Deep Graph Mating
– *Appendix* –

Yongcheng Jing[1]   **Seok-Hee Hong**[1]   **Dacheng Tao**[2]
[1]University of Sydney   [2]Nanyang Technological University
{yongcheng.jing,seokhee.hong}@sydney.edu.au, dacheng.tao@ntu.edu.sg

Contents

A Extended Related Work   **2**

    A.1 Graph Neural Networks (GNNs) 2

    A.2 Model Reuse 2

    A.3 Efficient Learning on Graphs 2

    A.4 Over-smoothing Alleviation 2

    A.5 Model Merging (Model Fusion) 3

B Model-specific Formulas of Equation 2   **3**

C Dataset Statistics and Descriptions   **3**

D Sensitivity Analysis   **4**

E Additional Results and Implementation Details   **5**

F Multi-model GRAMA   **8**

G Proofs   **8**

    G.1 Proof of Lemma 4.1 8

    G.2 Proof of Lemma 5.1 10

    G.3 Proof of Proposition 5.1 10

H Extended Limitation Discussion   **11**

I Broader Impacts   **12**

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Appendix

A   Extended Related Work

A.1   Graph Neural Networks (GNNs)

GNNs have demonstrated their considerable potential as a robust tool in the non-Euclidean domain, as highlighted in seminal works such as [35, 70, 61, 82, 10, 73, 39, 44, 63, 20, 38, 47]. GNNs facilitate the representation of non-Euclidean data through an end-to-end model, optimized via backpropagation. These models, which accept a broader format of input, have been employed in various fields beyond traditional areas like computer vision and natural language processing, covering numerous tasks reliant on grid data. For instance, recent studies, such as those by [14, 24], have focused on designing GNNs to handle molecular structures, which are naturally representable as graphs. Additionally, efforts like those by [55] aim to harness information from relational graphs. In this paper, we investigate the implementation of resource-efficient model reuse strategies for GNNs, aimed at minimising training efforts.

A.2   Model Reuse

Model reuse has been extensively explored in the Euclidean domain, particularly within convolutional neural networks (CNNs), as evidenced by significant works such as [17, 53, 50, 79, 12, 77, 84, 48]. This approach leverages pre-trained models to enhance performance or reduce training efforts for downstream tasks. However, the application of model reuse in the realm of GNNs is relatively nascent, despite the growing prevalence of topological data [33, 34, 54] and the availability of pre-trained GNNs online, which facilitate reproducibility and experimentation.

Notably, pioneering efforts by Yang et al. [76] introduced a knowledge distillation method specifically tailored for GNNs, aiming to create a lightweight model from a more complex teacher GNN. This work sparked further innovations, including those by Deng et al. [9], who introduced a more challenging scenario of graph-free knowledge distillation, and Joshi et al. [32], who enhanced the method with contemporary contrastive learning techniques. Furthermore, the study in [29] expands these concepts to multi-teacher scenarios, aiming to integrate knowledge from various heterogeneous pre-trained GNNs into a compact student model without the need for labeled data. However, this approach necessitates a cumbersome re-training process. In this paper, we introduce the first entirely learning-free model reuse paradigm, specifically designed for GNNs.

A.3   Efficient Learning on Graphs

GNNs have become the predominant tool in the field of topological graph data analysis [35, 69, 82, 27, 10]. Despite their impressive performance, the rapid evolution of GNNs introduces two primary challenges: the increasing scale of graph data [23, 7, 18], and the substantial growth in GNN model size [36, 6, 28], which leads to computationally intensive training [43, 80]. In response, researchers have developed several efficient graph representation learning techniques [26], including GNN quantisation to produce lightweight 1-bit representations of parameters and features [64, 4, 21, 30], neural architecture search to identify efficient architectures [59, 58, 60] tailored for GNNs [13, 49], and model reuse strategies that leverage knowledge from pre-trained models to reduce model size and training efforts [76, 9, 83, 11, 62, 32, 29, 16, 42, 31]. This paper specifically explores model reuse tailored for GNNs, aimed at facilitating efficient graph computing in a completely learning-free manner.

A.4   Over-smoothing Alleviation

Over-smoothing remains a pervasive issue in GNNs, where node features progressively blur into non-distinctiveness as the network depth increases, resulting in the dilution of essential information [52]. One approach to mitigate this effect involves architectural modifications to GNNs. Implementations such as adding skip connections or residual layers have shown promise [37]. Notable models like GCNII [6] and Jumping Knowledge Networks [71] incorporate these techniques to allow layers to leverage both their immediate inputs and outputs from prior layers or initial features, aiding in the preservation of the original node data.

2

Another strategy involves the use of normalisation methods during the training process to protect the diversity of features. Techniques like DropEdge [51], which intermittently omits edges during training to avert homogeneous feature blending, and PairNorm [81], designed to maintain consistent pairwise distances across layers, play crucial roles in preserving a balanced distribution of node features across the network. However, these techniques rely on ongoing learning processes, which contradicts our aim of achieving efficient GNN reuse without the need for further training in GRAMA.

In this work, we identify and explore the over-smoothing effects in reused GNN models and propose a learning-free normalisation layer that effectively counters the over-smoothing challenge in resource-efficient GNN reuse.

