+3.69\%$	-3.09%	-0.16%	$+0.18\%$
	Δ_{MLP}	$+35.70\%$	$+20.39\%$	$+7.43\%$	$+14.40\%$	$+18.47\%$	$+4.09\%$	$+4.99\%$
	PMLP$_{APP}$	$\mathbf{75.84 \pm 1.36}$	$\mathbf{67.52 \pm 0.82}$	$\mathbf{76.30 \pm 1.44}$	$\mathbf{88.47 \pm 1.64}$	$\mathbf{78.07 \pm 2.10}$	$\mathbf{91.64 \pm 0.46}$	$\mathbf{91.96 \pm 0.51}$
	Δ_{GNN}	$+1.09\%$	$+1.41\%$	-0.24%	-1.16%	-0.28%	$+0.00\%$	$+0.17\%$
	Δ_{MLP}	$+37.14\%$	$+20.14\%$	$+7.83\%$	$+17.01\%$	$+23.78\%$	$+4.72\%$	$+8.07\%$

Table. Experimental results for inductive node classification tasks.

A vanilla MLP without any special training tricks can be

as effective as GNNs

Experiments

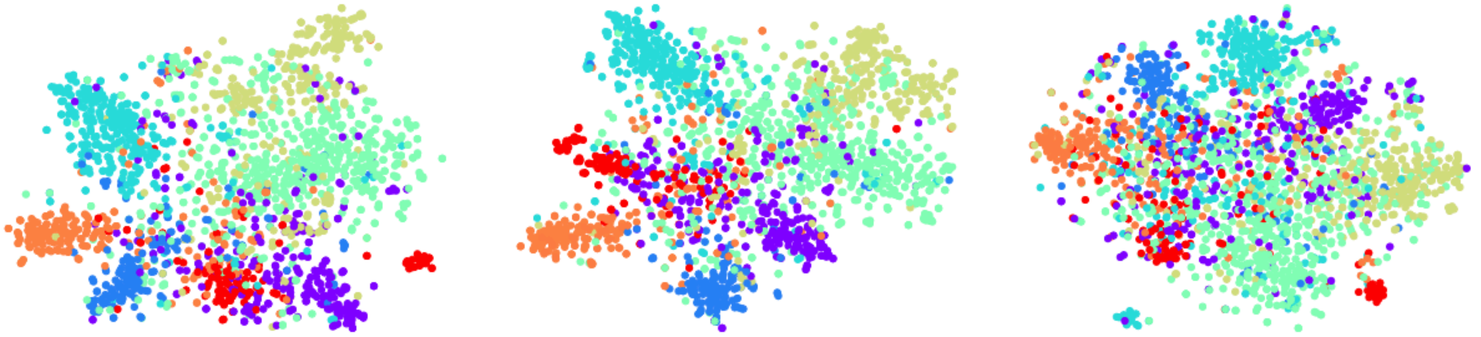
Q: *How does PMLP perform compared with MLP and GNN ?*

	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	**63.74 ± 2.28** (−7.68%/+18.34%)	**62.65 ± 0.35** (−8.62%/+16.32%)	**63.30 ± 0.17** (−8.51%/+17.53%)
Train loss	1.0480	1.1124	1.0396	1.5917	**1.5917**	**1.5917**	**1.5917**
Train time	63.48 ms	90.50 ms	87.24 ms	7.78 ms	**7.78 ms**	**7.78 ms**	**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	**69.71 ± 0.13** (−2.30%/+23.95%)	**70.09 ± 0.13** (−1.52%/+24.63%)	**65.72 ± 0.09** (−6.66%/+16.86%)
Train loss	0.4013	0.4018	0.4311	1.0841	**1.0841**	**1.0841**	**1.0841**
Train time	288.61 ms	665.06 ms	527.26 ms	39.73 ms	**39.73 ms**	**39.73 ms**	**39.73 ms**
Flickr ($\Delta_{GNN}/\Delta_{MLP}$)	49.66 ± 0.57	50.93 ± 0.16	45.31 ± 0.28	46.44 ± 0.14	**49.55 ± 1.30** (−0.22%/+6.70%)	**50.99 ± 0.39** (+0.12%/+9.80%)	**44.31 ± 0.24** (−2.21%/−4.59%)
Train loss	1.1462	1.4030	0.7449	1.3344	**1.3344**	**1.3344**	**1.3344**
Train time	36.82 ms	50.42 ms	163.82 ms	7.44 ms	**7.44 ms**	**7.44 ms**	**7.44 ms**

Table. Experimental results for inductive node classification tasks.

