Graph Ladling: Shockingly Simple Parallel GNN Training without Intermediate Communication

Ajay Jaiswal [1] **Shiwei Liu** [1] **Tianlong Chen** [1] **Ying Ding** [1] **Zhangyang Wang** [1]

Abstract

Graphs are omnipresent and GNNs are a powerful family of neural networks for learning over graphs. Despite their popularity, scaling GNNs either by deepening or widening suffers from prevalent issues of *unhealthy gradients, over-smoothening, information squashing*, which often lead to substandard performance. In this work, we are interested in exploring a principled way to scale GNNs capacity without deepening or widening, which can improve its performance across multiple small and large graphs. Motivated by the recent intriguing phenomenon of model soups, which suggest that fine-tuned weights of multiple large-language pre-trained models can be merged to a better minima, we argue to exploit the fundamentals of model soups to mitigate the aforementioned issues of memory bottleneck and trainability during GNNs scaling. More specifically, we propose not to deepen or widen current GNNs, but instead present **a data-centric perspective** of model soups tailored for GNNs, i.e., to build powerful GNNs by dividing giant graph data to build independently and parallelly trained multiple comparatively weaker GNNs without any intermediate communication, and *combining their strength* using a greedy interpolation soup procedure to achieve state-of-the-art performance. Moreover, we provide a wide variety of model soup preparation techniques by leveraging state-of-the-art graph sampling and graph partitioning approaches that can handle large graph data structures. Our extensive experiments across many real-world small and large graphs, illustrate the effectiveness of our approach and point towards a promising orthogonal direction for GNN scaling. Codes are available at: `https://github.com/VITA-Group/graph_ladling`.

[*]Equal contribution [1]University of Texas at Austin. Correspondence to: Ajay Jaiswal <ajayjaiswal@utexas.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Graphs are well-known data structures that model pairwise relationships using a set of vertices and edges and represent a myriad of real-world data from social networks, knowledge graphs, gene expression networks, etc. Graph neural networks (GNNs) (Kipf & Welling, 2017; Defferrard et al., 2016; Veličković et al., 2017; You et al., 2020; Gao et al., 2018; Chiang et al., 2019; Zheng et al., 2021b; Chen et al., 2018; Duan et al., 2022; Thekumparampil et al., 2018), which use message passing (MP) strategy at their core for aggregating knowledge from neighbors, have been widely accepted as powerful algorithmic tools for learning over graphs. Although message passing provides GNNs superior performance over traditional MLPs, the nature of evolving massive topological structures prevents MP-based GNNs from scaling to industrial-grade graph applications, and the majority of state-of-the-art GNNs are only tested on small graph datasets. Additionally, due to the prevalent issues of unhealthy gradients, over-smoothening, and information squashing (Li et al., 2018; NT & Maehara, 2019; Alon & Yahav, 2021; Jaiswal et al., 2022; Liu et al., 2021) while training GNNs, increasing model capacity either by deepening (stacking more layers) or widening (increasing neighborhood coverage) often lead to sub-standard performance.

Previously, conforming to the empirical scaling laws (Kaplan et al., 2020), where the final model quality has been found to have a power-law relationship with the amount of data, model size, and compute time; several works (Li et al., 2021; Jaiswal et al., 2022; Zhou et al., 2021b) have attempted to scale GNNs (up to 1000 layers) assuming that processing larger graphs would likely benefit from more parameters. Unlike conventional deep neural networks, exploiting scale to revamp information absorption is not straight-forward for GNNs, and numerous existing works rely on architectural changes, regularization & normalization, better initialization (Li et al., 2019; Chen et al., 2020; Li et al., 2018; Liu et al., 2020; Rong et al., 2020; Huang et al., 2020; Zhao & Akoglu, 2020; Zhou et al., 2021a; Jaiswal et al., 2022) for improving the trainability and try to overcome astonishingly high memory footprints by mini-batch training, i.e. sampling a smaller set of nodes or partitioning large graphs (Hamilton et al., 2017; Chen et al., 2018; Zou

et al., 2019; Chiang et al., 2019; Zeng et al., 2019). While these methods are a step in the right direction, they do not scale well as the models become deeper or wider, since memory consumption is still dependent on the number of layers. We are interested in exploring an orthogonal step: *Does there exist a principled way to scale GNNs capacity without deepening or widening, which can improve its performance across small and large graphs?*

Recently, for large pre-trained models with many applications in computer vision (Han et al., 2020; Li et al., 2023; Mao et al., 2022; Jaiswal et al., 2021a; Zheng et al., 2021a) and natural language processing (Talmor et al., 2018; Jaiswal et al., 2021b; Zheng et al., 2023; Liu et al., 2023; Chen et al., 2023; Jaiswal et al., 2023), several works (Wortsman et al., 2022b; Ilharco et al., 2022; Juneja et al., 2022) investigate the intriguing phenomenon of "model soup", and have shown that weights of multiple dense fine-tuned models (candidate ingredients of soup) can be merged together into better generalizable solutions lying in low error basins. Despite enormous attention in NLP, it is unexplored for GNNs, presumably due to traditional wisdom that unlike large pre-trained transformers in NLP, current state-of-the-art GNN's model capacity is under-parameterized apropos of gigantic graphs. Despite some recent works (WAN, 2022; Lin et al., 2022) illustrating the benefits of GNNs ensembling, they exhibit high computational cost during inference which worsens in the context of large graphs. Motivated by the mergability of multiple fine-tuned models illustrated by model soups, in this work, we raise the research question: *Is it possible to leverage the fundamentals of model soups to handle the aforementioned issues of memory bottleneck and trainability, during scaling of GNNs?*

To this end, we propose not to deepen or widen current GNNs, but instead explore a data-centric perspective of dividing ginormous graph data to build independently and parallelly trained multiple comparatively weaker models without any intermediate communication, and merge them together using *greedy interpolation soup* procedure to achieve state-of-the-art performance. Our work draws motivation from recent advancements in parallel training of *pretrained language models* (LMs). For example, Branch-Merge-Train (BTM) (Li et al., 2022) learns a set of independent expert LMs specializing in different domains followed by averaging to generalize to multiple domains, and Lo-Fi (Wortsman et al., 2022a) illustrates the futility of communication overhead in data-parallel multi-node finetuning of LMs. Although these techniques seem to work for *large pre-trained LMs*, it is still unclear and unexplored if they will work for comparatively much **smaller GNNs in the training-from-scratch regime**. Moreover, GNNs deal with *graph-structured relational data* unlike independent samples in LMs and have their own unique set of challenges in their trainability, which makes it interesting to understand if soup phenomenon will help or deteriorate GNNs performance.

