A MORE DETAILS OF PACIA

A.1 ENCODER

Encoder for MoleculeNet. As the main network of the encoder, to process a molecule with GNN, each node embedding $\mathbf{h}_v$ represents an atom, and each edge e_{vu} represents a chemical bond. Here, we use GIN (Xu et al., 2019) as the main network in encoder, which is a powerful GNN structure. In GIN, the aggregation function in (1) is specified as adding all neighbors up: and for the update function is adding the aggregated embeddings and the target node, and feeding to a MLP:

$$\mathbf{h}_v^l = \texttt{MLP}_G^l\left((1+\epsilon)\mathbf{h}_v^{l-1} + \sum\nolimits_{u\in\mathcal{H}(v)} \mathbf{h}_{\text{u}}^{l-1}\right), \tag{14}$$

where ϵ is a scalar parameter to distinguish the target node. To obtain the molecular representation, the readout function in (2) is specified as

$$\mathbf{r} = \texttt{MLP}_R\left(\text{MEAN}(\{\mathbf{h}_v^L | v \in \mathcal{V}\})\right). \tag{15}$$

Encoder for FS-Mol. Following existing works (Chen et al., 2022; Schimunek et al., 2023), we directly adopt the PNA (Corso et al., 2020) network provided in FS-Mol benchmark (Stanley et al., 2021) as the molecular encoder.

A.2 PREDICTOR

The classifier needs to make prediction of the query $\hat{\mathbf{h}}_{\tau,q}$, according to the N_τ labeled support samples $\{(\mathbf{h}_{\tau,s}, y_{\tau,s}) \mid \mathcal{X}_{\tau,s} \in \mathcal{S}_\tau\}$. We adopt an adaptive classifier (Requeima et al., 2019), which map the labeled samples in each class to the parameters of a linear classifier, i.e.,

$$\boldsymbol{w}_\pm = \texttt{MLP}_w\Big(\frac{1}{|\mathcal{S}_\tau^\pm|}\sum\nolimits_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^\pm} \mathbf{h}_{\tau,s}\Big),\ b_\pm = \texttt{MLP}_b\Big(\frac{1}{|\mathcal{S}_\tau^\pm|}\sum\nolimits_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^\pm} \mathbf{h}_{\tau,s}\Big), \tag{16}$$

where $\boldsymbol{w}_\pm$ has the same dimension with $\mathbf{h}_{\tau,q}$, and $b_\pm$ is a scalar. Then the prediction is made by

$$\hat{\boldsymbol{y}}_{\tau,q} = \texttt{softmax}([\boldsymbol{w}_-^\top \mathbf{h}_{\tau,q} + b_-,\ \boldsymbol{w}_+^\top \mathbf{h}_{\tau,q} + b_+]), \tag{17}$$

where $\texttt{softmax}(\boldsymbol{x}) = \exp(\mathbf{x})/\sum_i \exp\left([\boldsymbol{x}]_i\right)$ and $[\boldsymbol{x}]_i$ means the ith element in $\boldsymbol{x}$.

A.3 UNIFIED GNN ADAPTER

The choice of e can be various (Wu et al., 2023), while in this work we adopt a simple feature-wise linear modulation (FiLM) (Perez et al., 2018) function.

A.4 HYPERPARAMETERS

Here we provide the detailed hyperparameter setting of PACIA.

Hyperparameters on MoleculeNet. The maximum layer number of the GNN $L_{\text{enc}} = 5$, the maximum depth of the relation graph $L_{\text{rel}} = 5$, During training, for each layer in GNN, we set dropout rate as 0.5 operated between the graph operation and FiLM layer. The dropout rate of MLP in (9) (10) and (11) is 0.1. For all baselines, we use Adam optimizer (Kingma & Ba, 2015) with learning rate 0.006 and the maximum episode number is 25000. In each episode, the meta-training tasks are learned one-by-one, query set size $M = 16$. The ROC-AUC is evaluated every 10 epochs on meta-testing tasks and the best performance is reported. Table 4 shows the details of the other parts. Experiments are conducted on a 24GB NVIDIA GeForce RTX 3090 GPU, with Python 3.8.13, CUDA version 11.7, Torch version 1.10.1.

