Deep Graph Reprogramming
– *Supplementary Material* –

Yongcheng Jing[1], Chongbin Yuan[2], Li Ju[2], Yiding Yang[2], Xinchao Wang[2], Dacheng Tao[1]
[1]The University of Sydney, Australia, [2]National University of Singapore, Singapore
xinchao@nus.edu.sg, dacheng.tao@gmail.com

Summary

※ We provide in this document supporting materials that cannot fit into the manuscript due to the page limit, including more details of the method pre-analysis, more implementation details of the proposed approaches, and various ablation studies for the experiments in the main manuscript. In particular, we provide here **additional results on the fourteen datasets** in the main manuscript.

※ The **organizations** of this supplementary material are summarized as follows.
▸ **Sec. 1:** We begin in Sec. 1 by giving more details and explanations of *Method Pre-analysis*, corresponding to **Sec. 3.3 of the main manuscript**.
▸ **Sec. 2:** Also, we give in Sec. 2 more descriptions of the various datasets used in the main manuscript and this supplement, mentioned in **Sec. 5.1 of the main manuscript**.
▸ **Secs. 3, 4, 5, 6, 7, and 8:** Furthermore, in the following sections, we provide more implementation details, perform extensive ablation studies including various network architectures and heterogeneous types of GNNs, and demonstrate more visualization results to validate the effectiveness of the proposed series of complementary approaches for GARE.

Table of Contents

※ **The table of contents**, as well as the **lists of figures and tables** in this supplementary material are provided below, for the sake of easy reference.

Contents

List of Figures

List of Tables

1. More Details of Method Pre-analysis

We provide in this section more details and discussions on the method pre-analysis section in Sect. 3.3.1 and Sect. 3.3.2 of the main manuscript, including the validation of adversarial reprogramming attacks on graph data, and the rationale of using task-adaptive aggregators in model reusing.

1.1. Adversarial Reprogramming Attacks on Graph Data

In this section, we further explain and validate Remark. 1 in the main paper, which indicates the existence of adversarial reprogramming attacks on topological graphs:

Remark 1 (Adversarial Reprogramming Attacks on Graph Data)**.** *Graph neural networks are susceptible to adversarial reprogramming attacks, where an adversarial perturbation on graph data can readily repurpose a graph neural network to perform a task chosen by the adversary, without notifying the model provider.*

To further illustrate Remark. 1, we demonstrate in Fig. S1 an example adversarial reprogramming attack with graph data as inputs, where an attack redirects a machine-learning paper classification model to handle the intended topological shape recognition task, without informing the model provider, corresponding to the example in Tab. 1 of the main manuscript.

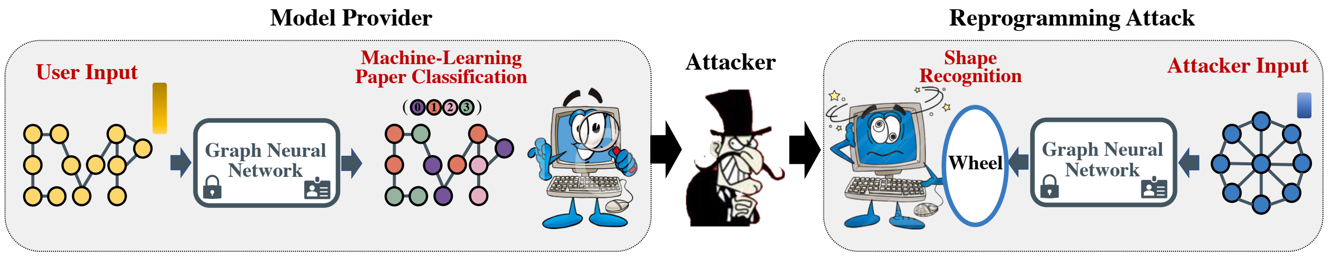

Figure S1. Illustrations of adversarial reprogramming attacks on graph data, corresponding to Tab. 1 of the main paper.

Here, in Tab. S1, we show the results of the example attack in Fig. S1 with detailed explanations in the captions. Specifically, to obtain the pre-trained node property prediction model in Tab. S1, we use the `Cora` dataset [8] in training, with the detailed architectures provided in Tab. S2. For the attacker's designated task of topological shape recognition, we instead use a generated `MiniGC-Dataset` provided in the deep graph library [14]. The method of generating adversarial examples here is the node-level perturbation mentioned in [9], where we specifically add the generated adversarial perturbations to the raw node features.

Table S1. Results of adversarial reprogramming attacks on graph data, where the adversary repurposes a node classification model from the model provider to perform the adversary's designated shape recognition task.