To our surprise, we found that independently trained GNNs from scratch can be smoothly aggregated using our greedy soup interpolation procedure to create a better generalizable GNN that performs exceptionally well. It suggests that linear scaling of GNNs either by deepening or widening is not necessarily the right approach towards building high-quality generalizable GNNs, and model soups can be an alternative. Note that, unlike the conventional model soup, we are the **first** to explore weight averaging for graph models, and in a well-motivated **data-centric** perspective (where it matters the most) considering the exploding size of real-world graphs. More specifically, in our work, we firstly illustrate easy adaptability of model soups across multiple SOTA GNN architectures trained on multiple small and large graphs (unexplored till now) and secondly present a novel data-centric perspective of model soups for large graph-structured relational data within constraints of available computational resource. Moreover, we extend current state-of-the-art graph sampling and partitioning techniques to facilitate the training of candidate soup ingredients which can be seamlessly combined at end for better generalization.

Our primary contributions can be summarized as:

- We illustrate the *harmonious adaptability* of model soups for graph-structured data and experimentally validate its performance benefits across multiple GNN architectures and graph scales. Our experiments reveal *orthogonal knowledge* stored in the candidate models which can be surprisingly aggregated during soup preparation using our greedy interpolation procedure.
- We present a novel *data-centric perspective* of model soups tailored for GNNs and carefully study the benefits of independent and parallel training of candidate models and their mergability in scenarios without the luxury of having computation resources to process entire graph, by extending state-of-the-art (SOTA) graph sampling and partitioning algorithms.
- Our extensive experiments across multiple large-scale and small-scale graphs {`Cora`, `Citeseer`, `PubMed`, `Flickr`, `Reddit`, `OGBN-ArXiv`, `OGBN-products`} using multiple GNN architectures {`GCN`, `JKNet`, `DAGNN`, `APPNP`, `SGC`, `GraphSAGE`, `CluterGCN`, `GraphSAINT`} validates the effectiveness of our approach.

2. Methodology

2.1. Preliminaries

Graph Neural Networks (GNNs) exploit an iterative message passing (MP) scheme to capture the structural information within nodes' neighborhoods. Let $G = (\mathcal{V}, \mathcal{E})$ be

a simple and undirected graph with vertex set $\mathcal{V}$, edge set $\mathcal{E}$ and .Let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ be the associated adjacency matrix, such that $\boldsymbol{A}_{ij} = 1$ if $(i, j) \in \mathcal{E}$, and $\boldsymbol{A}_{ij} = 0$ otherwise. $N = |\mathcal{V}|$ is the number of nodes. Let $\boldsymbol{X} \in \mathbb{R}^{N \times P_0}$ be the node feature matrix, whose i-th row represents a P_0-dimension feature vector for the i-th node. To facilitate understanding, a unified formulation of MP with K layers is presented as follows:

$$\boldsymbol{X}^{(\boldsymbol{K})} = \boldsymbol{A}^{(\boldsymbol{K-1})}\sigma(\boldsymbol{A}^{(\boldsymbol{K-2})}\sigma(\cdots\sigma(\boldsymbol{A}^{(0)}\boldsymbol{X}^{(0)} \boldsymbol{W}^{(0)}))\cdots)\boldsymbol{W}^{(\boldsymbol{K-2})})\boldsymbol{W}^{(\boldsymbol{K-1})} \quad (1)$$

where σ is an activation function (e.g. ReLU), $\boldsymbol{A}^{(l)}$ is the adjacency matrix at l-th layer, and $\boldsymbol{W}^{(l)}$ is the feature transformation weight matrix at l-th layer which will be learnt for the downstream tasks.

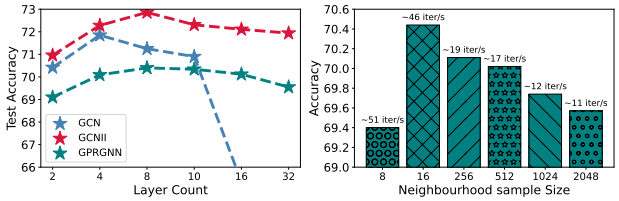

Figure 1. (Left) Performance comparison of GCN, GCNII, and GPRGNN on `OGBN-arxiv` dataset with increasing layer count. (Right) Performance comparison of 2-layer GraphSAGE on `OGBN-arxiv` dataset with increasing neighbor sampling threshold. Results illustrate that deepening and widening the model capacity doesn't necessarily help in improved performance.

2.2. GNNs and Issues with Model Explosion

Despite the enormous success of GNNs in learning high-quality node representations, many state-of-the-art GNNs are shallow due to notorious challenges in their trainability, resulting in sub-standard performance. Recently, we have been observing a massive explosion in the topological structure of real-world graph data, which raises demand to increase the model capacity to facilitate capturing long-range dependencies, and relaxing the widely-adopted restrictions to learn from limited neighbors subjected to resource constraints due to the design issues of message passing strategy.

Several works (Li et al., 2021; Jaiswal et al., 2022; Zhou et al., 2021b) have attempted the empirical scaling law from NLP, which suggests that over-parametrized models have better generalization capabilities, and proposed to similarly scale GNNs by stacking layers citing their increased ability to capture long-term dependencies. However, scaling GNNs capacity either by deepening to capture long-range dependencies or by widening to increase neighborhood aggregation is inhibited by the prevalent issues of unhealthy gradients during back-propagation leading to poor trainability, over-smoothening causing feature collapse, and information squashing due to overload and distortion of messages being propagated from distant nodes.