Hyperparameters on FS-Mol. The maximum layer number of the GNN $L_{\text{enc}} = 8$, the maximum depth of the relation graph $L_{\text{rel}} = 5$, During training, the dropout rate of MLP_L is 0.1. We use Adam optimizer (Kingma & Ba, 2015) with learning rate 0.0001 and the maximum episode number is 3000. In each episode, the meta-training tasks are learned with batch size 16, support set size $N_\tau = 64$, and the others are used as queries. The average precision s evaluated every 50 epochs on validation tasks and the the model with best validation performance is tested and reported. Table 5 shows the details of the other parts.

Table 4: Details of model structure for MoleculeNet.

	Layers	Output Dimension
MLP in (9)	input $\frac{1}{\lvert\mathcal{V}_{\tau,s}\rvert}\sum_{v\in\mathcal{X}_{\tau,s}}\left[\mathbf{h}_v^l \mid \boldsymbol{y}_{\tau,s}\right]$,, fully connected, LeakyReLU	300
	fully connected with with residual skip connection, $\frac{1}{K}[\sum_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^+}(\cdot) \mid \sum_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^-}(\cdot)]$	300
MLP in (10)	fully connected with residual skip connection	601
MLP in (11)	fully connected with residual skip connection	257
MLP_G^l in (14)	input $(1+\epsilon)\mathbf{h}_v^{l-1}+\mathbf{h}_{\text{agg}}^{l-1}$, fully connected, ReLU	600
	fully connected	300
MLP_L in (15)	input READOUT $\left(\{\mathbf{h}_v^T \mid v\in\mathcal{V}_{t,i}\}\right)$, fully connected, LeakyReLU	128
	fully connected	128
MLP in (4)	input $\exp(\lvert\mathbf{h}_{\tau,i}^{l-1}-\mathbf{h}_{\tau,i}^{l-1}\rvert)$, fully connected, LeakyReLU	256
	fully connected, LeakyReLU	128
	fully connected	1
MLP in (5)	fully connected, LeakyReLU	256
	fully connected, LeakyReLU	128
MLP_w in (16)	input $\frac{1}{K}\sum_{y_{\tau,s}=c}\boldsymbol{h}_{\tau,s}$, fully connected with residual skip connection, LeakyReLU	128
	2× (fully connected with residual skip connection, LeakyReLU)	128
	fully connected	128
MLP_b in (16)	input $\frac{1}{K}\sum_{y_{\tau,s}=c}\boldsymbol{h}_{\tau,s}$, fully connected with residual skip connection, LeakyReLU	128
	2× (fully connected with residual skip connection, LeakyReLU)	128
	fully connected	1

Table 5: Details of model structure for FS-Mol.

	Layers	Output Dimension
MLP in (9)	input $\frac{1}{\lvert\mathcal{V}_{\tau,s}\rvert}\sum_{v\in\mathcal{X}_{\tau,s}}\left[\mathbf{h}_v^l \mid \boldsymbol{y}_{\tau,s}\right]$,, fully connected, LeakyReLU	512
	fully connected with with residual skip connection,$[\frac{1}{\lvert\mathcal{S}_\tau^+\rvert}\sum_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^+}(\cdot) \mid \frac{1}{\lvert\mathcal{S}_\tau^-\rvert}\sum_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau^-}(\cdot)]$	512
MLP in (10)	fully connected with residual skip connection	1025
MLP in (11)	fully connected with residual skip connection	513
MLP in (4)	input $\exp(\lvert\mathbf{h}_{\tau,i}^{l-1}-\mathbf{h}_{\tau,i}^{l-1}\rvert)$, fully connected, LeakyReLU	256
	fully connected, LeakyReLU	128
	fully connected	1
MLP in (5)	fully connected, LeakyReLU	256
	fully connected, LeakyReLU	256
MLP_w in (16)	input $\frac{1}{\lvert\mathcal{S}_\tau^\pm\rvert}\sum_{y_{\tau,s}=c}\boldsymbol{h}_{\tau,s}$, fully connected with residual skip connection, LeakyReLU	256
	2× (fully connected with residual skip connection, LeakyReLU)	256
	fully connected	256
MLP_b in (16)	input $\frac{1}{\lvert\mathcal{S}_\tau^\pm\rvert}\sum_{y_{\tau,s}=c}\boldsymbol{h}_{\tau,s}$, fully connected with residual skip connection, LeakyReLU	256
	2× (fully connected with residual skip connection, LeakyReLU)	256
	fully connected	1