A.5 Model Merging (Model Fusion)

Model merging, also known as model fusion, seeks to combine multiple deep neural networks into a single model, relying solely on their parameters [67, 56, 1, 22, 57, 72, 40, 75, 41, 78, 46, 2, 25]. A significant challenge in model merging involves establishing neuron correspondences between different models. OTFusion [56], developed by Singh and Jaggi, utilises the Wasserstein distance to align weight matrices, facilitating parameter fusion without the need for re-training. In contrast, Git Re-basin [1] suggests minimising the L_2 distance between weight vectors to determine neuron associations. Building upon these works, recent advancements extend OTFusion's application to Transformer architectures [22] for enhanced efficiency and performance. A comprehensive survey on recent model merging techniques is provided in [40, 74]. Despite these developments, the integration of model merging techniques within GNNs remains unexplored. This paper presents GRAMA, the first method designed for weight space model merging in the non-Euclidean domain, addressing the specific challenges inherent to graph tasks.

B   Model-specific Formulas of Equation 2

To maintain clarity and simplicity, Eq. 2 in the main paper is based on the most basic form of GNNs. In this section, we develop more detailed, model-specific formulas for Eq. 2. Our model formulation presented here follows the unified mathematical framework detailed in [10], albeit with symbols adapted to those used in the main paper.

Eq. 2 tailored for GCN of Kipf and Welling [35] is as follows:

$$\Delta F_i \approx \sigma' \left(W \sum_{j \in \mathcal{N}(i)} \frac{X_j}{\sqrt{\deg_i}\sqrt{\deg_j}} \right) \cdot \left(\epsilon \sum_{j \in \mathcal{N}(i)} \frac{X_j}{\sqrt{\deg_i}\sqrt{\deg_j}} \right). \tag{1}$$

Eq. 2 tailored for GraphSAGE [15] is as follows:

$$\Delta F_i \approx \sigma' \left(W \, \text{Concat} \left(X_i, \text{Mean}_{j \in \mathcal{N}(i)} X_j \right) \right) \cdot \left(\epsilon \, \text{Concat} \left(X_i, \text{Mean}_{j \in \mathcal{N}(i)} X_j \right) \right). \tag{2}$$

In our future work, we are committed to developing more model-specific formulas tailored for more GNNs.

C   Dataset Statistics and Descriptions

Table A1: Summary of dataset statistics used in the main paper and the appendix.

Names	Task Descriptions	Nodes	Edges	# Graphs
ogbn-arxiv [65, 19]	Multi-class Node Classification	169,343	1,166,243	1
ogbn-products [5, 19]	Multi-class Node Classification	2,449,029	61,859,140	1
ogbn-proteins [8, 19]	Multi-label Node Classification	132,534	39,561,252	1
ogbg-molbace [68, 19]	Graph Classification	51,577	111,539	1,513
ogbg-molbbbp [68, 19]	Graph Classification	49,068	105,842	2,039
ModelNet40 [66]	3D Object Recognition	12,603,392	252,067,840	12,311
S3DIS [3]	3D Semantic Parsing	77,709,312	1,554,186,240	18,972

3

Table A2: Statistics on disjoint spaces per building area from Armeni et al. [3].

Area	Area (m^2)	Volume (m^3)	Office	Conf. Room	Auditorium	Lobby	Lounge	Hallway	Copy Room	Pantry	Open Space	Storage	WC	Total Num
Area 1	965	2850	31	2	-	-	-	8	1	1	-	-	1	45
Area 2	1100	3065	14	1	2	-	-	12	-	-	-	9	2	39
Area 3	450	1215	10	1	-	-	2	6	-	-	-	2	2	24
Area 4	870	2780	22	3	-	2	-	14	-	-	-	4	2	49
Area 5	1700	5370	42	3	-	1	-	15	-	1	-	4	2	55
Area 6	935	2670	37	1	-	-	1	6	1	1	-	-	-	53

Table A3: Object class statistics from the dataset of Armeni et al. [3].

Area	Ceiling	Floor	Wall	Beam	Column	Door	Window	Table	Chair	Sofa	Bookcase	Board
Area 1	56	45	235	62	58	87	30	70	156	7	91	28
Area 2	82	51	284	62	58	94	9	47	546	7	49	18
Area 3	38	24	160	14	13	38	9	31	68	10	42	13
Area 4	74	51	281	4	39	108	41	80	160	15	99	11
Area 5	77	69	344	4	75	128	53	155	259	12	218	43
Area 6	64	50	248	69	55	94	32	78	180	10	91	30

The datasets used in this paper cover various tasks, including node classification, graph classification, and 3D object recognition, offering a thorough assessment of the proposed methods. The `ogbn-arxiv` dataset supports multi-class node classification with 169,343 nodes and 1,166,243 edges, while `ogbn-products` presents a larger scale with 2,449,029 nodes and 61,859,140 edges. Another multi-class node classification dataset, `ogbn-proteins`, consists of 132,534 nodes and 39,561,252 edges, introducing additional complexity through multi-label classification. For graph classification, the `ogbg-molbace` and `ogbg-molbbbp` datasets are used, comprising 51,577 nodes and 111,539 edges across 1,513 graphs, and 49,068 nodes and 105,842 edges across 2,039 graphs, respectively. These datasets offer diverse challenges for evaluating GNNs.

The `ModelNet40` and `S3DIS` datasets are used for 3D object recognition and semantic parsing tasks, respectively. The `ModelNet40` dataset contains 12,603,392 nodes and 252,067,840 edges across 12,311 graphs, supporting the evaluation of 3D object recognition models. In comparison, the `S3DIS` dataset, intended for 3D semantic parsing, comprises 77,709,312 nodes and 1,554,186,240 edges distributed over 18,972 graphs, significantly increasing the data complexity and volume. Additional statistics for `S3DIS` are provided in Tabs. A1, A2, and A3.

D   Sensitivity Analysis

We include in Tab. A4 a sensitivity analysis with varying α for interpolation. This analysis explores the impact of different interpolation factors on the performance of our model across two datasets, Dataset A and Dataset B.