Figure 1 illustrates the effect of state-of-the-art GNNs scaling from the perspective of deepening and widening on `OGBN-arxiv`. It can be clearly observed that vanilla-GCN, GCNII and GPRGNN, all suffer significant performance from drop with increasing model capacity. Note that this substandard performance still exists in GCNII and GPRGNN, which are equipped with SOTA architectural modifications (eg. skip-connections), regularization, and normalization. Additionally, as shown in Figure 1(right), increasing the breadth of message passing to capture more and more neighborhood information for GraphSAGE on `OGBN-arxiv` does not necessarily help in improving the performance rather increases the memory footprints and reduce throughput (iterations/sec).

To this end, we argue that model explosion is not necessarily the only solution towards building high-quality generalizable GNNs and propose an orthogonal direction inspired by the recent distributed and parallel training platform of model soups. We propose to build independently and parallelly trained multiple comparatively weaker models without any intermediate communication, and merge their knowledge to create a better generalizable GNN. In the next sections, we will explain how model soups can be easily adapted for many state-of-the-art GNNs, and how can we smoothly extend the current graph sampling and partitioning to create candidate ingredients for the model soup.

Algorithm 1 Greedy Interpolation Soup Procedure

Input: Soup Ingredients $\boldsymbol{M} = \{M_1, M_2, ..., M_K\}$
$\boldsymbol{M}_{sorted} \leftarrow \texttt{SORT}_{ValAcc}(\boldsymbol{M})$
soup $\leftarrow \boldsymbol{M}_{sorted}[0]$
for $i = 1$ **to** K **do**
  for α **in** `linspace`$(0, 1, step)$ **do**
    if `ValAcc`(`interpolate`(soup, M_i, α)) $\geq$ `ValAcc`(soup) **then**
      soup $\leftarrow$ `interpolate`(soup, M_i, α)
    end if
  end for
end for

2.3. Model Soups and current state-of-the-art GNNs

In this section, we discuss about the harmonious adaptation of model soups for various state-of-the-art GNNs. Although model soups have been recently receiving ample attention for parallel training of large language pre-trained models, it is still unclear and unexplored if they will work for comparatively much smaller GNNs trained from scratch. We for the first time observed that unlike using a pre-trained initialization in LLMs, GNNs optimized independently from the **same random initialization** organically lie in the same

basin of the error landscape, and they can be smoothly aggregated using linear interpolation across their weight parameters, with significant performance gain.

More specifically, we propose to create $\boldsymbol{K}$ instances (soup ingredients) of the same GNN architecture of interest, sharing a common random initialization to ensure that the optimization trajectory does not deviate significantly during training facilitating smooth mergability. Subject to resource availability, GNN instances are trained independently in isolation across multiple GPUs using the hyperparameter configuration set $\{h_1, h_2, ..., h_K\}$ designed with slight variations in the learning rate, weight decay, seed values, etc.

Once the soup ingredients are ready, we start preparing the soup using our greedy interpolation soup procedure as illustrated in Algorithm 1, which in general follows (Wortsman et al., 2022b). The greedy soup sequentially adds each model as a potential ingredient in the soup, and only keeps the model in the soup if it leads to improving performance on the validation set. For merging two ingredients, we perform a search for an optimal interpolation ratio $\alpha \in [0, 1]$ that helps in performance gain, else the ingredient is discarded.

In comparison with several state-of-the-art GNNs, we experimentally illustrate that GNN soups prepared using ingredients having exactly the same model configuration, significantly perform better without any requirement of increasing model depth or width. For example, a 4-layer GCNII model soup prepared using 50 candidate ingredients, beats GCNII with a similar configuration on `PubMed` by 1.12 ± 0.36. Our experiments across multiple GNN architectures and datasets ranging from small graphs to large graphs {`Cora`, `Citeseer`, `PubMed`, `OGBN-ArXiv`, `OGBN-products`} strongly validate the universal effectiveness of model soups for GNNs.

Algorithm 2 Model soup with Graph Sampling

Input: ingredient_count: I_c; Graph $\mathcal{G}$, sampling_ratio: s; gpu_count: G_c; Hyperparameter Settings: $\mathcal{H}$, Sampling Criterion: S(node, edge, layer)
candidate_queue $\leftarrow \{M_1, M_2, ..., M_{I_c}\}$
soup $\leftarrow$ None
for $i = 1$ **to** $\texttt{range}((\frac{I_c}{G_c}) + 1)$ **do**
  $\{M_1, M_2, ..., M_{G_c}\} \leftarrow$ Dequeue G_c ingredients and distribute across available GPUs
  Train all distributed M_i in parallel with mini-batch sampling $\texttt{sample}(\mathcal{G}, s, S)$ and setting $h_i \in \mathcal{H}$
  soup $\leftarrow \texttt{greedy_interpolation}(\{M_1, M_2, ..., \mathrm{M}_{G_c}\} \cup$ soup)
end for
Return soup

2.4. Data-Centric Model soups and Large Graph Training Paradigms

In this section, we discuss our approach for preparing data-centric model soups in scenarios when we have resource constraints to perform message passing with the entire graph data in memory, by leveraging the SOTA graph sampling and partitioning mechanisms. As MP requires nodes aggregating information from their neighbors, the integral graph structures are inevitably preserved during forward and backward propagation, thus occupying considerable running memory and time. Following Equation 1, the key bottleneck relies in $\boldsymbol{A}^{(l)}\boldsymbol{X}^{(l)}$ for vanilla MP, requiring entire sparse adjacency matrix to be present in one GPU during training, which becomes challenging with growing topology of real-world graphs. Recently, numerous efforts with regards to graph sampling (Hamilton et al., 2017; Zou et al., 2019; Chen et al., 2018) and partitioning (Chiang et al., 2019; Zeng et al., 2019) have been proposed for approximating the iterative full-batch MP to reduce the memory consumption for training within one single GPU to mitigate the aforementioned issue and scale up GNNs.