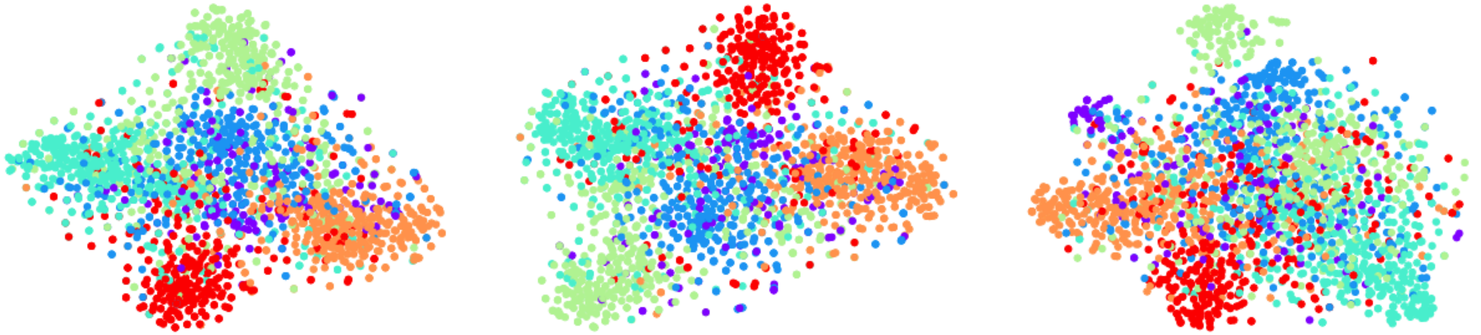
PMLPs do not need graph information in training, and can reduce the training time up to 65 times

Experiments



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

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

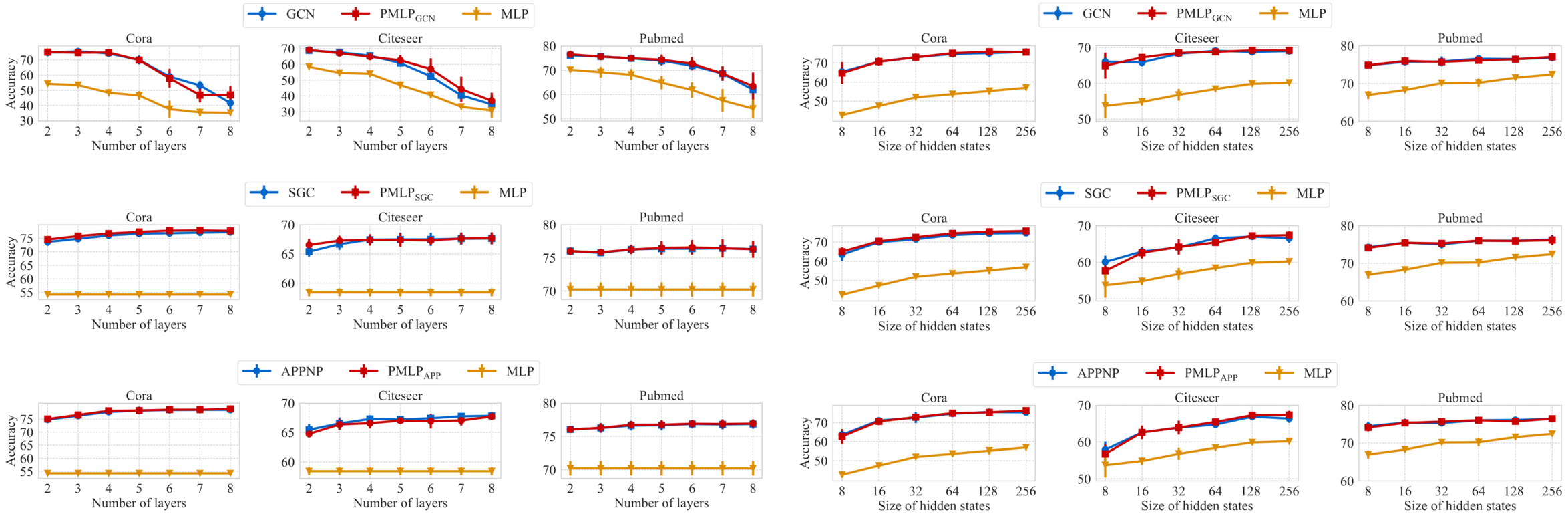


(a) GCN (b) PMLP_{GCN} (c) MLP

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

Experiments

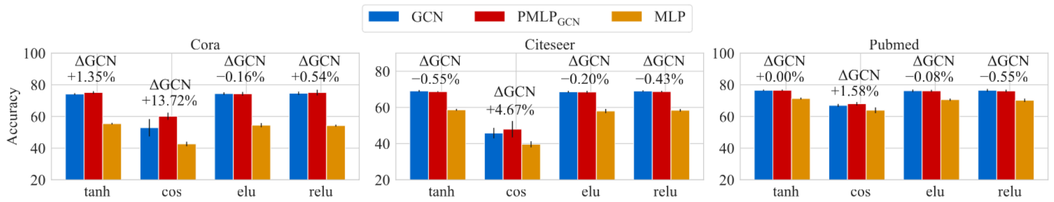
***Q1:** What is the impact of model layers and hidden sizes (hyperparameters)?*



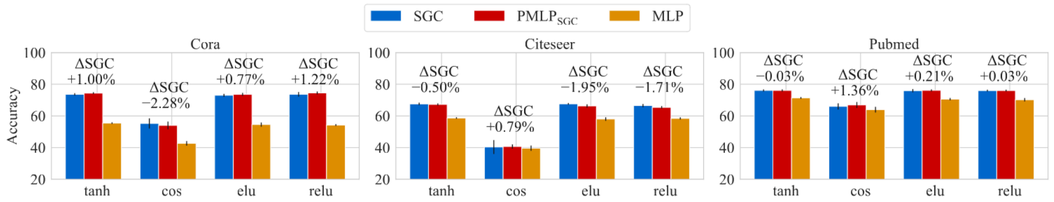
Results still hold across different hyperparameters.