The results demonstrate a clear trend: as α increases from 0.1 to 0.9, the performance varies significantly. For Dataset A, the performance gradually improves as α increases, reaching its peak at $\alpha = 0.9$ with a score of 0.7222. Similarly, for Dataset B, the performance decreases as α increases, peaking at $\alpha = 0.1$ with a score of 0.7254 and reaching its lowest at $\alpha = 0.9$ with a score of 0.5755.

Notably, $\alpha = 0.5$ generates a balanced performance across both datasets, with a score of 0.6531 for Dataset A and 0.5957 for Dataset B. This outcome aligns with our assumption that $\alpha = 0.5$ ensures unbiased knowledge incorporation from both parent models, providing a compromise between the extremes observed at lower and higher α values. This balanced performance at $\alpha = 0.5$ supports the idea that this interpolation factor is optimal for integrating the knowledge from two parent models without favouring one over the other.

Overall, the sensitivity analysis confirms that while varying α can lead to significant changes in performance, an interpolation factor of $\alpha = 0.5$ delivers a balanced and unbiased integration of the knowledge from the parent models.

4

Table A4: Sensitivity analysis of the interpolation factor α for Tab. 2 of the main paper.

α	0.1	0.3	0.5	0.7	0.9
Dataset A	0.6344	0.6345	0.6531	0.7077	0.7222
Dataset B	0.7254	0.6504	0.5957	0.5855	0.5755

E Additional Results and Implementation Details

In this section, we provide additional implementation details for our experiments. All experiments were conducted on a single RTX 4090 GPU, with each parent model pre-trained using different initialisations. The code and models are included in the supplemental material.

In Tab. A5, we present the architecture used for the experiments on `ogbn-arxiv`. During pre-training, we use the Adam optimiser, with a learning rate of 0.005.

Table A5: Detailed network architectures for the task of node property prediction on the `ogbn-arxiv` dataset.

Pre-trained Architectures	GNN Type	Layers	Input	Hidden	Output
`Architecture-ogbn-arxiv`	GCN	5	128	256	40

For the dataset `ogbn-products`, which is used for node-level property prediction, the architecture details with GraphSAGE are outlined in Tab. A6.

Table A6: GraphSAGE-based network architectures for the experiments on the `ogbn-products` dataset.

Pre-trained Architectures	GNN Type	Layers	Input	Hidden	Output
`Architecture-ogbn-products`	GraphSAGE	3	100	256	47

We also report the standard deviations for the results on these two node property prediction datasets in Tab. A7. Upon conducting 20 independent runs using various random seeds, our method demonstrated encouraging consistency, evidenced by the low standard deviations. Specifically, for the `ogbn-arxiv` dataset pair A/B, the standard deviations are 5.067×10^{-7} and 2.441×10^{-6}, respectively, while for the `ogbn-products` dataset pair C/D, they are 3.829×10^{-6} and 1.157×10^{-5}. These results indicate that our method reliably delivers stable performance across different datasets.

Table A7: Standard derivations of the results of our method for Tab. 2 in the main paper, obtained with 20 independent runs using different random seeds. Given a pair of the same pre-trained GNNs, our method achieves stable results.

`ogbn-arxiv`	`ogbn-products`
Dataset A / Dataset B	Dataset C / Dataset D
5.067×10^{-7} / 2.441×10^{-6}	3.829×10^{-6} / 1.157×10^{-5}

Fig. A1 presents additional visualisation results by employing t-SNE [45] to depict the features from the intermediate layers. The representations of our PMC and CMC demonstrate more distinct clustering.

Table A8: Architecture details for the task of multi-label node classification on `ogbn-proteins`.

Pre-trained Architectures	GNN Type	GIN Layers	Input	Hidden	Output
`Architecture-ogbn-proteins`	GIN	3 (2 MLPs per layer)	8	256	112

5
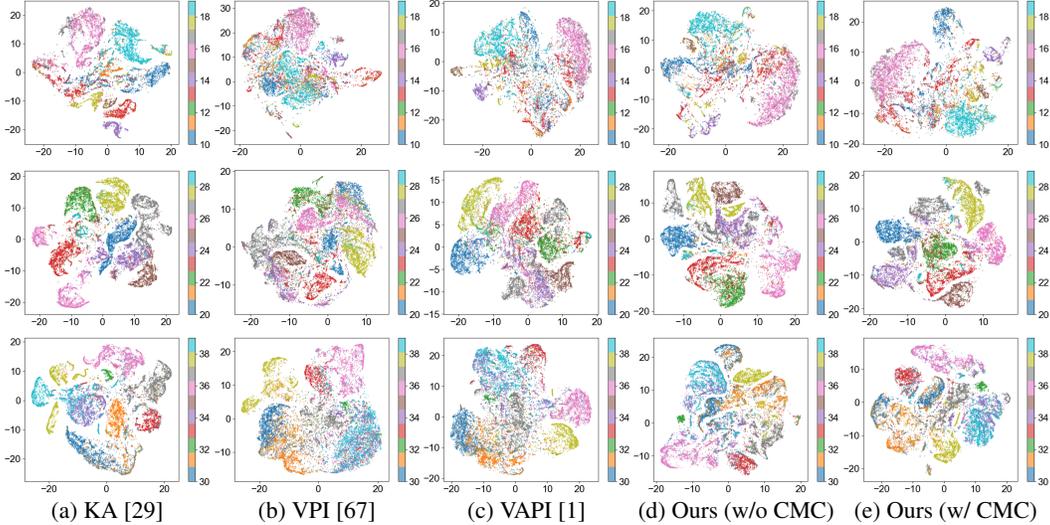
Figure A1: Additional t-SNE visualisations of various methods for the remaining 30 classes of the `ogbn-arxiv` dataset.

| (a) KA [29] | (b) VPI [67] | (c) VAPI [1] | (d) Ours (w/o CMC) | (e) Ours (w/ CMC) |

The details of our GIN-based architecture for experiments on `ogbn-proteins` are presented in Tab. A8, while those of the GAT-based architecture for graph-level tasks are provided in Tab. A9.