Provided the formulation of Equation 1; graph sampling and partitioning paradigms pursue an optimal way to perform batch training, such that each batch will meet the memory constraint of a single GPU for message passing. We restate a unified illustration to elucidate the formulation of graph sampling and partitioning used for training our model soup ingredient M_i as follows:

$$\boldsymbol{X}^{(\boldsymbol{K})}_{\mathcal{B}_0}[M_i] = \tilde{\boldsymbol{A}}^{(\boldsymbol{K-1})}_{\mathcal{B}_1}\sigma(\tilde{\boldsymbol{A}}^{(\boldsymbol{K-2})}_{\mathcal{B}_2}\sigma(\cdots\sigma(\tilde{\boldsymbol{A}}^{(0)}_{\mathcal{B}_K}\boldsymbol{X}^{(0)}_{\mathcal{B}_K}[M_i]\boldsymbol{W}^{(0)}[M_i]))\cdots)\boldsymbol{W}^{(\boldsymbol{K-2})}[M_i])\boldsymbol{W}^{(\boldsymbol{K-1})}[M_i] \quad (2)$$

where $\tilde{\boldsymbol{A}}^{(\boldsymbol{l})}$ indicate the adjacency matrix for the l-th layer sampled from the full graph, $\mathcal{B}_l$ is the set of nodes sampled at l-th layer, M_i and $\boldsymbol{W}^{(l)}[M_i]$ indicate the i-th candidate ingredient and weight associated with its l-th layer. This mini-batch training approach combined with a sampling and partitioning strategy significantly helps in alleviating time consumption and memory usage which grow exponentially with the GNN depth. It is worth noting that we have explored extension to scalable infrastructure topics like distributed training with multiple GPUs to alleviate the expenses of training ingredients for model soup.

2.4.1. Model soup with Graph Sampling

Given a large graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, we explore three categories of widely-used sampling strategies: *node-wise sampling, edge-wise sampling, and layer-wise sampling*; to facilitate the training of candidates in our data-centric model soup.

Node-wise Sampling : Node-wise sampling approaches propose to sample nodes from the sampling space $\mathcal{N}(v)$, which is 1-hop neighborhood of v as $\mathcal{B}_{l+1} = \bigcup_{v \in \mathcal{B}_l}\{x|x \sim$

$Q \cdot \boldsymbol{P}_{\mathcal{N}(v)}\}$, where Q denotes the number of samples, and $\boldsymbol{P}$ is the sampling distribution. In our work, we have used GraphSAGE node sampling where $\boldsymbol{P}$ is the uniform distribution. GraphSAGE sampling strategy fixes the conventional issue of "node explosion" by clipping the neighborhood sample count to Q for each node.

Edge-wise Sampling : Edge-wise sampling proposes to sample edges from the sampling space $\mathcal{E}(v_K)$ which is K-hop neighborhood of v as $\mathcal{B}_{l+1} = \bigcup_{v \in \mathcal{B}_l}\{x|x \sim Q \cdot \boldsymbol{P}_{\mathcal{E}(v_k)}\}$, where $\mathcal{B}_{l+1}$ denotes the set of nodes induced due to edge-sampling and $\boldsymbol{P}$ is the uniform distribution. In our work, we have used uniform distribution for sampling a fixed amount of edges for each mini-batch training.

Layer-wise Sampling : $\mathcal{B}_{l+1} = \{x|x \sim Q \cdot \boldsymbol{P}_{\mathcal{N}(\mathcal{B}_l)}\}$, where $\mathcal{N}(\mathcal{B}_l) = \bigcup_{v \in \mathcal{B}_l}$ denotes the 1-hop neighbourhood of all the nodes in $\mathcal{B}_l$ In our work, following FastGCN, the sampling distribution $\boldsymbol{P}$ is designed using importance sampling where the probability for node u of being sampled is $p(u) \propto ||\boldsymbol{A}(u,:)||^2$. Experimentally, we found that layer-wise induced adjacency matrix is usually sparser than the others, which accounts sub-optimal performance in practice.

As shown in Algorithm 2, for our data-centric model soup, we first initialize our soup ingredients using a common random initialization. Depending upon the resource availability, we dequeue our ingredients across different GPUs, and train them in parallel with mini-batch sampling defined by a sampling strategy (node, edge, layer) and sampling ratio. Our model soup is prepared using greedy interpolation procedure (Algorithm 1) in an "available and added" fashion once the candidate ingredients are ready to be added to the soup.

2.4.2. Model soup with Graph Partitioning

Despite the success of sampling-based approaches, they still suffer from the issue of neighborhood explosion even with restricted node sampling when GNNs go deep leading to memory bottleneck. It has been observed that the efficiency of a mini-batch algorithm can be characterized by the notion of "embedding utilization", which is proportional to the number of links between nodes in one batch or within-batch links (Chiang et al., 2019). Such findings motivated the design of batches using graph clustering algorithms that aim to construct partitions of nodes so that there are more graph links between nodes in the same partition than nodes in different partitions. More specifically, in graph partitioning, all the GNN layers share the same subgraph induced from the entire graph based on a partitioning strategy $\boldsymbol{P}_{\mathcal{G}}$ as $\mathcal{B}_K = \mathcal{B}_{K-1} = ... = \mathcal{B}_0 = \{x|x \sim Q \cdot \boldsymbol{P}_{\mathcal{G}}\}$, such that the sampled nodes are confined in the induced subgraph.