B ADOPTING MAML FOR PROPERTY-LEVEL ADAPTATION

Denote all model parameter as $\boldsymbol{\Theta}$. The model first predict samples in support set and get loss to do local-update. Denote the loss for local-update as $\mathcal{L}_\tau^S(\boldsymbol{\Theta}) = \sum_{\mathcal{X}_{\tau,s}\in\mathcal{S}_\tau}\boldsymbol{y}_{\tau,s}^\top\log\left(\hat{\boldsymbol{y}}_{\tau,s}\right)$, where $\hat{\boldsymbol{y}}_{\tau,s}$ is the prediction made by the main network with parameter $\boldsymbol{\Theta}$. The loss for global-update is

calculated with samples in query set, denoted as , $\mathcal{L}_\tau^Q(\mathbf{\Theta}'_\tau) = \sum_{\mathcal{X}_{\tau,q} \in \mathcal{Q}_\tau} \boldsymbol{y}_{\tau,q}^\top \log(\hat{\boldsymbol{y}}_{\tau,q})$, where $\hat{\boldsymbol{y}}_{\tau,q}$ is the prediction made by the main network with parameter $\mathbf{\Theta}'_\tau$. Then Algorithm 3 can be adopted for meta-training, and Algorithm 4 for meta-testing.

Algorithm 3 Meta-training with MAML

Input: meta-training task set $\boldsymbol{\mathcal{T}}_{\text{train}}$
1: initialize $\mathbf{\Theta}$ randomly;
2: **while** not done **do**
3: **for** each task $\mathcal{T}_\tau \in \boldsymbol{\mathcal{T}}_{\text{train}}$ **do**
4: evaluate $\nabla_{\mathbf{\Theta}} \mathcal{L}_\tau^S(\mathbf{\Theta})$ with respect to all samples in $\mathcal{S}_\tau$;
5: compute adapted parameters with gradient descent: $\mathbf{\Theta}'_\tau = \mathbf{\Theta} - \nabla_{\mathbf{\Theta}} \mathcal{L}_\tau^S(\mathbf{\Theta})$;
6: **end for**
7: update $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \nabla_{\mathbf{\Theta}} \sum_{\mathcal{T}_\tau \in \boldsymbol{\mathcal{T}}_{\text{train}}} \mathcal{L}_\tau^Q(\mathbf{\Theta}'_\tau)$;
8: **end while**
9: **return** learned $\mathbf{\Theta}^*$.

Algorithm 4 Meta-testing with MAML

Input: learned $\mathbf{\Theta}*$, a meta-testing task $\mathcal{T}_\tau$;
1: evaluate $\nabla_{\mathbf{\Theta}} \mathcal{L}_\tau^S(\mathbf{\Theta})$ with respect to all samples in $\mathcal{S}_\tau$;
2: compute adapted parameters with gradient descent: $\mathbf{\Theta}'_\tau = \mathbf{\Theta} - \nabla_{\mathbf{\Theta}} \mathcal{L}_\tau^S(\mathbf{\Theta})$;
3: make prediction $\boldsymbol{y}_{\tau,q}$ for $\mathcal{X}_{\tau,q} \in \mathcal{Q}_\tau$ with adapted parameter $\mathbf{\Theta}'_\tau$;

B.1 Comparison with Existing Works

We compare the proposed PACIA with existing few-shot MPP approaches in Table 6. As shown, we manage to compare in perspectives of support of pre-training, property-level adaptation, molecule-level adaptation. fast-adaptation and adaptation strategy. With the help of hypernetworks, our method not only introduces novel molecule-level adaptation, but also can adapt on property-level more effectively and efficiently.