Experiments

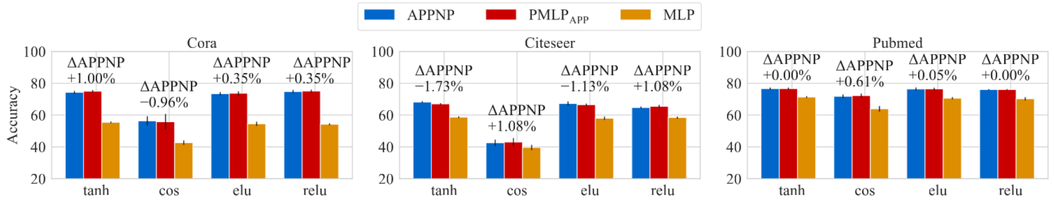
Q2/3: *What is the impact of different* ***activation functions*** *in FF layers /* ***message passing schemes*** *in MP layers?*



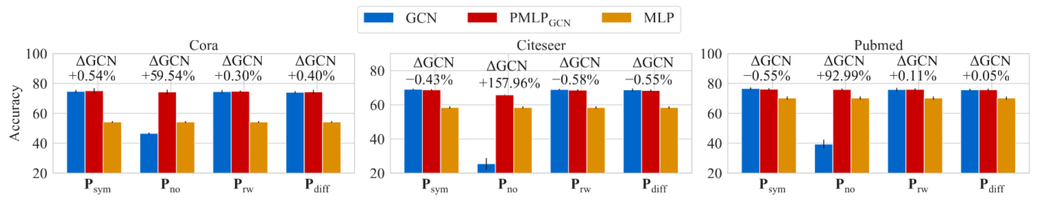
(a) GCN v.s. PMLP$_{GCN}$ v.s. MLP



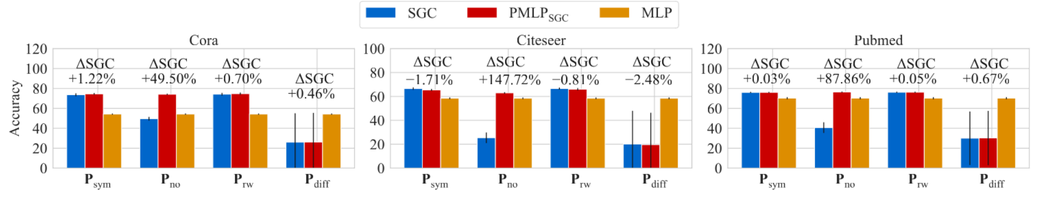
(b) SGC v.s. PMLP$_{SGC}$ v.s. MLP



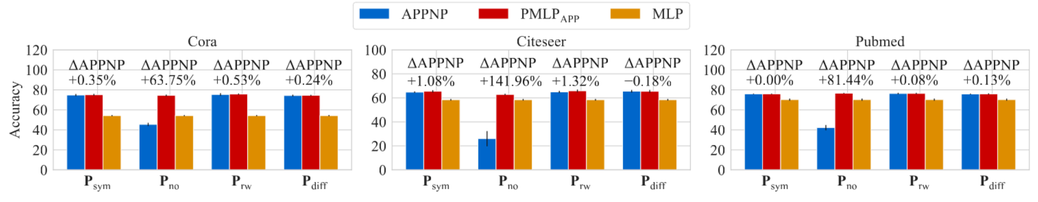
(c) APPNP v.s. PMLP$_{APP}$ v.s. MLP



(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

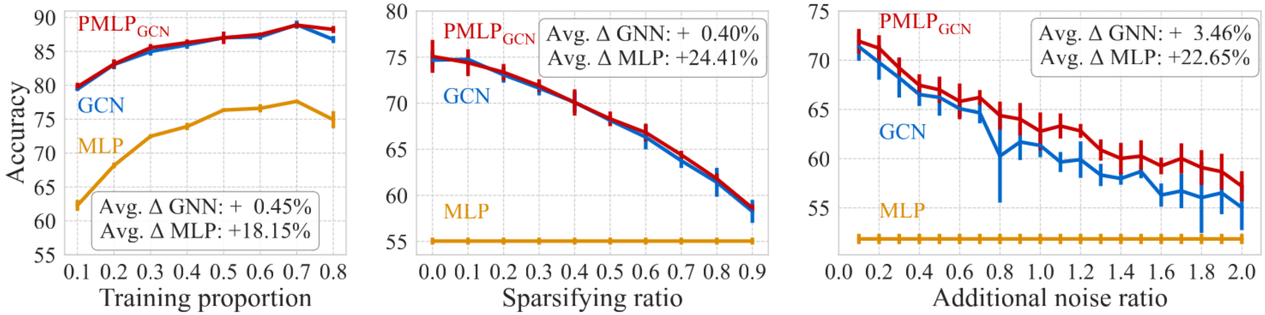
Results still hold across different model implementations.