Table A9: Network architectures used in our experiments on `ogbg-molbace` and `ogbg-molbbbp`.

Pre-trained Architectures	GNN Type	Layers	Attention Heads	Output Layer	Input	Hidden	Output
Architecture-`ogbg-molbace`	GAT	4	{1, 1, 1}	Linear	9	256	1
Architecture-`ogbg-molbbbp`	GAT	4	{1, 1, 1}	Linear	9	256	1

Tabs. A10 and A11 detail the architectures used for 3D object recognition and semantic parsing tasks on the `ModelNet40` and `S3DIS` datasets, respectively.

Table A10: Detailed network architectures for the task of 3D object recognition on `ModelNet40`.

Pre-trained Models	GNN Type	Layers	Feature Map Channels
Architecture-`ModelNet40`	DGCNN	8	[64, 64, 128, 256, 1024]

Fig. A2 presents additional visualisation results for reusing GNNs in 3D object recognition, illustrating the structures of the feature spaces. Our proposed method produces results with a feature structure more closely resembling those of the cumbersome re-training-based KA method, compared to VPI and VAPI, highlighting the superiority of the proposed method.

Table A11: Architecture details for semantic parsing on the `S3DIS` dataset.

Pre-trained Models	Layers	GNN Type	Feature Map Channels
Architecture-`S3DIS`	9	DGCNN	[64, 64, 64, 64, 64, 1024, 512, 256]

6
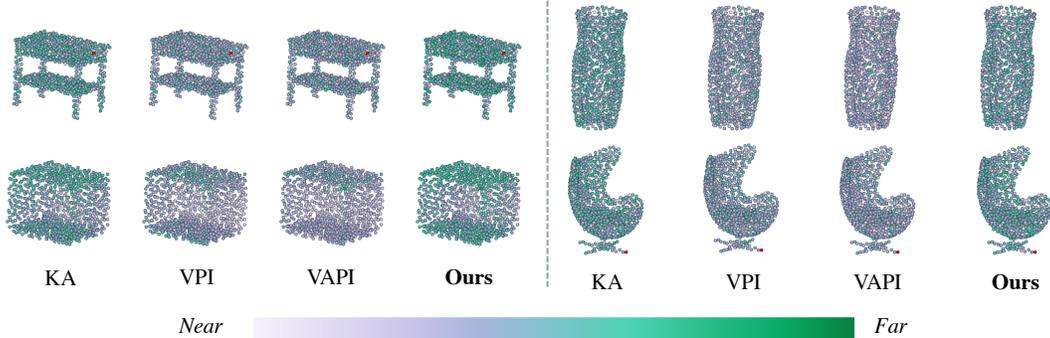
Figure A2: Additional visualisations of the feature space structure, shown by measuring the distance from the red point to other points. The features visualised are derived from the models' intermediate layers.

In particular, we performed additional experiments across various network architectures and dataset splits using the `ModelNet40` dataset. Detailed descriptions of the additional architecture are provided in Tab. A12.

Table A12: Additional network architectures for the task of 3D object recognition on `ModelNet40` in the appendix.

Pre-trained Models	GNN Type	Layers	Feature Map Channels
Architecture-`ModelNet40`-Appendix	DGCNN	7	[32, 32, 64, 128, 512]

The corresponding GRAMA results for the additional architecture in Tab. A12 are presented in Tab. A13, demonstrating that our proposed approach consistently yields promising results across various architectures.

Table A13: Additional results with different network architectures on the `ModelNet40` dataset.

Architecture	Partition	Datasets	Parent 1	Parent 2	Ours (w/o CMC)	Ours (w/ CMC)
Architecture-Appendix	20%/80%	Dataset 1	0.9184	0.8920	0.8236	0.8846
Architecture-Appendix	20%/80%	Dataset 2	0.8175	0.9299	0.8279	0.8550

Furthermore, we demonstrate the performance of our DuMCC across various dataset partition ratios in Tab. A14, supplementing the 20%/80% split discussed in the main paper. Notably, DuMCC exhibits promising performance across different pre-trained parent models trained on diverse partitioned datasets.

Table A14: Additional results with different dataset splits on the `ModelNet40` dataset.

Architecture	Partition	Datasets	Parent 1	Parent 2	Ours (w/o CMC)	Ours (w/ CMC)
Architecture-Main	10%/90%	Dataset 1	0.9246	0.8393	0.8451	0.8782
Architecture-Main	10%/90%	Dataset 2	0.7621	0.9294	0.8374	0.8422

We further conducted additional experiments by re-training a model using combined parent datasets with ground-truth labels. The results, shown in Tab. A15, establish an upper bound for GRAMA's performance.

7

Table A15: Results of retraining a multi-dataset model that can jointly combine the expertise of both parent models.