Table 6: Comparison of the proposed PACIA with existing few-shot MPP methods.

Method	Support Pre-training	Hierarchical adaptation Property-level	Hierarchical adaptation molecule-level	Fast adaptation	Adaptation Strategy
IterRefLSTM	×	√	×	√	Pair-wise similarity
Meta-MGNN	√	√	×	×	Gradient
PAR	√	√	×	×	Attention+Gradient
ADKF-IFT	√	√	×	×	Gradient+statistical learning
MHNfs	√	√	√	√	Attention+pair-wise similarity
GS-META	√	×	√	√	Message passing
PACIA	√	√	√	√	Hypernetwork

From the perspective of hypernetwork, the usage of the hypernetwork for encoder is related to GNN-FiLM (Brockschmidt, 2020), which considers a GNN as main network. It builds hypernetwork with target node as input to generate parameters of FiLM layers, to equip different nodes with different aggregation functions in the GNN. What and how to adapt are similar to ours, but it is different that the input of our hypernetwork for encoder is $\mathcal{S}_\tau$ and how we encode a set of labeled graphs.

In few-shot learning, some recent works (Requeima et al., 2019; Lin et al., 2021) use hypernetworks to process the task context to make the model task-adaptive. Their hypernetworks have similar functionality of our hypernetwork for encoder, that are used to map the support set to parameter to modulate the main network, but their main networks are convolutional neural network (CNN) and MLP respectively, there is significant difference about what to modulate. As for the usage of the hypernetwork for predictor, which is used to evaluate an unlabeled sample with a set of labeled ones to encode model architecture, did not appear in the literature.

C More Details of Experiments

C.1 Datasets

MoleculeNet. There are four sub-datasets for few-shot MPP: Tox21 (National Center for Advancing Translational Sciences, 2017), SIDER (Kuhn et al., 2016), MUV (Rohrer & Baumann, 2009) and ToxCast (Richard et al., 2016), which are included in MoleculeNet (Wu et al., 2018). We adopt the task splits provided by existing works(Altae-Tran et al., 2017; Wang et al., 2021). Tox21 is a collection of nuclear receptor assays related to human toxicity, containing 8014 compounds in 12 tasks, among which 9 are split for training and 3 are split for testing. SIDER collects information about side effects of marketed medicines, and it contains 1427 compounds in 21 tasks, among which 21 are split for training and 6 are split for testing. MUV contains compounds designed to be challenging for virtual screening for 17 assays, containing 93127 compounds in 17 tasks, among which 12 are split for training and 5 are split for testing. ToxCast collects compounds with toxicity labels, containing 8615 compounds in 617 tasks, among which 450 are split for training and 167 are split for testing.

FS-Mol. FS-Mol benchmark, which contains a set of few-shot learning tasks for molecular property prediction carefully collected from ChEMBL27 (Mendez et al., 2019) by Stanley et al. (2021). Following existing works (Chen et al., 2022; Schimunek et al., 2023), we use the same 10% of all tasks which contains 233,786 unique compounds, split into training (4,938 tasks), validation (40 tasks), and test (157 tasks) sets. Each task is associated with a protein target.

C.2 Baselines

We compare our method with following baselines:

- **Siamese** (Koch et al., 2015): It learns two neural networks which are symmetric on structure to identity whether the input molecule pairs are from the same class. The performance is copied from (Altae-Tran et al., 2017) due to the lack of code.
- **ProtoNet**[1] (Snell et al., 2017): It makes classification according to inner-product similarity between the target and the prototype of each class. This method is incorporated as a classifier after the GNN encoder.
- **MAML**[2] (Finn et al., 2017): It learns a parameter initialization and the model is adapted to each task via few gradient steps on the support set. We adopt this method for all parameters in a model composed of a GNN encoder and a linear classifier.
- **EGNN**[3] (Kim et al., 2019): It builds a relation graph that samples are refined, and it learns to predict edge-labels in the relation graph. This method is incorporated as the predictor after the GNN encoder.
- **GNN-FiLM**(Brockschmidt, 2020): GNN-FiLM has built hypernetwork with target node as input to generate parameters of FiLM layers, which equips different nodes with different aggregation functions in the GNN. We adopt this as encoder and a MLP as classifier and train on all samples in meta-training tasks and samples in support set of all meta-testing tasks;
- **IterRefLSTM** (Altae-Tran et al., 2017): It introduces matching networks combined with long short-term memory (LSTM) to refine the molecular representations according to the task context. The performance is copied from (Altae-Tran et al., 2017) due to the lack of code.
- **PAR**[4] (Wang et al., 2021): It introduces an attention mechanism to capture task-dependent property and an inductive relation graph between samples, and incorporates MAML to train.
- **ADKF-IFT**[5] (Wang et al., 2021): It adopt gradient-based strategy to learn the encoder where it proposes Implicit Function Theory to avoid computing the hyper-gradient. And a Gaussian Process is learned from scratch in each task as classifier.