In our work, we propose a novel and efficient strategy for graph partition-based model soup, building upon the ClusterGCN and show that it can outperform ClusterGCN by $> 1\%$ on two large graphs `OGBN-arxiv` and `Flickr`. We adopted the multi-clustering framework proposed in ClusterGCN (METIS) to partition the graph $\mathcal{G}$ into fixed K clusters $\mathcal{V} = \{V_1, V_2, ..., V_K\}$ which will be used to train each candidate ingredient of the model soup. We propose to save the cluster partition to ensure that there is *no overhead of expensive graph partitioning* again and again for improving the efficiency while training model soup ingredients. Once the clusters are decided, each candidate ingredient chooses q (hyperparameter) vertex partitions independently and forms a subgraph that can be used for training. This partitioning strategy allows us to train comparatively deeper GNNs wrt. sampling approaches. Once the candidate ingredients are trained using hyperparameter settings $\mathcal{H}$, we prepare our model soup in an "*available and added*" strategy using greedy interpolation procedure (Algorithm 1) as shown in Algorithm 3.

Algorithm 3 Model soup with Graph Partitioning

Input: ingredient_count: I_c; Graph $\mathcal{G}$, sampling_ratio: s; gpu_count: G_c; Hyperparameter Settings: $\mathcal{H}$, Epoch Count: E
Partition the graph $\mathcal{G}$ into fixed K clusters $\mathcal{V} = \{V_1, V_2, ..., V_K\}$ using METIS and save.
candidate_queue $\leftarrow \{M_1, M_2, ..., M_{I_c}\}$
soup $\leftarrow$ None
for $i = 1$ **to** $(\texttt{range}((\frac{I_c}{G_c}) + 1)$ **do**
  $\{M_1, M_2, ..., M_{G_c}\} \leftarrow$ Dequeue G_c ingredients and distribute across available GPUs
  for For All distributed M_i in parallel **do**
    for iter in $\texttt{range}(E)$ **do**
      Randomly choose q clusters $\{t_1, t_2, ..., t_q\} \in \mathcal{V}$
      Form a subgraph $\tilde{\mathcal{G}}$ with nodes $\tilde{\mathcal{V}} = \{V_{t_1}, V_{t_2}, ..., V_{t_k}\}$ and edges $\boldsymbol{A}_{\tilde{\mathcal{V}},\tilde{\mathcal{V}}}$
      Train M_i with subgraph $\tilde{\mathcal{G}}$ setting $h_i \in \mathcal{H}$
    end for
  end for
  soup $\leftarrow$ `greedy_interpolation`($\{M_1, M_2, ..., M_{G_c}\} \cup$ soup)
end for
Return soup

3. Experiments and Analysis

3.1. Dataset and Experimental Setup

Our experiments are conducted on two GPU servers equipped with RTX A6000 and RTX 3090 GPUs. The hyperparameters for soup ingredients corresponding to different datasets training are selected via our built-in efficient parameter sweeping functionalities from pre-defined ranges nearby our optimal setting in Appendix B. For our small-scale experiments, we use three standard citation network datasets: `Cora`, `Citeseer`, and `Pubmed`, while our large-scale experiments are conducted with four popular large-scale

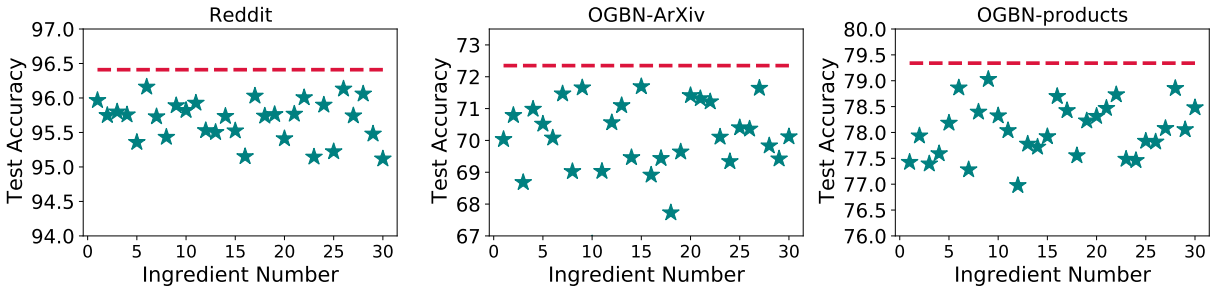

Figure 2. Plot illustrating individual performance of each participating candidate ingredient of our data-partition soup. Red dashed line indicates the performance of the model soup generated using our greedy interpolation procedure.

Table 1. Performance comparison of model soups generated using 50 candidate ingredients independently trained on the benchmarking datasets wrt. vanilla performance of several SOTA GNNs. Note that our ingredients have exactly the same architectural design as their vanilla counterpart. Results reported for vanilla GNNs are averaged across 30 independent runs.

Dataset	GCN		GCNII		JKNet		DAGNN		APPNP		SGC	
	Vanilla	Our Soup	Vanilla	Our Soup	Vanilla	Our Soup	Vanilla	Our Soup	Vanilla	Our Soup	Vanilla	Our Soup
Cora	82.39	83.52	82.19	82.77	79.06	79.89	83.39	83.45	83.64	83.70	79.31	80.22
Citeseer	71.36	72.01	72.97	73.56	66.98	68.01	73.05	73.76	72.13	73.15	72.22	73.53
PubMed	79.56	80.22	78.06	79.44	77.24	77.42	80.58	80.92	80.30	80.62	78.06	78.94
ArXiv	68.53	69.45	72.56	72.92	66.41	67.63	71.22	72.01	66.95	67.37	61.98	63.15

open benchmarks: `Flickr`, `Reddit`, `OGBN-ArXiv`, and `OGBN-products` (Appendix A). For our evaluation on our chosen datasets, we have closely followed the data split settings and metrics reported by the recent benchmark (Duan et al., 2022; Chen et al., 2021). We consider variations in the key hyperparameters {random seed, batch size, learning rate, weight decay, and dropout rate} for generating the candidate ingredient models in our model soup. Unless explicitly specified, we have used 50 candidate ingredients for small-scale graphs and 30 candidate ingredients for large-scale graphs experiments and our model soup is prepared using interpolation hyperparameter $\alpha \in [0-1]$ with a step size of 0.01 in Algorithm 1. For comparison with SOTA models, we have adopted the official implementation (Appendix C) of JKNet, DAGNN, APPNP, GCNII, SGC, ClusterGCN, and GraphSAINT. Note that while generating our model soup, we make sure to use exactly the same architectural design for our candidate ingredients to ensure a fair comparison.