Experiments

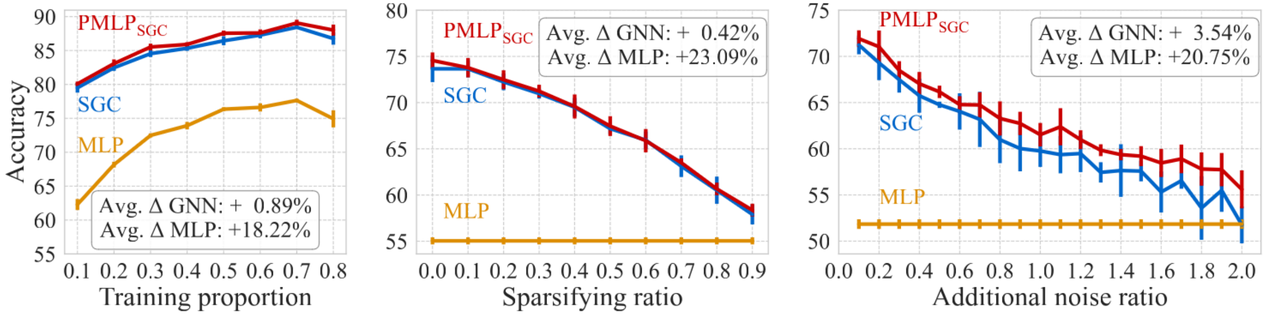
Q4/5/6: *What is the impact of training proportion, graph sparsity and noisy graph structures?*

Results still hold for different graph properties and PMLP is more robust to graph structure noise.

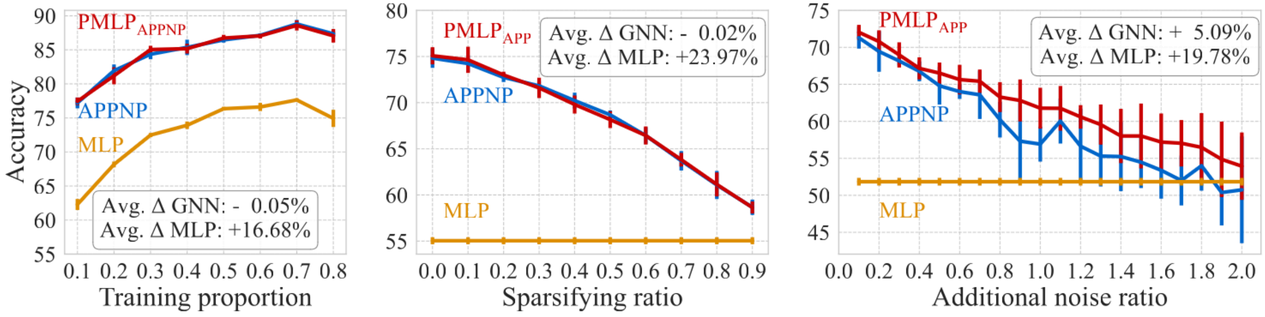
(See more discussions on residual connections, oversmoothing and heterophily in our paper.)



(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

Quick Guide of Implementation

- **PMLP can be implemented in many different ways. The simplest way only requires one line of code.**

```python
# version A: three models (mlp, pmlp, gnn) in one class
class My_GNN(nn.Module):
    ...
    def forward(self, x, edges, use_conv = True):
        ...
        x = self.feed_forward_layer(x)
        if use_conv:
            x = self.message_passing_layer(x, edges)
        ...
        return x

my_gnn = My_GNN()

# in the training loop
my_gnn.train()
for epoch in range(args.epochs):
    prediction = my_gnn(x, edge, use_conv = False)

# for inference
my_gnn.eval()
prediction = my_gnn(x, edge, use_conv = True)
```

```python
# version B: one line of code is all you need
class My_GNN(nn.Module):
    ...
    def forward(self, x, edges):
        ...
        x = self.feed_forward_layer(x)
        if not self.training: # only modified part
            x = self.message_passing_layer(x, edges)
        ...
        return x

my_gnn = My_GNN()

# in the training loop
my_gnn.train()
for epoch in range(args.epochs):
    prediction = my_gnn(x, edges)

# for inference
my_gnn.eval()
prediction = my_gnn(x, edges)
```

Quick Guide of Implementation

- **PMLP can be implemented in many different ways. The simplest way only requires one line of code.**

Q: *What could we do with PMLP?*

- *Accelerate GNN training.*
- *Empower pre-trained NNs with test-time message passing.*
- *Empower GNN in some scenarios.*
- *Simple and useful tool for analytical purposes.*

Quick Guide of Implementation

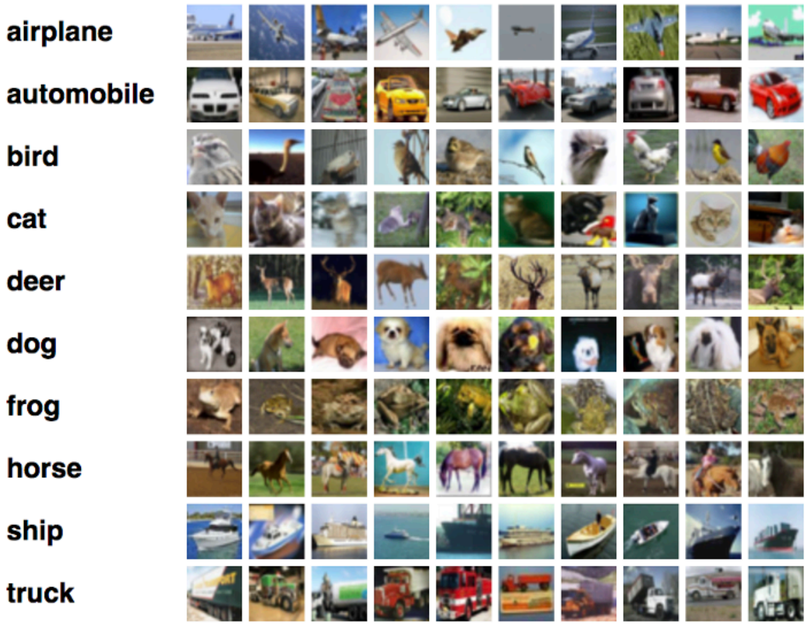
- **PMLP can be implemented in many different ways. The simplest way only requires one line of code.**