Tables	Datasets	Parent 1	Parent 2	Re-training
Tab. 2	`ogbn-arxiv`	0.7193 / 0.5516	0.6564 / 0.7464	0.6903 / 0.7268
Tab. 2	`ogbn-products`	0.7982 / 0.7308	0.7626 / 0.7904	0.7981 / 0.7787
Tab. 3	`ogbn-proteins`	0.7478	0.7222	0.7514
Tab. 3	`ogbg-molbace, ogbg-molbbbp`	0.7247 / 0.4681	0.4067 / 0.6366	0.6954 / 0.6087
Tab. 4	`ModelNet40`	0.9159 / 0.8151	0.8862 / 0.9275	0.9390 / 0.9243
Tab. 5	`S3DIS`	0.8181	0.8174	0.8428

F   Multi-model GRAMA

In this section, we explore the feasibility of multi-model GRAMA by conducting additional experiments on disjoint partitions of the `ogbn-arxiv` dataset, focusing on the simultaneous reuse of three pre-trained models. The results, presented below, demonstrate that our proposed DuMCC approach is also effective in multi-model GRAMA contexts.

Table A16: Results of multi-model GRAMA involving the simultaneous reuse of three pre-trained parent models.

Models	Performance: Dataset 1	Performance: Dataset 2	Performance: Dataset 3
Parent 1	0.6645	0.6476	0.6040
Parent 2	0.4796	0.7044	0.4651
Parent 3	0.5805	0.6078	0.7728
Child	0.5574	0.6360	0.5478

G   Proofs

G.1   Proof of Lemma 4.1

Lemma 4.1 (Amplified Sensitivity of GNNs to Parameter Misalignment) *GNNs exhibit greater sensitivity to mismatches in parameter alignment compared to CNNs, amplified by the degree of connectivity and heterogeneity of the node features in the graph topology:*

$$\Delta F_i \approx \sigma' \left(\sum_{j \in \mathcal{N}(i)} W \cdot X_j \right) \cdot \sum_{j \in \mathcal{N}(i)} \epsilon \cdot X_j, \tag{3}$$

where ΔF_i refers to the change in output at node i due to the perturbations ϵ in the weights W, and σ' represents the derivative of the activation function. X_j denotes the features of the nodes within the neighbourhood $\mathcal{N}(i)$ of node i.

Proof. For CNNs, consider the scenario where a convolution operation is applied to an input feature matrix X using a filter W. The convolution aims to extract spatially relevant features from X by applying W across its entirety.

We introduce a perturbation ϵ to W such that $W' = W + \epsilon$. This allows us to examine its impact on the output. The output F from the original convolution operation is derived as follows:

$$F_{i,j} = \sum_{k,l} W_{k,l} \cdot X_{i+k,j+l}. \tag{4}$$

8

After the perturbation, the new output F' becomes:

$$
\begin{aligned}
F'_{i,j} &= \sum_{k,l}(W_{k,l} + \epsilon_{k,l}) \cdot X_{i+k,j+l} \\
&= F_{i,j} + \sum_{k,l} \epsilon_{k,l} \cdot X_{i+k,j+l}.
\end{aligned}
\tag{5}
$$

Here, the first term $F_{i,j}$ is the original convolution output, and the second term represents the impact of the perturbation across the spatial dimensions of X.

Thus, the change in the convolution output $\delta F_{i,j}$ due to the perturbation can be formulated as:

$$
\delta F_{i,j} = \sum_{k,l} \epsilon_{k,l} \cdot X_{i+k,j+l},
\tag{6}
$$

which illustrates how each element of the perturbation $\epsilon_{k,l}$ affects the corresponding local regions of X.

By applying the activation function σ, which introduces non-linearity into the network, to both the original and perturbed outputs, and using the first-order Taylor expansion at $F_{i,j}$, we can obtain:

$$
\sigma(F_{i,j} + \delta F_{i,j}) \approx \sigma(F_{i,j}) + \sigma'(F_{i,j}) \cdot \delta F_{i,j},
\tag{7}
$$

where σ' is the derivative of σ at $F_{i,j}$.

As such, by substituting Eq. 6 into Eq. 7, the change in the activated output $\Delta F_{i,j}$ due to the weight perturbations can be approximated as:

$$
\Delta F_{i,j} \approx \sigma'(F_{i,j}) \cdot \sum_{k,l} \epsilon_{k,l} \cdot X_{i+k,j+l}.
\tag{8}
$$

Eq. 8 demonstrates the localised impact of weight perturbations within CNNs, illustrating how changes to the weights affect outputs primarily in the specific regions where those weights are applied. The effect of such perturbations is regulated by the derivative of the activation function at each output location. As a result, the perturbation's influence is confined to the region corresponding to the receptive field of the weights within the convolutional filter.

For GNNs, consider a node i with a neighborhood $\mathcal{N}(i)$, and let the feature vector at node j in the neighborhood be X_j. If the weight matrix W is also perturbed by ϵ, then: $W' = W + \epsilon$.

The output F_i for node i before perturbation can be formulated as:

$$
F_i = \sigma\left(W \cdot \sum_{j \in \mathcal{N}(i)} X_j \right).
\tag{9}
$$

After perturbation, the output F'_i thereby becomes:

$$
F'_i = \sigma\left((W + \epsilon) \cdot \sum_{j \in \mathcal{N}(i)} X_j \right).
\tag{10}
$$

By using a Taylor approximation similar to the CNN case in Eq. 7, F'_i in Eq. 10 can be further derived as:

$$
F'_i \approx F_i + \sigma'\left(W \cdot \sum_{j \in \mathcal{N}(i)} X_j \right) \cdot \left(\epsilon \cdot \sum_{j \in \mathcal{N}(i)} X_j \right).
\tag{11}
$$

Thus, the change ΔF_i can be formulated as:

$$
\Delta F_i \approx \sigma'\left(W \cdot \sum_{j \in \mathcal{N}(i)} X_j \right) \cdot \left(\epsilon \cdot \sum_{j \in \mathcal{N}(i)} X_j \right).
\tag{12}
$$

9

Eq. 12 shows that in GNNs, the effect of weight changes can be amplified by the aggregation of inputs from a node's neighborhood, contrasting with the more localised impact seen in CNNs (Eq. 8). The sensitivity in GNNs is further influenced by the complex graph structure, which can lead to diverse impacts depending on the node's position and connectivity within the graph. These characteristics render GNNs particularly susceptible to small parameter misalignment during model merging processes.