3.2. Model Soups and SOTA GNNs

In this section, we conduct a systematic and extensive study to understand the harmonious adaptation of model soups for state-of-the-art GNNs on popular graph benchmarks `Cora`, `Citeseer`, `Pubmed`, and `OGBN-ArXiv`. Note that although model soups have recently attracted significant attention for large pre-trained language models, it is still unclear and unexplored if they can work for comparatively much smaller graph neural networks trained from scratch which learn from graph-structured data with relational properties unlike NLP and vision datasets having independent training samples. Model soups *provide an orthogonal way of increasing model capacity without deepening or widening GNNs* which brings many unwanted trainability issues in GNNs. Table 1 illustrates the performance summarization of our model soup generated using 50 candidate ingredients independently trained on the benchmarking datasets with respect to the vanilla performance of several SOTA GNNs following the exact same architectural design (4 layers with 256 hidden dimension) to ensure a fair comparison. The results reported for vanilla GNNs within Table 1 are averaged across 30 independent runs using different seed values selected at random from $[1-100]$ without replacement.

We first observe that across all datasets and GNN architecture, model soups outputs significantly outperform their vanilla counterpart. More noticeably, it improves GCN performance on Cora by 1.13%, GCNII performance on PubMed by 1.38%, JKNet, DAGNN, and SGC performance on OGBN-ArXiv by 1.22%,0.79%,1.17% respectively, and APPNP performance on Citeseer by 1.02%. Moreover, Table 2 illustrates the current state of various fancy architectural and regularization modifications recently proposed to facilitate deepening of GCNs and help in improving their performance. It can be clearly observed that our model soup prepared by combining the strength of 50 candidate ingredients of 2-layer GCNs can significantly outperform all these fancy methods, bolstering our claim that model explosion by deepening and widening is necessarily not the only and right direction for building high-quality generalizable GNNs.

Table 2. Illustration of the current state of various fancy architectural and regularization modifications recently proposed to facilitate deepening of GCNs and help in improving their performance. Note that our model soup prepared by combining the strength of 50 candidate ingredients of 2-layer GCNs can significantly outperform all these fancy methods.

Category	Settings	Cora			Citeseer			PubMed		
		2	16	32	2	16	32	2	16	32
Vanilla-GCN	-	81.10	21.49	21.22	71.46	19.59	20.29	79.76	39.14	38.77
Skip Connection	Residual	74.73	20.05	19.57	66.83	20.77	20.90	75.27	38.84	38.74
	Initial	79.00	78.61	78.74	70.15	68.41	68.36	77.92	77.52	78.18
	Jumping	80.98	76.04	75.57	69.33	58.38	55.03	77.83	75.62	75.36
	Dense	77.86	69.61	67.26	66.18	49.33	41.48	72.53	69.91	62.99
	Identity	82.98	67.23	40.57	68.25	56.39	35.28	79.09	79.55	73.74
Normalization	BatchNorm	69.91	61.20	29.05	46.27	26.25	21.82	67.15	58.00	55.98
	PairNorm	74.43	55.75	17.67	63.26	27.45	20.67	75.67	71.30	61.54
	NodeNorm	79.87	21.46	21.48	68.96	18.81	19.03	78.14	40.92	40.93
	CombNorm	80.00	55.64	21.44	68.59	18.90	18.53	78.11	40.93	40.90
Random Dropping	DropNode	77.10	27.61	27.65	69.38	21.83	22.18	77.39	40.31	40.38
	DropEdge	79.16	28.00	27.87	70.26	22.92	22.92	78.58	40.61	40.50
Our Model Soup (2-layer GCN)		**83.47 ± 0.32**			**72.11 ± 0.14**			**80.30 ± 0.25**		

Table 3. Performance Comaprison of GradientGCN (Jaiswal et al., 2022) soup with 50 candidate ingredients on three WebKB datasets (Texas, Wisconsin, Cornell), and Actor dataset.

	Texas	Wisconsin	Cornell	Actors
GCN	60.12±4.22	52.94±3.99	54.05±7.11	25.46±1.43
SGC	56.41±4.25	51.29±6.44	58.57±3.44	26.17±1.15
GCNII	64.28±2.93	59.19±9.07	58.51±1.66	30.95±1.04
JKNet	61.08±6.23	52.76±5.69	57.30±4.95	28.80±0.97
APPNP	60.68±4.50	54.24±5.94	58.43±3.74	28.65±1.28
GradientGCN	69.19±6.56	70.31±4.75	74.16±6.48	34.28±1.12
Ours	70.45 ±2.77	72.01 ±3.89	75.90 ±4.76	34.69 ±0.51