Designated Tasks	**Before Attack**	**After Attack**	**Re-train from Scratch**
Node Property Prediction (*Model Provider*)	Acc: 87.69%	-	Acc: 87.69%
Shape Recognition (***Adversarial Attacker***)	-	**Acc: 80.00%**	Acc: 87.00%

From Tab. S1, it is noticeable that the original model provided by the model provider is easily repurposed to tackle the unseen graph-level task of shape recognition, whereas the original model function is to handle a node-level classification task. The results on shape recognition are also promising, with an accuracy of 80%, which is on par with that of re-training from scratch shown in the last column of Tab. S1. We also perform the experiment of directly feeding the raw data of shape recognition, without adding adversarial perturbation, to the pre-trained model. Expectedly, the accuracy of such a vanilla manner is only 0.16%, which demonstrates the effectiveness of the adversarial perturbations.

Table S2. Detailed architectures used in Tab. S1 of this supplementary material.

Architecture	**Layers**	**Attention Heads**	**Input**	**Hidden**	**Output**
Pre-trained Model {`Cora`}	2	{8, 1}	1433	8	7

In aggregate, the observation in Tab. S1 validates that adversarial reprogramming attacks not only exist in the Euclidean image domain, but are also effective in the non-Euclidean graph domain. This motivates our idea that flips the role of adversarial reprogramming attack on its head, by paradoxically converting their function as threats to machine learning systems to resource-efficient model reusing where only limited pre-trained models are available.

1.2. Aggregation Matters for Reusing

We provide in this section more explanations and discussions on Remark. 2 of the main manuscript, which suggest the importance of adaptive aggregation methods in reusing GNNs:

Remark 2 (Aggregation Matters for Reusing)**.** *Various aggregators lead to diversified downstream task performance with the same model. There exists an optimal aggregation method tailored for each pair of downstream tasks and pre-trained models.*

To validate Remark 2, we perform a pilot experiment, by dividing the `Cora` dataset into two subsets, with the first subset containing four classes and the second one including three node categories. We pre-train a model on the first `Cora` subset and obtain a frozen GNN that can predict the first four classes in `Cora`. Then, we aim to reuse this pre-trained model to handle the task of node classification with the last three separate and unseen categories. The network architecture of the pre-trained model on the first subset of `Cora` is demonstrated in Tab. S3.

Table S3. Detailed architectures used in the section of "Rationale Behind MERE", corresponding to Fig. 2 of the main paper.

Architecture	**Layers**	**Attention Heads**	**Input**	**Hidden**	**Output**
Pre-trained Model {`Cora-subset`}	2	{8, 1}	1433	8	4

Here, our goal is to validate the influence of aggregators in model reusing. As such, we adopt the simplest vanilla reusing method, by just using the pre-trained model to directly handle the novel three downstream categories, but changing the aggregation behaviors. The corresponding results of various aggregation methods are shown in Tab. S4, where we specifically use eight prevalent aggregation methods as examples as mentioned in [3].

Table S4. Vanilla GNN reusing results on the `Cora-subset` dataset with various aggregation behaviors. The detailed network architecture for producing the results can be found in Tab. S3.

Various Aggregation Methods	Mean	Max	Min	Std	Var	Skewness	Kurtosis	Hyperskewness
Downstream Performance (Acc)	0.4420	0.4203	0.5000	0.2536	0.2826	0.4022	0.2572	0.3696

Notably, the results in Tab. S4 show that various aggregation manners can lead to totally distinct model reusing results, where the min aggregation method is optimal for our task of `Cora-subset`. Such observation leads to our idea of using the task-adaptive aggregation method to enhance the model capability in different downstream tasks.

2. Dataset Statistics and Descriptions

We provide in Tab. S5 the statistics of several graph benchmarks used in the main manuscript.

Specifically, the first three datasets, *i.e.*, `Cora`, `Citeseer` and `Pubmed` [8], are all citation network datasets used for single-label node classification. Amazon Computers and Amazon Photo are the segments of the Amazon co-purchase graphs from [6]. Moreover, `ogbn-arxiv` dataset contains a directed graph, which denotes the citation network among all Computer Science (CS) papers in arXiv. `ogbg-molesol`, `ogbg-molbace`, and `ogbg-molbbbp` are all molecular property prediction datasets. Besides, the `QM7b` dataset also aims at molecular property regression, comprising 7,211 molecules with totally 14 regression targets, including E (PBE0), LUMO (PBE0), α (PBE0), LUMO (ZINDO), α (SCS), IP (ZINDO), HOMO (GW), EA (ZINDO), HOMO (PBE0), $E^*_{1\text{st}}$ (ZINDO), HOMO (ZINDO), E^*_{max} (ZINDO), LUMO (GW), and I_{max} (ZINDO). The `PROTEINS` dataset, on the other hand, focuses on protein classification, such as enzymes or non-enzymes.