The proof is complete. □

G.2 Proof of Lemma 5.1

> **Lemma 5.1** (Variance Reduction in Interpolated Graph Embeddings) *The variance of the graph embeddings in an interpolated child GNN is typically smaller than the variances of the embeddings from the individual pre-trained parent GNNs.*

Proof. Assume that $\mathcal{G}_a$ and $\mathcal{G}_b$ are two parent GNN models that have been trained independently, as defined in Alg. 1. For a particular center node i, the output embeddings produced by these models at layer ℓ are denoted as $F_{i,a}^{\ell}$ and $F_{i,b}^{\ell}$, respectively, consistent with Sect. G.1.

We then perform a linear parameter interpolation between two parent models $\mathcal{G}_a$ and $\mathcal{G}_b$ to obtain a child GNN $\mathcal{G}$. For simplicity, we consider a scenario in which node features are linearly combined. Aligning with the equations in Sect. 4, we have: $F_i^{\ell} \approx \alpha F_{i,a}^{\ell} + (1 - \alpha)F_{i,b}^{\ell}$.

Consider each feature dimension of $F_{i,a}^{\ell}$ and $F_{i,b}^{\ell}$ to be a random variable, due to input variations, model initialisation, or stochastic training processes. We can then model $F_{i,a}^{\ell}$ and $F_{i,b}^{\ell}$ as having variances σ_a^2 and σ_b^2, respectively.

The variance of the interpolated features F_i^{ℓ} in each dimension can be computed as follows:

$$\text{Var}(F_i^{\ell}) = \text{Var}\left(\alpha F_{i,a}^{\ell} + (1 - \alpha)F_{i,b}^{\ell}\right). \tag{13}$$

By exploiting the properties of variance, Eq. 13 can be further derived into:

$$\text{Var}(F_i^{\ell}) = \alpha^2 \text{Var}(F_{i,a}^{\ell}) + (1 - \alpha)^2 \text{Var}(F_{i,b}^{\ell}) + 2\alpha(1 - \alpha)\text{Cov}(F_{i,a}^{\ell}, F_{i,b}^{\ell}). \tag{14}$$

Assume that the embeddings from $\mathcal{G}_a$ and $\mathcal{G}_b$ are independent, the covariance term $\text{Cov}(F_{i,a}^{\ell}, F_{i,b}) = 0$. Thus, Eq. 14 can be simplified into:

$$\text{Var}(F_i^{\ell}) = \alpha^2 \sigma_a^2 + (1 - \alpha)^2 \sigma_b^2. \tag{15}$$

Now, we consider a specific case where $\sigma_a^2 = \sigma_b^2 = \sigma^2$ for simplicity. As such, we have:

$$\text{Var}(F_i^{\ell}) = \alpha^2 \sigma^2 + (1 - \alpha)^2 \sigma^2 \quad = (1 - 2\alpha + 2\alpha^2)\sigma^2 = (1 - 2\alpha(1 - \alpha))\sigma^2. \tag{16}$$

Given that $\alpha(1 - \alpha) \geq 0$ for α within the range [0, 1], and that $2\alpha(1 - \alpha)$ reaches its maximum value when $\alpha = 0.5$, it follows that:

$$\text{Var}(F_i^{\ell}) \leq \sigma^2. \tag{17}$$

Specifically, in the framework of GRAMA, the interpolation coefficient α is usually set to 0.5. This choice is made to ensure an equitable integration of knowledge from both pre-trained models, facilitating a balanced contribution that avoids favouring the characteristics of either model. Consequently, at $\alpha = 0.5$, the variance $\text{Var}(F_i^{\ell})$ is minimised, indicating that the variance of node representations in the resulting child GNN is typically less than that in the parent GNNs.

The proof is complete.

□

G.3 Proof of Proposition 5.1

10

Proposition 5.1 (Increased Susceptibility to Over-Smoothing in Child GNNs) *Interpolated child GNNs exhibit increased susceptibility to over-smoothing compared to their parent networks, as measured by Dirichlet energy:*

$$\mathcal{E}(X^\ell) \leq \max\left(\mathcal{E}(X_a^\ell), \mathcal{E}(X_b^\ell)\right), \tag{18}$$

where $\mathcal{E}(X^\ell)$ denotes the Dirichlet energy for the node features X^ℓ at layer ℓ of the child GNN.

Proof. In the literature [52], *Dirichlet Energy* is employed to measure the smoothness of graph signals, where a lower value indicates higher similarity or smoothness among node features. To quantify over-smoothing in GNNs, Dirichlet energy is typically defined as:

$$\mathcal{E}(X^\ell) = \frac{1}{N} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} \|X_i^\ell - X_j^\ell\|^2, \tag{19}$$

where X represents the node features at layer ℓ, and V is the set of vertices. $\mathcal{N}_i$ denotes the neighbors of node i, and N is the total number of vertices.