3.3. Data-Centric Model Soup with Graph Sampling and Graph Partitioning

In this section, we provide experimental results for preparing data-centric model soup in scenarios when we do not have the luxury of resources to perform message passing on the entire graph, by leveraging the SOTA graph sampling (node, edge, layer) and partitioning mechanisms to prepare the candidate ingredients of the soup. Table 4 illustrates the performance comparison of state-of-the-art graph sampling approaches GraphSAGE, FastGCN, and LADIES with respect to our node, edge, and layer-wise sampled soup prepared using Algorithm 2. Results of GraphSAGE, FastGCN, and LADIES are reported as mean across 30 independent runs while our soup results are reported as the mean of 5 independent runs. We additionally report the performance of graph ensemble prepared using the 30 independent runs of best-performing baseline (GraphSAGE). It can be clearly observed that our Node sampled soup performs best and comfortably outperforms all the baselines along with the graph ensemble which has a hefty inference cost, by significant margins. Similar to the sub-standard performance of layer-wise sampling approaches FastGCN and LADIES, our Layer sampled soup has comparatively low (although better than FastGCN and LADIES) performance, possibly because layer-wise induced adjacency matrix is usually sparser than the others, which accounts for its sub-optimal performance. In Table 6, we attempted to analyze the activation memory usage of activations and the hardware throughput (higher is better) of GraphSAGE (neighborhood sample size of 40) with respect to our data-centric model soup using node sampling equivalent to one-fouth of GraphSAGE (i.e., neighborhood sample size of 10) on OGBN-products. We found that with *comparatively less memory requirement*, our approach can $\sim 1\%$ better performance, eliciting the high potential in improving GNNs performance by combining the strength of multiple weak models. Next, Table 5 illustrates the performance comparison of our Graph Partition Soup with respect to two well-known graph partitioning GNNs: ClusterGCN and GraphSAINT. Results of ClusterGCN and GraphSAINT are reported as mean across 30 independent runs while our soup results are reported as mean of 5 independent runs each generated using 30 candidate ingredients using Algorithm 3. We also present the performance of graph ensemble prepared using the 30 independent runs of the best-performing baseline (ClusterGCN). Across all benchmarking datasets (Flicket, Reddit, OGBN-ArXiv, and OGBN-products), our data partition soup outperforms both GraphSAINT and ClusterGCN. More noticeably, our method beats ClusterGCN with a similar architecture design by $> 1\%$ margin on Flickr and OGBN-ArXiv dataset. In addition, Figure 2 illustrates the individual performance of each participating candidate ingredient of our data-partition soup, and it can be clearly observed that there is *orthogonal knowledge* stored in the learned weights of these networks. This gets elucidated by merging candidates and thereby improving overall performance which demonstrates the strength of our data-centric model soups.

3.4. Effect of Ingredient Count on Data-Centric Soup

In this section, we try to understand the strength of increasing ingredient count to our final data-centric model soup performance. Table 7 illustrates the performance comparison of our data partition soup on OGBN-ArXiv dataset,

Table 4. Performance comparison of state-of-the-art graph sampling approaches GraphSAGE, FastGCN, and LADIES wrt. our node, edge, and layer-wise sampled soup prepared using Algorithm 3. Results of GraphSAGE, FastGCN, and LADIES are reported as mean across 30 independent runs while our soup results are reported as mean of 5 independent runs, where each run uses 50 candidate ingredients.

Method	Flickr N=89,250 E=899,756	Reddit N=232,965 E=11,606,919	OGBN-ArXiv N=169,343 E=1,166,243	OGBN-products N= 2,449,029 E=61,859,140
GraphSAGE	53.63 ± 0.13%	96.50 ± 0.03%	71.55 ± 0.41%	80.61 ± 0.16%
FastGCN	49.89 ± 0.62%	79.50 ± 1.22%	66.10 ± 1.06%	73.46 ± 0.20%
LADIES	50.04 ± 0.39%	86.96 ± 0.37%	62.78 ± 0.89%	75.31 ± 0.56%
GraphSAGE Ensemble	53.71	96.61	71.58	80.85
Node Sampled Soup	54.47 ± 0.13 %	97.28 ± 0.08 %	72.83 ± 0.21 %	81.34± 0.28 %
Edge Sampled Soup	52.91 ± 0.56 %	92.66± 0.34 %	70.45± 0.29 %	75.58 ± 0.45 %
Layer Sampled Soup	51.08 ± 0.22 %	86.01± 0.17 %	68.30 ± 0.54 %	74.95± 0.38 %

Table 5. Performance comparison of our Graph Partition Soup with respect to two well-known graph partitioning GNNs: ClusterGCN and GraphSAINT. Results of ClusterGCN and GraphSAINT are reported as mean across 30 independent runs while our soup results are reported as mean of 5 independent runs, where each run uses 30 candidate ingredients.

Method	Flickr N=89,250 E=899,756	Reddit N=232,965 E=11,606,919	OGBN-ArXiv N=169,343 E=1,166,243	OGBN-products N= 2,449,029 E=61,859,140
ClusterGCN	51.20 ± 0.13%	95.68 ± 0.03%	71.29 ± 0.44%	78.62 ± 0.61%
GraphSAINT	51.81 ± 0.17%	95.62 ± 0.05%	70.55 ± 0.26%	75.36 ± 0.34%
ClusterGCN Ensemble	51.49	95.70	71.43	78.74
Our Data Partition Soup	52.23 ± 0.12 %	96.41± 0.08 %	72.35 ± 0.19 %	79.34 ± 0.28 %

Table 6. The memory usage of activations and the hardware throughput (higher is better) of GraphSAGE with respect to our data-centric model soup using node sampling equivalent to one-fouth of GraphSAGE on OGBN-products.

	Act. Memory(MB)	Throughput(iter/sec)	Accuracy
GraphSAGE	415.94	37.69	79.5 ± 0.36
Ours	369.51	44.30	80.42 ± 0.41

Table 7. Performance comparison of our data partition soup on OGBN-ArXiv dataset, with varying candidate ingredient counts.

Ingredient Count	10	20	30	50	100
Performance	71.426	71.691	72.353	72.388	72.814

with varying candidate ingredient counts. It can be clearly observed that *increasing candidates generally lead to better performance*, thanks to the greedy interpolation procedure. However, due to the increase in computational and time cost, we restrict the number of ingredients to 30 for all experiments related to large graphs, and 50 for small graphs.

3.5. Does intermediate communication benefit soup?

In this section, we attempt to answer another abaltion question: *How does intermediate communication across candidate ingredients (i.e. souping at intervals during training) benefit the performance of the final model soup?* To this end, we prepared a data partition model soup of GCNs using OGBN-ArXiv dataset, where we executed souping across candidate ingredients at regular intervals of 100 epochs during training. To our surprise, we found that the final soup performance is -0.745% less compared to our communication-free approach presumably due to a lack of diversity and orthogonal knowledge across the candidate ingredients. Moreover, intermediate communication incurs additional soup preparation overhead along with a new communication interval hyperparameter for optimization.