Q: *What are potential extensions of PMLP?*

- *Parameterized MP layers, such as GAT.*
- *Transductive / semi-supervised learning settings.*
- *Other tasks (link prediction, graph classification, etc.)*
- *GNN problems (oversmoothing, heterophily, etc.)*

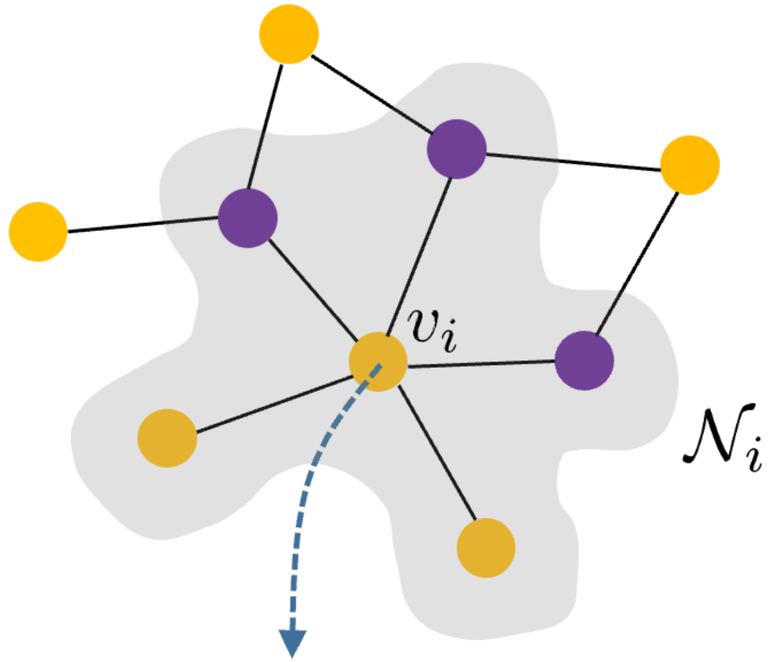
GNNs are Inherently Good Generalizers

Why such results are interesting?



$$(x_i, y_i) \sim p(x, y)$$

each instance is drawed from the same data distribution independently (i.i.d.)



$$(x_i, y_i) \sim p(x, y|\mathcal{N}_i)$$

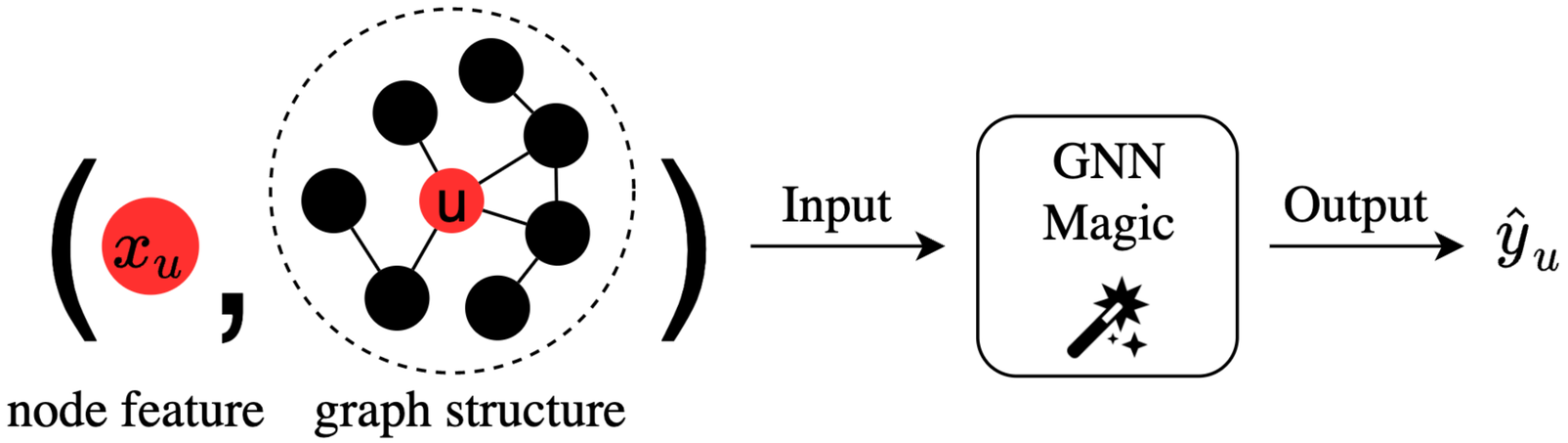
instances have inter-connection and cannot be treated as i.i.d. samples

GNNs are Inherently Good Generalizers

Why such results are interesting?