[1] https://github.com/jakesnell/prototypical-networks
[2] https://github.com/learnables/learn2learn
[3] https://github.com/khy0809/fewshot-egnn
[4] https://github.com/tata1661/PAR-NeurIPS21
[5] https://github.com/Wenlin-Chen/ADKF-IFT

- **Pre-GNN**[6] (Hu et al., 2019): It trains a GNN encoder on ZINC15 dataset with graph-level and node-level self-supervised tasks, and fine-tunes the pre-trained GNN on downstream tasks. We adopt the pre-trained GNN encoder and a linear classifier.
- **GraphLoG**[7] (Xu et al., 2021): It introduces hierarchical prototypes to capture the global semantic clusters. And adopts an online expectation-maximization algorithm to learn. We adopt the pre-trained GNN encoder and a linear classifier.
- **MGSSL**[8] (Hu et al., 2019): It trains a GNN encoder on ZINC15 dataset with graph-level, node-level and motif-level self-supervised tasks, and fine-tunes the pre-trained GNN on downstream tasks. We adopt the pre-trained GNN encoder and a linear classifier.
- **GraphMAE**[9] (Hou et al., 2022): It presents a masked graph autoencoder for generative self-supervised graph pre-training and focus on feature reconstruction with both a masking strategy and scaled cosine error. We adopt the pre-trained GNN encoder and a linear classifier.
- **Meta-MGNN**[10] (Guo et al., 2021): It incorporates self-supervised tasks such as bond reconstruction and atom type prediction to be jointly optimized via MAML. It uses the pre-trained GNN encoder provided by (Hu et al., 2019).
- **Pre-PAR**: The same as PAR but uses the pre-trained GNN encoder provided by (Hu et al., 2019).
- **Pre-ADKF-IFT**: The same as ADKF-IFT but uses the pre-trained GNN encoder provided by (Hu et al., 2019).

C.3 Performance Comparison with Pre-training

Baselines with Pre-training. We compare with the following baselines with (w/) pre-training: (i) Methods which fine-tune pre-train GNN encoders, including **Pre-GNN** (Hu et al., 2019), **GraphLoG** (Xu et al., 2021), **MGSSL** (Zhang et al., 2021), **GraphMAE** (Hou et al., 2022); (ii) Few-shot MPP methods incorporating pre-trained encoders provided by (Hu et al., 2019), including **Meta-MGNN** (Guo et al., 2021), **Pre-PAR** (Wang et al., 2021) and **Pre-ADKF-IFT** Chen et al. (2022). All encoders have the same structure (Hu et al., 2019) and are pre-trained on ZINC15 dataset (Sterling & Irwin, 2015). We equip our PACIA with the same pre-trained encoder, and name it as **Pre-PACIA**.

Table 7: Test ROC-AUC obtained with pre-trained GNN encoder.