We also use a distributed human action recognition dataset using wearable motion sensor networks, termed as `WARD`, to validate the proposed methods. There are five sensors used to capture the data in `WARD`. Every sensor generates 5 data streams and in total 5×5 data streams are produced. We will elaborate in Sect. 8 on how to construct graphs from the raw data and convert the problem of action recognition into that of subgraph classification.

For the task of point cloud classification, we adopt the `ModelNet40` dataset [17] as well as the `ShapeNet` part dataset [19]. Specifically, the `ModelNet40` dataset contains 12,311 CAD models of 40 man-made object categories, of which

Table S5. Summary of the fourteen datasets used in the main manuscript and supplementary material. Additional dataset statistics for point cloud classification are shown in Tab. S6.

Names	Task Descriptions	Feature Dimensions	Nodes	Edges	# Graphs
1. Cora [8]	Machine-Learning Paper Classification	1,433	2,708	5,429	1
2. Citeseer [8]	Computer-Science Paper Classification	3,703	3,327	4,732	1
3. Pubmed [8]	Diabete-related Publication Classification	500	19,717	44,338	1
4. ogbn-arxiv [4, 13]	Subject Area Prediction of arXiv Papers	128	169,343	1,166,243	1
5. Amazon Computers [6]	Computer-Product Category Prediction	767	13,752	574,418	1
6. Amazon Photo [6]	Photo-Product Category Prediction	745	7,650	287,326	1
7. QM7b [7]	Molecule Property Regression	1	111,180	1,766,366	7,211
8. ogbg-molesol [4, 16]	Molecule Property Regression	9	14,991	30,856	1,128
9. PROTEINS [1]	Protein Property Prediction	3	43,471	205,559	1,113
10. ogbg-molbace [4, 16]	Molecule Property Classification	9	51,577	111,539	1,513
11. ogbg-molbbbp [4, 16]	Molecule Property Classification	9	49,068	105,842	2,039
12. WARD [12, 18]	Distributed Human Action Recognition	125	3,521,550	15,846,975	35,2155
13. ModelNet40 [17]	3D Object Recognition	3	12,603,392	252,067,840	12,311
14. ShapeNet [19]	3D Object Recognition	3	17,286,144	345,722,880	16,881

9,843 CAD models are used for training and 2,468 CAD models are for testing. For each CAD model, we sample 1,024 3D points from the mesh surfaces, as also done in [15]. Also, the ShapeNet part dataset contains 16,881 3D shapes from 16 categories. For each 3D shape, we also sample 1,024 points. Detailed class-by-class statistics of ShapeNet part dataset are provided in Tab. S6.

Table S6. Detailed dataset statistics of the ShapeNet part dataset [19] for the task of point cloud classification.

	Total	Aero	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motor	Mug	Pistol	Rocket	Skateboard	Table
# shapes	16,881	2,690	76	55	898	3,758	69	787	392	1,547	451	202	184	283	66	152	5,271
# shape labels	16	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

3. Additional Results on Heterogeneous Node Property Prediction

In this section, we provide additional results of using heterogeneous data reprogramming to tackle heterogeneous node property prediction.

Implementation Details. In total, we use five datasets, Citeseer, Cora, Pubmed, Amazon Computers, and Amazon Photo as examples to validate the effectiveness of the proposed approaches. In particular, we'd like to clarify that the different pre-trained models on various datasets have a maximum number of output dimensions, meaning that the number of classes in the downstream datasets cannot exceed that of the pre-trained one. So, in Tab. 2 of the main manuscript where we use Citeseer with six categories as the pre-trained task, we have to use the corresponding six categories for Cora, Amazon Computers, and Amazon Photo.

For the sake of the consistency between the main manuscript and the supplementary material, when we use the datasets of Cora, Amazon Computers, and Amazon Photo everywhere in this work, we consistently only use the first six target classes of the corresponding datasets, as shown in Tab. S7. For the task of Pubmed with only three classes, we predict the full target three classes, by simply using the corresponding three output neurons in the pre-trained Citeseer model for the final prediction. Nevertheless, it is indeed possible to leverage the technique of adaptive prototype learning to alleviate the dilemma of such output limits. However, due to the page limit, we have to elaborate on this part in our future work.

For the dataset splittings of Citeseer, Cora, Pubmed, we use the splitting protocol in [2]. Specifically, for Cora, we use 1208 samples for training, 500 samples for validation, and 1000 samples for testing; for Citeseer, we use 1827 samples for training, 500 samples for validation, and 1000 samples for testing; for Pubmed, 18217 samples are used for training, 500 samples are used for validation, and 1000 samples are used for testing. For Amazon Computers and Amazon

Photo, since there is no standard splitting protocol, in our experiment, we randomly split these two datasets with the ratio of TrainingSet : ValidationSet : TestingSet $= 6 : 2 : 2$.