In the context of the GRAMA framework, Eq. 19 can be extended as:

$$\begin{aligned}
\mathcal{E}(X) &= \frac{1}{N} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} \|\alpha(X_{i,a} - X_{j,a}) + (1-\alpha)(X_{i,b} - X_{j,b})\|^2 \\
&= \frac{1}{N} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} (\alpha^2 \|X_{i,a} - X_{j,a}\|^2 + (1-\alpha)^2 \|X_{i,b} - X_{j,b}\|^2 \\
&\qquad\qquad + 2\alpha(1-\alpha)\langle X_{i,a} - X_{j,a}, X_{i,b} - X_{j,b}\rangle).
\end{aligned} \tag{20}$$

Assume that $X_{i,a}$ and $X_{i,b}$ are independently distributed, due to the independent training regimes of $\mathcal{G}_a$ and $\mathcal{G}_b$. The cross-term $\langle X_{i,a} - X_{j,a}, X_{i,b} - X_{j,b}\rangle$ will typically not contribute significantly to increasing the total energy compared to the individual energies of $X_{i,a}$ and $X_{i,b}$. Therefore, Eq. 20 simplifies to:

$$\begin{aligned}
\mathcal{E}(X) &\approx \frac{1}{N} \sum_{i \in V} \sum_{j \in \mathcal{N}_i} \left(\alpha^2 \|X_{i,a} - X_{j,a}\|^2 + (1-\alpha)^2 \|X_{i,b} - X_{j,b}\|^2\right) \\
&= \alpha^2 \mathcal{E}(X_a) + (1-\alpha)^2 \mathcal{E}(X_b).
\end{aligned} \tag{21}$$

From Eq. 21, the following inequality can be derived:

$$\mathcal{E}(X) \leq \max\left(\mathcal{E}(X_a), \mathcal{E}(X_b)\right). \tag{22}$$

Eq. 22 demonstrates that the Dirichlet energy of the resulted GNN from PMC is bounded above by the maximum Dirichlet energy of the parent networks.

Notably, with an interpolation coefficient $\alpha = 0.5$ in Eq. 21, the Dirichlet energy is effectively reduced due to the equal weighting average of the energies from both parent models. This reduction in Dirichlet energy typically results in greater homogeneity of node features across the network, thereby increasing the susceptibility of the child GNN to over-smoothing. Such over-smoothing can affect the network's expressive power and discriminative capability, which are crucial for effectively performing downstream tasks.

The proof is complete.

$\square$

H   Extended Limitation Discussion

In this section, we discuss potential solutions to address the limitations outlined in the main paper. The current DuMCC framework does not accommodate scenarios where parent models have different architectures or address tasks at different levels, challenges that fall within the scope of heterogeneous GRAMA.

11

A potential solution could involve Partial GRAMA, which entails first identifying shared features between two parent models that have different architectures or handle varied tasks. Subsequently, this possible method would yield a multi-head child GNN that selectively integrates elements of the pre-trained parent GNNs. Although this method would increase the model size compared to full GRAMA, which combines the entire models, it would be expected to offer a more favourable balance and trade-off between model size and performance.

To explore the possibility of a completely data-independent GRAMA scheme, one potential solution is to generate fake graphs, as described in [9], followed by alignment and calibration using these generated graphs.

I Broader Impacts

Graph-based artificial intelligence (AI) is becoming increasingly essential to modern society, especially in key industries like transportation and healthcare, where data is often modeled as topological graphs. In such graphs, nodes correspond to clusters of entities, and edges represent direct relationships, such as pathways and connections in transportation networks or atoms and bonds in molecular structures. The use of graph-based AI spans various applications from enhancing autonomous vehicle navigation and predicting traffic patterns to accelerating drug development and improving medical diagnoses.

In the realm of analysing topological graph data, Graph Neural Networks (GNNs) have become the leading AI technology in recent years. Nonetheless, the increasing reliance on GNN models has led to concerns about their environmental impact due to higher energy demands. This paper proposes leveraging existing GNNs to reduce the need for training from scratch, aiming to improve the energy efficiency of graph-based AI systems. By doing so, it enhances the sustainability of graph-based AI operations, contributing to lower energy emissions and decreasing production costs. This reduction in costs is expected to drive economic growth, potentially leading to more employment opportunities and broader economic benefits over time.

References

[1] Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *ICLR*, 2023.

[2] Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes. *arXiv preprint arXiv:2403.13187*, 2024.

[3] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016.

[4] Mehdi Bahri, Gaétan Bahl, and Stefanos Zafeiriou. Binary graph neural networks. *arXiv preprint arXiv:2012.15823*, 2020.

[5] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016.

[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

[7] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 2015.

[8] Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

[9] Xiang Deng and Zhongfei Zhang. Graph-free knowledge distillation for graph neural networks. In *IJCAI*, 2021.

[10] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

12

[11] Kaituo Feng, Changsheng Li, Ye Yuan, and Guoren Wang. Freekd: Free-direction knowledge distillation for graph neural networks. In *KDD*, 2022.

[12] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *ICML*, 2018.

[13] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *IJCAI*, 2021.

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[16] Yunzhi Hao, Yu Wang, Shunyu Liu, Tongya Zheng, Xingen Wang, Xinyu Wang, Mingli Song, Wenqi Huang, and Chun Chen. Attribution guided layerwise knowledge amalgamation from graph neural networks. In *ICONIP*, 2023.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[18] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. In *NeurIPS Datasets and Benchmarks*, 2021.

[19] 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.

[20] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *NeurIPS*, 2018.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[22] Moritz Imfeld, Jacopo Graldi, Marco Giordano, Thomas Hofmann, Sotiris Anagnostidis, and Sidak Pal Singh. Transformer fusion with optimal transport. In *ICLR*, 2024.