Method	Tox21		SIDER		MUV		ToxCast	
	10-shot	1-shot	10-shot	1-shot	10-shot	1-shot	10-shot	1-shot
Pre-GNN	$83.02_{(0.13)}$	$82.75_{(0.09)}$	$77.55_{(0.14)}$	$67.34_{(0.30)}$	$67.22_{(2.16)}$	$65.79_{(1.68)}$	$73.03_{(0.67)}$	$71.26_{(0.85)}$
GraphLoG	$81.61_{(0.35)}$	$79.23_{(0.93)}$	$75.18_{(0.27)}$	$67.52_{(1.40)}$	$67.83_{(1.65)}$	$66.56_{(1.46)}$	$73.92_{(0.15)}$	$73.10_{(0.39)}$
MGSSL	$83.24_{(0.09)}$	$\underline{83.21}_{(0.12)}$	$77.87_{(0.18)}$	$69.66_{(0.21)}$	$68.58_{(1.32)}$	$66.93_{(1.74)}$	$73.51_{(0.45)}$	$72.89_{(0.63)}$
GraphMAE	$84.01_{(0.27)}$	$81.54_{(0.18)}$	$76.07_{(0.15)}$	$67.60_{(0.38)}$	$67.99_{(1.28)}$	$\underline{67.50}_{(2.12)}$	$74.15_{(0.33)}$	$72.67_{(0.71)}$
Meta-MGNN	$83.44_{(0.14)}$	$82.67_{(0.20)}$	$77.84_{(0.34)}$	$74.62_{(0.41)}$	$68.31_{(3.06)}$	$66.10_{(3.98)}$	$74.69_{(0.57)}$	$73.29_{(0.85)}$
Pre-PAR	$84.95_{(0.24)}$	$\underline{83.01}_{(0.28)}$	$\underline{78.05}_{(0.15)}$	$\underline{75.29}_{(0.32)}$	$69.88_{(1.57)}$	$66.96_{(2.63)}$	$75.48_{(0.99)}$	$\underline{73.90}_{(1.21)}$
Pre-ADKF-IFT	$\underline{86.06}_{(0.35)}$	$80.97_{(0.48)}$	$70.95_{(0.60)}$	$62.16_{(1.03)}$	$\mathbf{95.74}_{(0.37)}$	$67.25_{(3.87)}$	$\mathbf{76.22}_{(0.13)}$	$71.13_{(1.15)}$
Pre-PACIA	$\mathbf{86.40}_{(\mathbf{0.27})}$	$\mathbf{84.35}_{(\mathbf{0.14})}$	$\mathbf{83.97}_{(\mathbf{0.22})}$	$\mathbf{80.70}_{(\mathbf{0.28})}$	$\underline{73.43}_{(1.96)}$	$\mathbf{69.26}_{(\mathbf{2.35})}$	$\mathbf{76.22}_{(\mathbf{0.73})}$	$\mathbf{75.09}_{(\mathbf{0.95})}$

Performance with Pre-training. Table 7 shows the results. We can see that Pre-PACIA obtains significantly better performance except the 10-shot case on MUV, surpassing the second-best method Pre-ADKF-IFT by 3.10%. MGSSL defeats the other methods which fine-tune pre-trained GNN encoders, i.e., Pre-GNN, GraphLoG, and GraphMAE. However, it still performs worse than Pre-PACIA equipped with Pre-GNN, which validates the necessity of designing a few-shot MPP method

[6] http://snap.stanford.edu/gnn-pretrain
[7] http://proceedings.mlr.press/v139/xu21g/xu21g-supp.zip
[8] https://github.com/zaixizhang/MGSSL
[9] https://github.com/THUDM/GraphMAE
[10] https://github.com/zhichunguo/Meta-Meta-MGNN

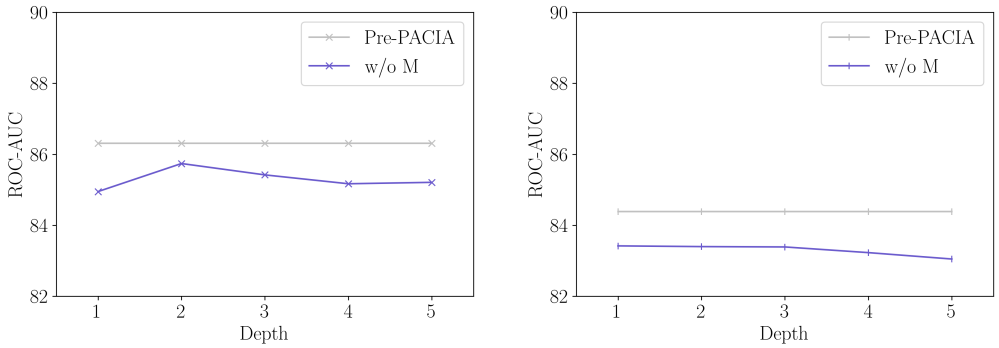
instead of simply fine-tuning a pre-trained GNN encoder. Moreover, comparing Pre-PACIA and PACIA in Table 1, the pre-trained encoder brings 3.05% improvement in average performance due to a better starting point of learning.