At the pre-training stage, the learning rate is set to 0.005. At the reusing stage, the ascent step size for optimizing the padded features is set to 0.0001 with a weight decay of 5×10^{-4} by default. We use the Adam optimizer for both stages. The results are obtained by computing the average of 20 independent runs. The network architectures are given in Tab. S7, which follow the official architecture design in deep graph library [14] without specific modifications.

Table S7. Detailed network architectures for producing the results in Tab. 2 of the main manuscript and also those in Tabs. S8 and S9 of this supplementary material.

Pre-trained Architectures	**Layers**	**Attention Heads**	**Input**	**Hidden**	**Output**	**Parameter Sizes**
Citeseer	2	{8, 1}	3703	8	6	474,892
Cora	2	{8, 1}	1433	8	6	184,332
Amazon Computers	2	{8, 1}	767	8	6	99,084
Amazon Photo	2	{8, 1}	745	8	6	96,268

Ablation Studies. In Tab. S8 and Tab. S9, we demonstrate the results of the ablation studies with various pre-trained GNNs, corresponding to Tab. 2 of the main manuscript. In particular, we use Cora and Amazon Computers as the pre-trained tasks for Tab. S8 and Tab. S9, respectively. It is noticeable that the proposed method still achieves promising results with different pre-trained models. For example, the three results of the downstream Amazon Photo task with the pre-trained models Citeseer (Tab. 2 of the main paper), Cora (Tab. S8), and Amazon Computers (Tab. S9) are all equally encouraging, showing that the proposed *MetaFP* and *ReAgg* methods make it possible for resource-efficient model reusing under the scenarios of having only a limited number of pre-trained models.

Table S8. Ablation studies of using different pre-trained GNNs, corresponding to Tab. 2 in the main manuscript. Here, we reuse a pre-trained node classification model on Cora to handle the tasks of Amazon Computers and Amazon Computers with heterogeneous feature dimensions.

Pre-trained Task **Heterogeneous** Task Type	Cora ***Machine-Learning Paper Classification***			
Pre-trained Results	*Accuray*: 0.9121			
Downstream Tasks **Heterogeneous** Task Types	Amazon Computers ***Computer-Product Category Prediction***		Amazon Photo ***Photo-Product Category Prediction***	
Reusing Methods	Vanilla	Ours	Vanilla	Ours
Downstream Results	*Accuray*: 0.1497	*Accuray* : **0.8792**	*Accuray*: 0.1354	*Accuray*: **0.8785**
Re-training from Scratch	*Accuray*: 0.9485		*Accuray*: 0.9561	

Table S9. Ablation studies of using a pre-trained computer-product category prediction model to tackle Amazon Computers and Pubmed with various input dimensions, corresponding to Tab. 2 in the main manuscript.

Pre-trained Task **Heterogeneous** Task Type	Amazon Computers ***Computer-Product Category Prediction***			
Pre-trained Results	*Accuray*: 0.9485			
Downstream Tasks **Heterogeneous** Task Types	Amazon Photo ***Photo-Product Category Prediction***		Pubmed ***Diabete-Publication Classification***	
Reusing Methods	Vanilla	Ours	Vanilla	Ours
Downstream Results	*Accuray*: 0.2012	*Accuray*: **0.9161**	*Accuray*: 0.4270	*Accuray* : **0.8320**
Re-training from Scratch	*Accuray*: 0.9561		*Accuray*: 0.8840	

4. Additional Results on Heterogeneous Graph Classification and Regression

In this section, we demonstrate more results of exploiting the heterogeneous data reprogramming method of *MetaFP* to handle the heterogeneous cross-level graph classification and regression tasks.

Table S10. Network architectures for heterogeneous downstream graph classification and regression tasks, corresponding to Tab. 4 in the main manuscript and also Tabs. S11 and S12 in this supplementary material.

Pre-trained Architectures	**Layers**	**Attention Heads**	**Input**	**Hidden**	**Output**	**Parameter Sizes**
`Cora`	2	{8, 1}	1433	8	6	184,332
`Amazon Computers`	2	{8, 1}	767	8	6	99,084
`Amazon Photo`	2	{8, 1}	745	8	6	96,268

Implementation Details. To demonstrate the effectiveness of the proposed methods for heterogeneous cross-level graph analysis, here we use node classification models as the pre-trained ones, and reuse them to handle the task of heterogeneous graph classification and regression. The network architectures of the pre-trained node property prediction models in Tab. 4 of the main manuscript and Tabs. S11 and S12 are provided in Tab. S10. Also, to address the issue of different output dimensions of node-level and graph-level tasks, we adopt the slimmable strategy in dynamic networks, *i.e.*, simply using the part of the output neurons to generate the prediction results, and ignoring the other extra unaligned output dimensions.