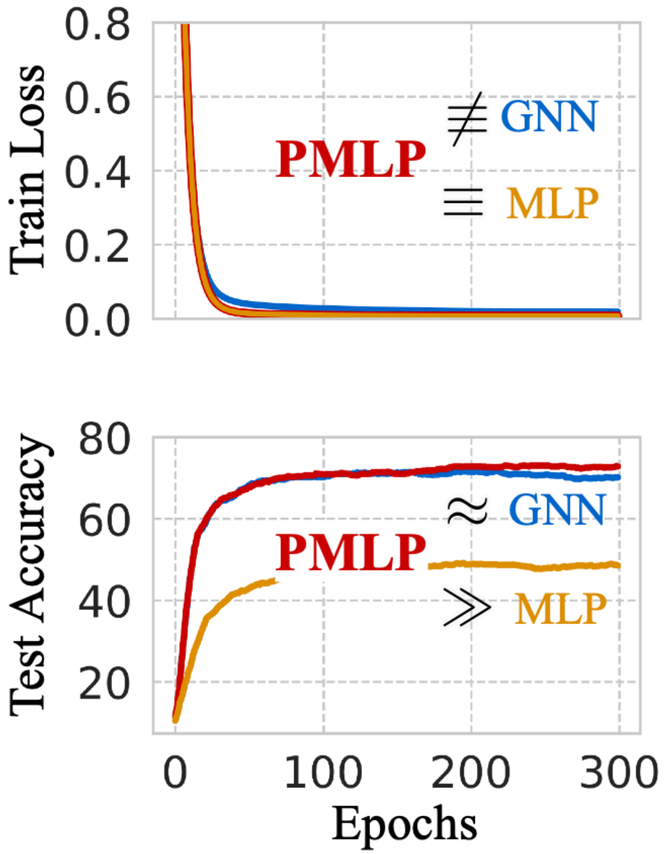
Conventional wisdom :

GNNs are more expressive than MLPs and capable of somehow learning to encode graph information through repeated message passing operations in a way useful for the classification.



This might NOT be the key factor for explaining GNNs' success in node-level tasks.

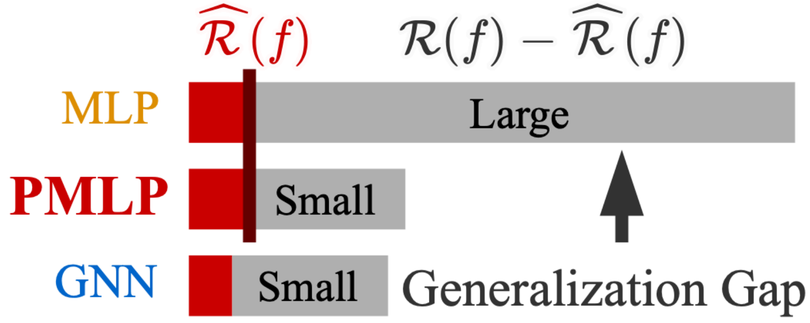
GNNs are Inherently Good Generalizers



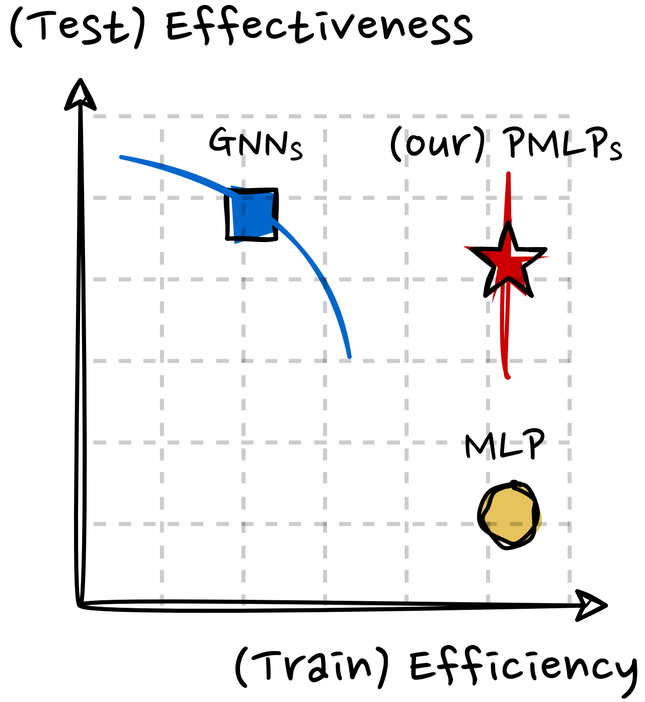
- **Representational Power :** what function class neural networks can approximate and to what extent they can minimize the empirical risk $\widehat{\mathcal{R}}(f)$.

- **Generalization :** how well the learned function can generalize to unseen in- and out-of-distribution samples, reflected by the generalization gap $\mathcal{R}(f) - \widehat{\mathcal{R}}(f)$

The GNN architecture used in inference inherently encourages lower generalization gap.



GNNs are Inherently Good Generalizers



What lies between MLP and GNN?

MLP —large gap→ PMLP :

- Same optimization process and learning dynamics
- Same model expressivity and capability to fit the data
+ Different generalizability due to the GNN architecture in inference (key factor)

PMLP —small gap→ GNN :

- Same GNN architecture in inference
+ Different optimization process and model expressivity

PMLPs pinpoint the major source of performance gap between GNNs and MLPs in node classification stems from **the inherent generalization capability of GNN architectures**.