C.4 A Closer Look at Molecule-level Adaptation

In this section, we pay a closer look at our molecule-level adaptation mechanism, proving evidence of its effectiveness.

C.4.1 Performance under Different Propagation Depth

Figure 5 compares Pre-PACIA with "w/o M" (introduced in Section 5.3) using different fixed layers of relation graph refinement on Tox21, where the maximum Depth$L = 5$. As can be seen, Pre-PACIA equipped performs much better than "w/o M" which takes the same depth of relation graph refinement as in PAR. This validates the necessity of molecule-level adaptation.



(a) 10-shot case.

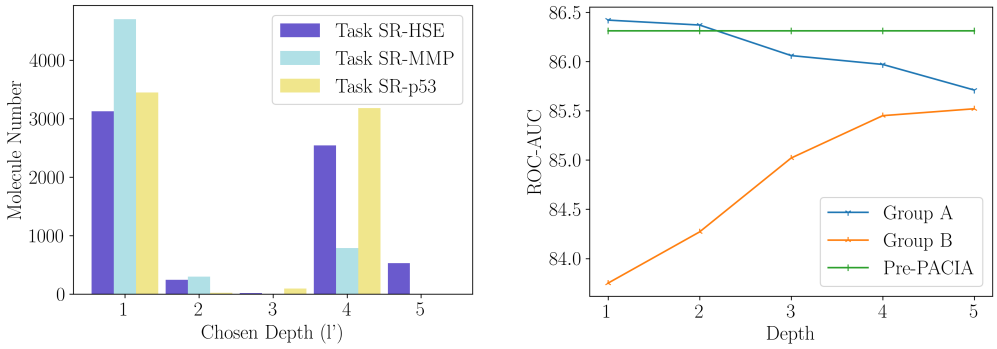
(b) 1-shot case.

Figure 5: Comparing Pre-PACIA with "w/o M" using different fixed propagation depth of relation graph on Tox21.

C.4.2 Distribution of Propagation Depth

Figure 6(a) plots the distribution of learned l' for query molecules in meta-testing tasks for 10-shot case of Tox21. The three meta-testing tasks contain different number of query molecules in scale: 6447 in task SR-HSE, 5790 in task SR-MMP, and 6754 in task SR-p53. We can see that Pre-PACIA choose different tl for query molecules in the same task. Besides, the distribution of learned l' varies across different meta-testing tasks: molecules in task SR-MPP mainly choose smaller depth while molecules in the other two tasks tend to choose greater depth. This can be explained as most molecules in task SR-MPP are relatively easy to classify, which is consistent with the fact that Pre-PACIA obtains the highest ROC-AUC on SR-MPP among the three meta-testing tasks (83.75 for SR-HSE, 88.79 for SR-MPP and 86.39 for SR-p53).

Further, we pick out molecules with $l' = 1$ (denote as **Group A**) and $l' = 4$ (denote as **Group B**) as they are more extreme cases. We then apply "w/o M" with different fixed depth for Group A and Group B, and compare them with Pre-PACIA. Figure 6(b) shows the results. Different observations can be made for these two groups. Molecules in Group A have good performance with smaller depth relation graph, they can achieve higher ROC-AUC score than the average of all molecules using Pre-PACIA. These indicate they are easier to classify and it is reasonable that Pre-PACIA choose $l' = 1$ for them. While molecules in Group B are harder to classify and requires $l' = 4$.



(a) Distribution of learned l'.

(b) Performance comparison.

Figure 6: Examine molecule-level adaptation of Pre-PACIA on 10-shot tasks of Tox21.