The detailed dataset statistics of the various datasets used in this section can be found in Sect. 2. In the pre-training phase, the experimental settings are the same as those in Sect. 3, *i.e.*, with a learning rate of 0.005 and the Adam optimizer. During model reusing, we set the ascent step size as 0.0001 with a weight decay of 5×10^{-4}.

Table S11. Ablation studies of reusing various pre-trained GNNs, corresponding to Tab. 4 in the main paper. Here, we pre-train a model on `Amazon Computers` and then reuse it to tackle the graph regression task of `QM7b` as well as the graph classification task of `PROTEINS`.

Pre-trained Task / **Heterogeneous** Task Type	`Amazon Computers` / ***Computer-Product Category Prediction***			
Pre-trained Results	*Accuray*: 0.9485			
Downstream Tasks / **Heterogeneous** Task Types	`QM7b` / ***Molecule Property Regression***		`PROTEINS` / ***Protein Property Prediction***	
Reusing Methods	Vanilla	Ours	Vanilla	Ours
Downstream Results	*MAE*: 13.1409	*MAE* : **2.4634**	*Accuray*: 0.5268	*Accuray*: **0.6250**
Re-training from Scratch	*MAE*: 0.7264		*Accuray*: 0.6964	

Table S12. Ablation studies of reusing a pre-trained node classification model on `Amazon Photo` to handle the unseen graph-level regression and classification tasks of `QM7b` and `PROTEINS`, corresponding to Tab. 4 in the main paper.

Pre-trained Task / **Heterogeneous** Task Type	`Amazon Photo` / ***Photo-Product Category Prediction***			
Pre-trained Results	*Accuray*: 0.9561			
Downstream Tasks / **Heterogeneous** Task Types	`QM7b` / ***Molecule Property Regression***		`PROTEINS` / ***Protein Property Prediction***	
Reusing Methods	Vanilla	Ours	Vanilla	Ours
Downstream Results	*MAE*: 24.1829	*MAE* : **2.3093**	*Accuray*: 0.3304	*Accuray*: 0.6071
Re-training from Scratch	*MAE*: 0.7264		*Accuray*: 0.6964	

Ablation Studies. We show in Tab. S11 and Tab. S12 the ablation study results of various pre-trained models for heterogeneous downstream graph classification and regression tasks, corresponding to Tab. 4 of the main manuscript. As can be observed from Tabs. S11, S12 and Tab. 4 in the main paper, the proposed *MetaFP* approach delivers gratifying results with all these three different pre-trained tasks of `Amazon Computers`, `Amazon Photo`, and `Cora`. Such observation validates the resource-efficient property of the proposed method in Sect. 3 again: getting rid of the restriction on well-provided pertinent pre-trained models.

5. Additional Results on Homogenous Node Property Prediction

In this section, we illustrate additional results of leveraging the homogenous data reprogramming method of *EdgSlim* to deal with the task of homogenous node property prediction.

Table S13. Network architectures used in Tab. 5 of the main manuscript and Tab. S14 of this supplementary material.

Pre-trained Architectures	Layers	Attention Heads	Input	Hidden	Output	Parameter Sizes
`Architecture-ogbn-arxiv-V1`	4	{8, 8, 8, 1}	128	8	20	35.75K
`Architecture-ogbn-arxiv-V2`	3	{8, 8, 1}	128	8	20	27.43K

Implementation Details. The architecture details for Tab. 5 of the main manuscript and Tab. S14 in this supplementary material are provided in Tab. S13. In particular, `Architecture-ogbn-arxiv-V1` and `Architecture-ogbn-arxiv-V2` represent the two distinct pre-trained architectures used in the ablation studies in Tab. S14. The dataset details can be found in Tab. S5. Here, we divide the `ogbn-arxiv` dataset into two subsets, termed as `ogbn-arxiv-s1` and `ogbn-arxiv-s2`, where each subset contains 20 separate categories in the full `ogbn-arxiv` dataset. The pre-trained task is to predict the 20 classes in `ogbn-arxiv-s1`, whereas the downstream task is to classify the distinct 20 categories in `ogbn-arxiv-s2`. During pre-training, we use the Adam optimizer, with a learning rate of 0.005 and a weight decay of 5×10^{-4}, which are the same as other datasets without specific modifications or hyperparameter tuning.

Table S14. Ablation studies of reusing the pre-trained node classification models on `ogbn-arxiv` with various network architectures elaborated in Tab. S13.