[23] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *ICLR*, 2022.

[24] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.

[25] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *ICLR*, 2023.

[26] Yongcheng Jing. *Efficient representation learning with graph neural networks*. PhD thesis, 2023.

[27] Yongcheng Jing, Yining Mao, Yiding Yang, Yibing Zhan, Mingli Song, Xinchao Wang, and Dacheng Tao. Learning graph neural networks for image style transfer. In *ECCV*, 2022.

[28] Yongcheng Jing, Xinchao Wang, and Dacheng Tao. Segment anything in non-euclidean domains: Challenges and opportunities. *arXiv preprint arXiv:2304.11595*, 2023.

[29] Yongcheng Jing, Yiding Yang, Xinchao Wang, Mingli Song, and Dacheng Tao. Amalgamating knowledge from heterogeneous graph neural networks. In *CVPR*, 2021.

[30] Yongcheng Jing, Yiding Yang, Xinchao Wang, Mingli Song, and Dacheng Tao. Meta-aggregator: Learning to aggregate for 1-bit graph neural networks. In *ICCV*, 2021.

[31] Yongcheng Jing, Chongbin Yuan, Li Ju, Yiding Yang, Xinchao Wang, and Dacheng Tao. Deep graph reprogramming. In *CVPR*, 2023.

13

[32] Chaitanya K Joshi, Fayao Liu, Xu Xun, Jie Lin, and Chuan-Sheng Foo. On representation knowledge distillation for graph neural networks. *arXiv preprint arXiv:2111.04964*, 2021.

[33] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

[34] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. Alphafold 2. *CASP*, 2020.

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

[36] Guohao Li, Matthias Müller, Guocheng Qian, Itzel C Delgadillo, Abdulellah Abualshour, Ali Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *TPAMI*, 2021.

[37] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[38] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[39] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *AAAI*, 2018.

[40] Weishi Li, Yong Peng, Miao Zhang, Liang Ding, Han Hu, and Li Shen. Deep model fusion: A survey. *arXiv preprint arXiv:2309.15698*, 2023.

[41] Chang Liu, Chenfei Lou, Runzhong Wang, Alan Yuhan Xi, Li Shen, and Junchi Yan. Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In *ICML*, 2022.

[42] Haibo Liu, Di Zhang, Liang Wang, and Xin Song. Multi-teacher local semantic distillation from graph neural networks. In *ADMA*, 2023.

[43] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, Dongrui Fan, Shirui Pan, and Yuan Xie. Survey on graph neural network acceleration: An algorithmic perspective. In *IJCAI*, 2022.

[44] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In *ICML*, 2019.

[45] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.

[46] Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. In *NeurIPS*, 2022.

[47] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[48] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[49] Yijian Qin, Ziwei Zhang, Xin Wang, Zeyang Zhang, and Wenwu Zhu. Nas-bench-graph: Benchmarking graph neural architecture search. In *NeurIPS Datasets and Benchmarks*, 2022.

[50] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[51] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

[52] T Konstantin Rusch, Michael Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *SAM Research Report*, 2023.

14

[53] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[54] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020.

[55] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

[56] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In *NeurIPS*, 2020.

[57] George Stoica, Daniel Bolya, Jakob Brandt Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training. In *ICLR*, 2024.

[58] Xiu Su, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Bcnet: Searching for network width with bilaterally coupled network. In *CVPR*, 2021.

[59] Xiu Su, Shan You, Jiyang Xie, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Searching for network width with bilaterally coupled network. *TPAMI*, 2022.

[60] Xiu Su, Shan You, Jiyang Xie, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. Vitas: Vision transformer architecture search. In *ECCV*, 2022.

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

[62] Can Wang, Zhe Wang, Defang Chen, Sheng Zhou, Yan Feng, and Chun Chen. Online adversarial distillation for graph neural networks. *arXiv preprint arXiv:2112.13966*, 2021.

[63] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In *AAAI*, 2018.

[64] Junfu Wang, Yunhong Wang, Zhen Yang, Liang Yang, and Yuanfang Guo. Bi-gcn: Binary graph convolutional network. *arXiv preprint arXiv:2010.07565*, 2020.

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

[66] 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.

[67] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.

[68] 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.

[69] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *TNNLS*, 2020.

[70] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[71] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

[72] Zhengqi Xu, Ke Yuan, Huiqiong Wang, Yong Wang, Mingli Song, and Jie Song. Training-free pretrained model merging. In *CVPR*, 2024.

15

[73] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

[74] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*, 2024.

[75] Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *ICLR*, 2024.

[76] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *CVPR*, 2020.

[77] Jingwen Ye, Zunlei Feng, and Xinchao Wang. Flocking birds of a feather together: Dual-step gan distillation via realer-fake samples. In *VCIP*, 2022.

[78] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*, 2023.

[79] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

[80] Shichang Zhang, Atefeh Sohrabizadeh, Cheng Wan, Zijie Huang, Ziniu Hu, Yewen Wang, Jason Cong, and Yizhou Sun. A survey on graph neural network acceleration: Algorithms, systems, and customized hardware. *arXiv preprint arXiv:2306.14052*, 2023.

[81] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *ICLR*, 2020.

[82] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[83] Sheng Zhou, Yucheng Wang, Defang Chen, Jiawei Chen, Xin Wang, Can Wang, and Jiajun Bu. Distilling holistic knowledge with graph neural networks. In *ICCV*, 2021.

[84] Xiatian Zhu, Shaogang Gong, et al. Knowledge distillation by on-the-fly native ensemble. In *NeurIPS*, 2018.

16