Effects of Model Architectures on Generalization

- **(Neural Tangent Kernel) NTK perspective on MLP, PMLP and GNN** : NTK perspective allows us to conveniently study the inherent effects of model architectures due to disentanglement of weights and kernel feature map.

random output learned weights

$$f(\boldsymbol{x};\boldsymbol{\theta}) \approx f(\boldsymbol{x};\boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

feature map

As the network width tends to infinity, the feature map becomes constant and is determined by the model architecture (e.g., MLP, GNN and CNN), inducing a kernel called NTK

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

Effects of Model Architectures on Generalization

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 :** PMLP corresponds to transforming the kernel feature map of MLP to that of GNN, while fixing trained MLP weights.

$$f_{mlp}(\mathbf{x}) = \mathbf{w}^{*\top}_{mlp}\, \phi_{mlp}(\mathbf{x}) \qquad f_{pmlp}(\mathbf{x}) = \mathbf{w}^{*\top}_{mlp}\, \phi_{gnn}(\mathbf{x}) \qquad f_{gnn}(\mathbf{x}) = \mathbf{w}^{*\top}_{gnn}\, \phi_{gnn}(\mathbf{x})$$

The key factor of good testing performance

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*

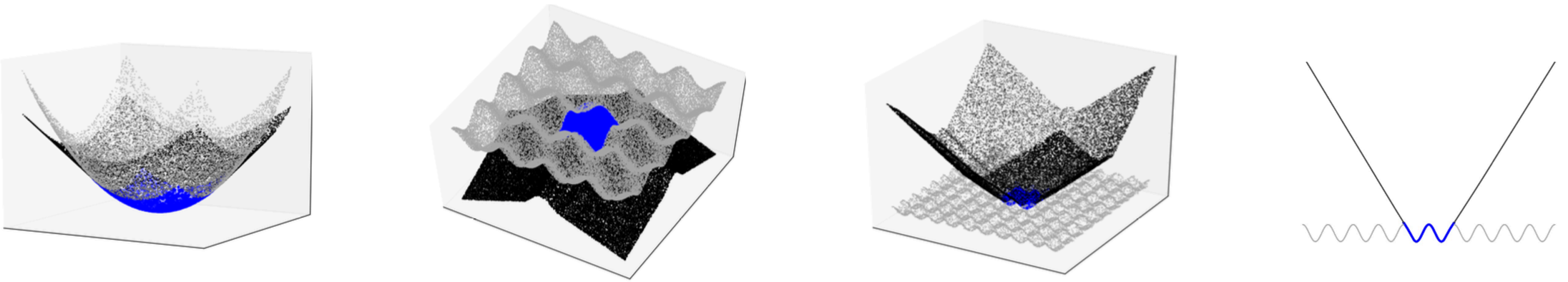
$$\phi_{gnn}(\boldsymbol{x}_i) = c \sum_{j \in \mathcal{N}_i \cup \{i\}} \left[\mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)\top} \mathbf{X}^\top \boldsymbol{a}_j \right), \boldsymbol{w}^{(k)\top} \mathbf{X}^\top \boldsymbol{a}_j \cdot \mathbb{I}_+ \left(\boldsymbol{w}^{(k)\top}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

Effects of Model Architectures on Generalization

Theorem 3. *(Xu et al., 2021). 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*

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

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.



Xu, Keyulu, et al. "How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks." in ICLR 2021.

Effects of Model Architectures on Generalization

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*

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

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.

- **Interpretation**: Similar to MLP, both PMLP and GNN (with sum/mean pooling) converge to linear functions when testing samples are far away from the training data.

Effects of Model Architectures on Generalization

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).

Theorem 3. *(Xu et al., 2021). 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*

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

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.