Architectures	Types	Model Parameter Sizes	*Pre-trained Performance*	*Reusing Performance*	
				Vanilla	Ours
`Architecture-ogbn-arxiv-V1`	Pre-trained Acc	35.75K	0.7884	N/A	N/A
`Architecture-ogbn-arxiv-V1`	Downstream Acc	35.75K	N/A	0.2334	**0.6034**
`Architecture-ogbn-arxiv-V2`	Pre-trained Acc	27.43K	0.7849	N/A	N/A
`Architecture-ogbn-arxiv-V2`	Downstream Acc	27.43K	N/A	0.2191	**0.5507**

Ablation Studies. Tab. S14 demonstrates the results of the ablation studies of different pre-trained architectures. As can be observed from the last column of Tab. S14, the proposed *EdgSlim* leads to promising downstream performance without re-training or fine-tuning and outperforms the results of vanilla reusing by at least 30%. Also, the results are obtained at a low computational cost, with only three epochs. The physical edge elimination time is even less than one second for both architectures on a single NVIDIA GeForce RTX 2080 Ti GPU.

6. Additional Results on Homogenous Graph Classification and Regression

In this section, we give more results of applying the proposed homogenous data reprogramming approach of *MetaGP* to tackle the downstream tasks of homogenous node property prediction.

Implementation Details. To demonstrate the effectiveness of the proposed *MetaGP* method under the scenarios of homogenous graph classification and regression with homogenous input dimensions, we specifically use the three datasets of `ogbg-molbace`, `ogbg-molbbbp`, and `ogbg-molesol` that aim to classify or regress the graph properties, with more detailed statistics and descriptions in Sect. 2. The architecture details are provided in Tab. S15. In particular, different from the node classification task, the output layer of the graph-level tasks are linear layers. The learning rate setting is set to 0.005, with a weight decay of 5×10^{-4}, which is the same as other experiments. We use the Adam optimizer for pre-training.

Table S15. Network architectures for producing the results in Tab. 6 of the main manuscript and also those in Tab. S16 in this supplementary material.

Pre-trained Architectures	**Layers**	**Attention Heads**	**Output Layer**	**Input**	**Hidden**	**Output**	**Parameter Sizes**
`ogbg-molbace`	4	{1, 1, 1}	Linear	9	256	1	69.63K
`ogbg-molbbbp`	4	{1, 1, 1}	Linear	9	256	1	69.63K

Ablation Studies. We show in Tab. S16 the ablation studies of varying pre-trained models. In particular, instead of using `ogbg-molbace` as the pre-trained task as Tab. 6 in the main manuscript does, here we perform extensive ablation studies by pre-training a GNN on `ogbg-molbbbp` and considering `ogbg-molbace` as the downstream tasks. The results in Tab. S16 demonstrate that the proposed *MetaGP* is competent for various-domain downstream tasks even with different pre-trained models. As such, our method is readily applicable to scenarios where there is a limited number of pre-trained GNNs.

Table S16. Ablation studies of reusing different pre-trained GNNs, corresponding to Tab. 6 in the main manuscript. Here, the pre-trained model is designated for `ogbg-molbbbp`, whereas `ogbg-molbace` and `ogbg-molesol` are considered as the two target downstream tasks.

Pre-trained Task	`ogbg-molbbbp`			
Homogenous Task Type	***Graph Classification***			
Pre-trained Results	*ROC-AUC*: 0.6709			
Downstream Tasks	`ogbg-molbace`		`ogbg-molesol`	
Homogenous Task Types	***Graph Classification***		***Graph Regression***	
Reusing Methods	Vanilla	Ours	Vanilla	Ours
Downstream Results	*ROC-AUC*: 0.4330	*ROC-AUC*: **0.5903**	*RMSE*: 7.979	*RMSE*: **2.8183**
Re-training from Scratch	*ROC-AUC*: 0.7734		*RMSE*: 1.300	

7. Additional Results on 3D Object Recognition

In this section, we give more implementation details and more visualization results for the task of point cloud classification with `ModelNet40` and `ShapeNet`.

Implementation Details. In addition to node classification, graph classification, and graph regression tasks with citation networks and molecular graphs, we also conduct extensive experiments by reusing a GNN for 3D object recognition tasks. Here, we adopt two prevalent point cloud classification datasets, entitled `ModelNet40` and `ShapeNet`, of which the detailed statistics can be found in Tab. S5. We follow the official dataset splitting protocol in [15, 17], where 9,843 CAD models are used for training and 2,468 CAD models are for testing in pre-training. For each CAD model in both `ModelNet40` and `ShapeNet`, we sample 1,024 3D points from the mesh surfaces and also rescale the associated coordinates to fit into the unit sphere, as also done in [15]. The learning rate is set as 0.001 and the batch size is set to 16. We adopt the Adam optimizer [5]. The detailed architecture designs are summarized in Tab. S17. During the reusing stage, since `ModelNet40` contains 40 categories whereas `ShapeNet` has 16 classes, we simply use the first 16 output channels of the pre-trained `ModelNet40` as the output predictions for `ShapeNet`.

Table S17. Detailed network architectures for the task of 3D object recognition on `ModelNet40` and `ShapeNet`.

Pre-trained Models	**Layers**	**GNN Type**	**Feature Map Channels**	**MLPs**
Architecture-`ModelNet40`	8	EdgeConv	[64, 64, 128, 256, 1024]	[512, 256, 40]

More Visualization Results. In Fig. S3, we show more qualitative results of reusing GNNs for point cloud classification, by visualizing the structures of the feature spaces, corresponding to Fig. 6 of the main manuscript. The column termed "Vanilla" in Fig. S3 contains the results of vanilla model reusing, corresponding to "Vanilla" in Tab. 7 of the main manuscript. Meanwhile, the columns termed "Our" and "Re-train" in Fig. S3 indicate the results with the proposed *MetaGP* and those of re-training from scratch, respectively. It can be observed that the proposed method yields results that have a very similar feature structure to those of the cumbersome re-training ones, demonstrating the superiority of our approach.

8. Additional Results on Distributed Action Recognition

In this section, we provide more implementation details, as well as extensive ablation studies to validate the effectiveness of the proposed method on the task of distributed human action recognition.

Graph Construction from Time-series Data. To validate the effectiveness of the proposed method on the task of distributed action recognition, the first step is to transform the time-series data corresponding to different actions into graphs.

We begin our explanation of how to construct the graphs by firstly introducing the used action recognition dataset. The distributed human action recognition using wearable motion sensor network dataset `WARD` [18] is captured with five sensors. Each tensor has a triaxial accelerometer and a biaxial gyroscope. The sensors are at the left and right forearms, waist, left and right ankles, respectively, as shown in Fig. S2. Every sensor outputs 5 data streams. As such, totally 5×5 data streams are captured. Each data stream is recorded at 30Hz from humans with thirteen daily action categories.

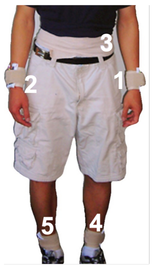

Figure S2. Illustrations of sensor positions (image credited to [11]).

For the construction of the graphs from `WARD`, as also done in [12], we use every 25 sequential data points from a single sensor as a node, and combine the nodes for every 50 sequential points into a graph in a fully-connected manner. As such, we can obtain a temporally growing graph, where every node is associated with a 5-channel 1-D signal $\mathbf{x}_t^i \in \mathbb{R}^{25\times 5}$, with $i = 1, 2 \ldots 5$ denoting the sensor index and $t = 1, 2 \ldots T$ representing the time index. All the nodes at the adjacent time index are also connected, resulting in a fully connected graph with ten nodes. As such, we can readily transform the problem of distributed human action recognition into that of graph classification that can be handled by the typical GNN models.

Implementation Details. The detailed architectures corresponding to Tab. 8 of the main manuscript and Tab. S19 in this supplementary material are provided in Tab. S18. In our experiment, we randomly select 8 action categories as pre-trained tasks and 5 human action classes as the downstream tasks. For `GCN`, we use the SGD optimizer, whereas for `GAT`, we adopt the Adam optimizer, as also done in [11]. The fixed model is pre-trained with a learning rate of 1×10^{-3} and a batch size of 100. Similar to other graph-level analyses, we use a linear layer as the output layer for action prediction.

Table S18. Detailed network architectures of reusing a GNN to handle the downstream distributed action recognition tasks.

Pre-trained Architectures	Layers	Attention Heads	Output Layer	Input	Hidden	Output	Parameter Sizes
`GCN`	3	-	Linear	125	384	8	16,807
`GAT`	3	{1, 1}	Linear	125	384	8	16,881

Ablation Studies. We perform extensive ablation studies in Tab. S19 by validating the effectiveness of the proposed *MetaGP* method with various GNN types, with `GCN` in Tab. 8 of the main paper and `GAT` in Tab. S19 of the supplementary material. Notably, the proposed method delivers gratifying performance with both the `GCN` and `GAT` architectures, without any re-training nor fine-tuning.

Table S19. Ablation studies of model reusing with heterogeneous-type GNNs on the task of distributed human action recognition, such as the GCN and GAT mechanisms, corresponding to Tab. 8 of the main manuscript.

Tasks	**Pre-trained Action Categories**								
	`Up`	`ReLi`	`WaLe`	`TuLe`	`Down`	`Jog`	`Push`	`ReSt`	**Overall Acc**
Pre-trained Acc (GAT [10])	0.9928	0.8969	0.9921	0.9904	0.9704	0.9962	0.9694	0.9757	0.9675

Tasks	**Downstream Action Categories**					
	`ReSi`	`WaFo`	`TuRi`	`WaRi`	`Jump`	**Overall Acc Acc**
Downstream Acc (GAT [10])	0.4709	0.8041	0.3697	0.6404	0.6008	0.6117