Effects of Model Architectures on Generalization

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*

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

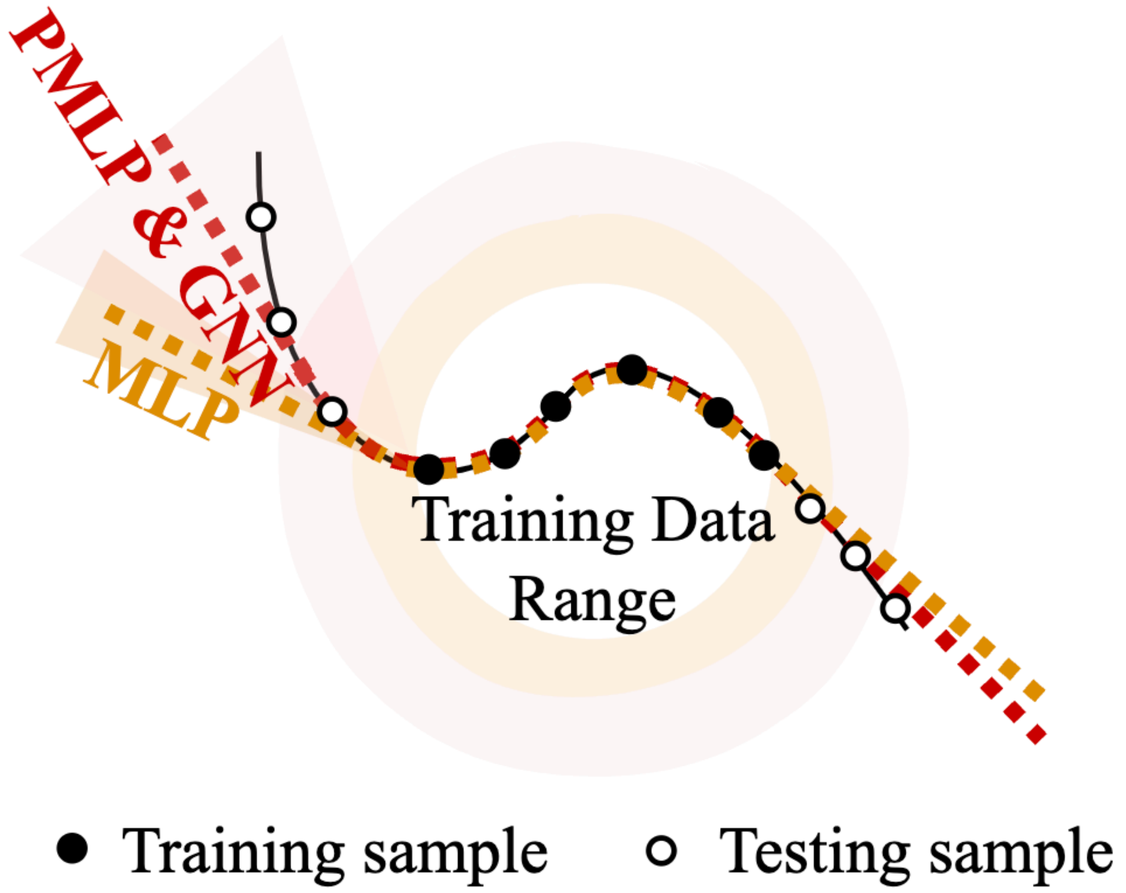
where $\alpha_{min} = \min\{\alpha_i\}_{i \in \mathcal{N}_0 \cup \{0\}} \in [0,1]$, and $\tilde{d}_{max} \geq 1$ denotes the maximum node degree in the testing node $\boldsymbol{x}_0$'s neighbors (including itself).

- **Interpretation**: PMLP and GNN's convergence rates (to linear models) are smaller than that of MLP. This indicates they are less vulnerable to linearalization, and prone to generalize better on testing samples near the training data.

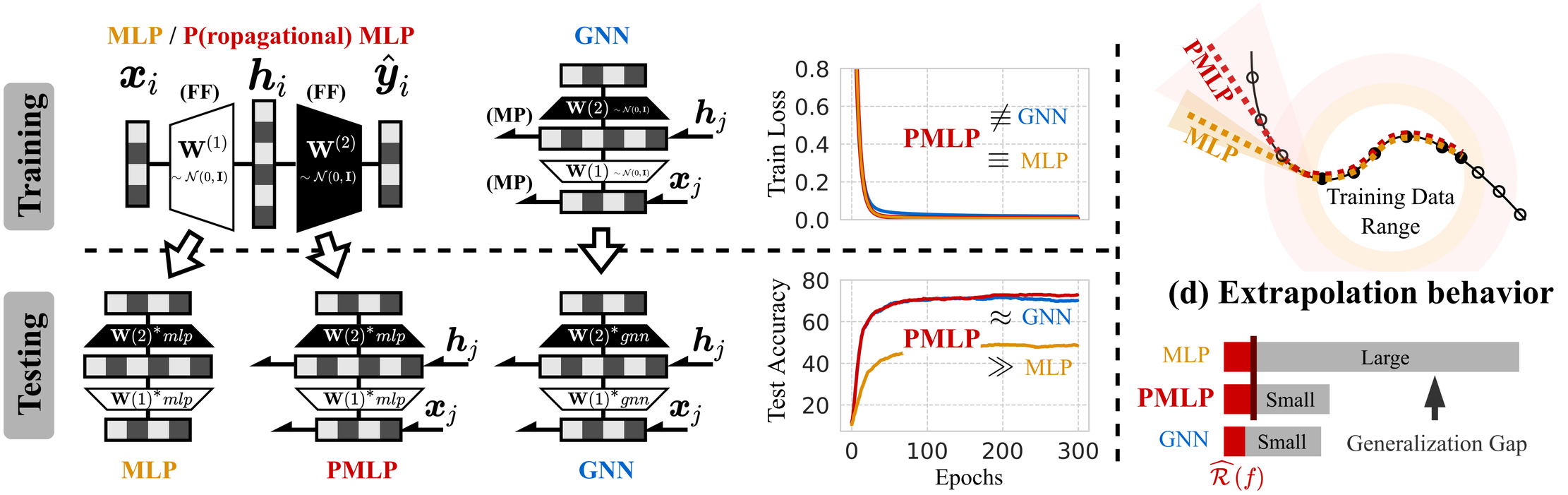
Effects of Model Architectures on Generalization

- **Theoretical Insight (informal)**

  Due to their architectures used in inference, both GNN and PMLP can better extrapolate out-of-distribution testing nodes for node-level tasks.



Thank you !



(a) Illustration of model architecture (b) Learning behavior (c) Intrinsic generalizability of GNN

Code and related materials in https://github.com/chr26195/PMLP