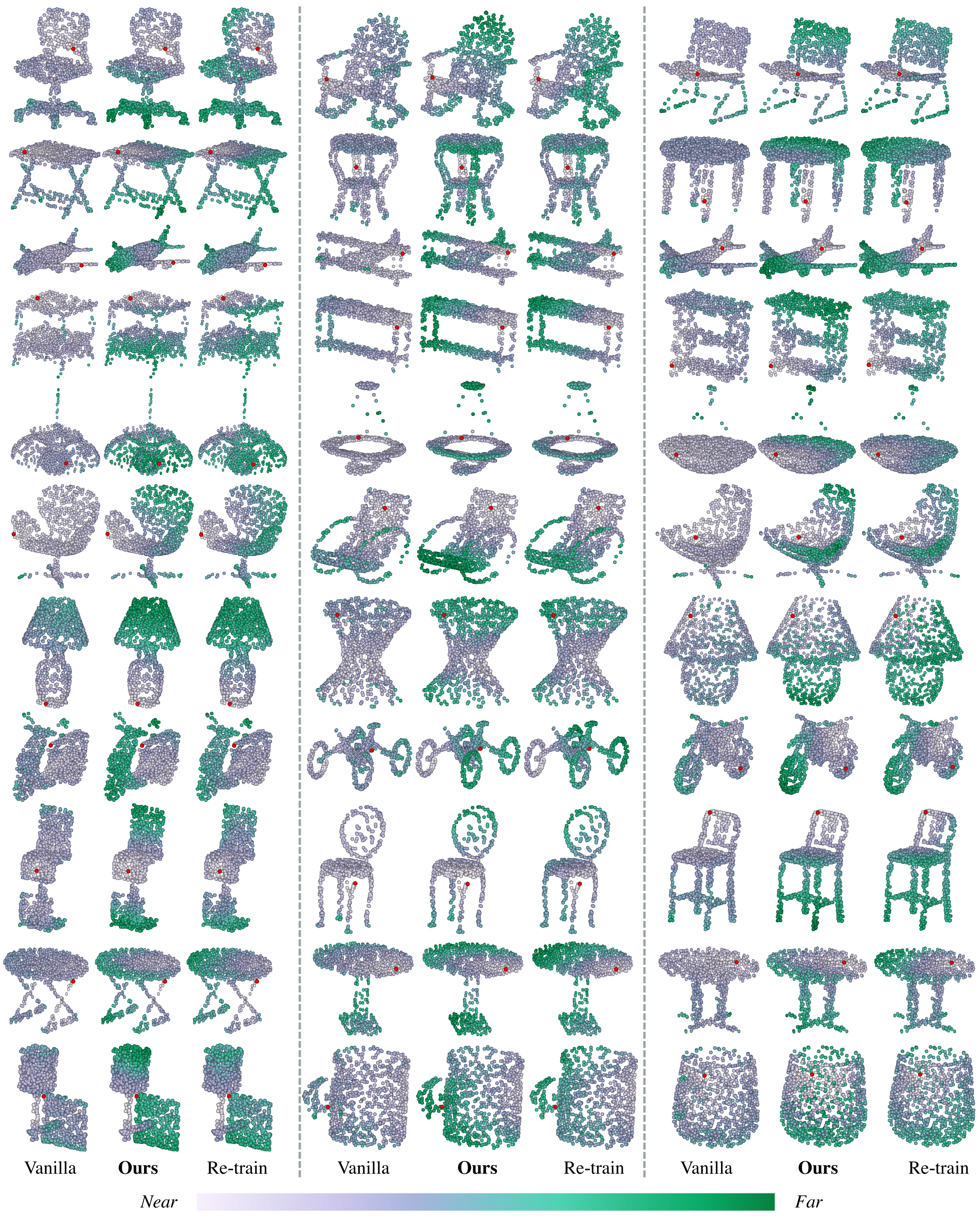

Figure S3. Visualization results of the structures of the feature space, depicted as the distance between the red point and the rest of the others. The visualized features are extracted from the intermediate layer of the models.

References

[1] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 2005. 5

[2] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018. 5

[3] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *NeurIPS*, 2020. 4

[4] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 5

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 9

[6] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015. 4, 5

[7] Grégoire Montavon, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *NJP*, 2013. 5

[8] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008. 3, 4, 5

[9] Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S Yu, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *TKDE*, 2022. 3

[10] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018. 10

[11] Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. Lifelong graph learning. In *CVPR*, 2022. 2, 10

[12] Chen Wang, Le Zhang, Lihua Xie, and Junsong Yuan. Kernel cross-correlator. In *AAAI*, 2018. 5, 10

[13] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *QSS*, 2020. 5

[14] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR Workshop*, 2019. 3, 6

[15] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *TOG*, 2019. 5, 9

[16] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 2018. 5

[17] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 4, 5, 9

[18] Allen Y Yang, Roozbeh Jafari, S Shankar Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks. *JAISE*, 2009. 5, 10

[19] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *TOG*, 2016. 2, 4, 5