4. Background Work

Model soups (Wortsman et al., 2022b) proposed a greedy mechanism for averaging weights of multiple fine-tuned large language models with varying hyperparameters uniformly. They found that averaging many fine-tuned vision models improve out-of-domain generalization. Recently, (Li et al., 2022) introduced branch-train-merge which is at the intersection of model combination and distributed training. They consider the case where the training data is partitioned into different textual domains, then train an individual expert model for each domain. Merging all of these experts via weight averaging or ensembling to outperform the dense baseline of training one large model on all of the data. Lo-fi (Wortsman et al., 2022a) proposes splitting up a large language model fine-tuning job into multiple smaller jobs, and dedicating its fine-tuning across multiple nodes in isolation. Unlike standard data-parallel multi-node finetuning where gradients between nodes are communicated at each step, Lo-fi removes all communication between nodes during fine-tuning. (Jin et al., 2022) propose a dataless knowledge fusion method and study the problem of merging individual models built on different training data sets to obtain a single

model that performs well both across all data set domains and can generalize on out-of-domain data.

Despite many recent advancements in model parameter merging, to the surprise, it has not been explored for GNNs, where it can benefit the most, given the high demand for distributed and parallel training for graph data. Our work differs from recent works in model soups as: *firstly*, unlike over-parameterized large language models, we explore comparatively under-parameterized GNNs; *secondly*, we work with graph-structured relational data instead of independent training samples in NLP or vision; *thirdly,* we explore model merging on GNNs trained from scratch rather in the fine-tuning settings. GNNs have their own set of unique training challenges and it was unexplored if the model soup mechanism will be beneficial or hurt the performance.

5. Conclusion

In this work, we explore a principled way to scale GNNs capacity without deepening or widening which can improve its performance across multiple small and large graph. We present the first data-centric perspective of model soups to build powerful GNNs by dividing giant graph data to build independently and parallelly trained multiple comparatively weaker GNNs without any intermediate communication, and combining their strength using a greedy interpolation soup procedure to achieve state-of-the-art performance. Moreover, we provide a wide variety of model soup preparation techniques by leveraging SOTA graph sampling and graph partitioning approaches. Our future work will aim to develop a theoretical framework to explain the benefits of data-centric GNN soups.

Acknowledgement

Z. Wang is in part supported by US Army Research Office Young Investigator Award W911NF2010240 and the NSF AI Institute for Foundations of Machine Learning (IFML).

References

Mage: Automatic diagnosis of autism spectrum disorders using multi-atlas graph convolutional networks and ensemble learning. *Neurocomputing*, 469:346–353, 2022. ISSN 0925-2312.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. *ArXiv*, abs/2006.05205, 2021.

Chen, J., Ma, T., and Xiao, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. *ArXiv*, abs/2007.02133, 2020.

Chen, T., Zhou, K., Duan, K., Zheng, W., Wang, P., Hu, X., and Wang, Z. Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *arXiv preprint arXiv:2108.10521*, 2021.

Chen, T., Zhang, Z., JAISWAL, A. K., Liu, S., and Wang, Z. Sparse moe as the new dropout: Scaling dense and self-slimmable transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Duan, K., Liu, Z., Wang, P., Zheng, W., Zhou, K., Chen, T., Hu, X., and Wang, Z. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=2QrFr_U782Z.

Gao, H., Wang, Z., and Ji, S. Large-scale learnable graph convolutional networks. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y., and Tao, D. A survey on visual transformer. *ArXiv*, abs/2012.12556, 2020.

Huang, W., Rong, Y., Xu, T., Sun, F., and Huang, J. Tackling over-smoothing for general graph convolutional networks. *ArXiv*, abs/2008.09864, 2020.

Ilharco, G., Wortsman, M., Gadre, S. Y., Song, S., Hajishirzi, H., Kornblith, S., Farhadi, A., and Schmidt, L. Patching open-vocabulary models by interpolating weights. *arXiv preprint arXiv:2208.05592*, 2022.

Jaiswal, A., Li, T., Zander, C., Han, Y., Rousseau, J. F., Peng, Y., and Ding, Y. Scalp-supervised contrastive learning for

cardiopulmonary disease classification and localization in chest x-rays using patient metadata. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1132–1137. IEEE, 2021a.

Jaiswal, A., Tang, L., Ghosh, M., Rousseau, J., Peng, Y., and Ding, Y. Radbert-cl: Factually-aware contrastive learning for radiology report classification. *Proceedings of machine learning research*, 158:196–208, 2021b.

Jaiswal, A., Wang, P., Chen, T., Rousseau, J. F., Ding, Y., and Wang, Z. Old can be gold: Better gradient flow can make vanilla-gcns great again. *arXiv preprint arXiv:2210.08122*, 2022.

Jaiswal, A., Chen, T., Rousseau, J. F., Peng, Y., Ding, Y., and Wang, Z. Attend who is weak: Pruning-assisted medical image localization under sophisticated and implicit imbalances. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 4987–4996, 2023.

Jin, X., Ren, X., Preotiuc-Pietro, D., and Cheng, P. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849*, 2022.

Juneja, J., Bansal, R., Cho, K., Sedoc, J., and Saphra, N. Linear connectivity reveals generalization strategies. *arXiv preprint arXiv:2205.12411*, 2022.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

Li, G., Müller, M., Thabet, A. K., and Ghanem, B. Can gcns go as deep as cnns? *ArXiv*, abs/1904.03751, 2019.

Li, G., Müller, M., Ghanem, B., and Koltun, V. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pp. 6437–6449. PMLR, 2021.

Li, M., Gururangan, S., Dettmers, T., Lewis, M., Althoff, T., Smith, N. A., and Zettlemoyer, L. Branch-train-merge: Embarrassingly parallel training of expert language models. *ArXiv*, abs/2208.03306, 2022.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Li, T., Shetty, S., Kamath, A., Jaiswal, A., Jiang, X., Ding, Y., and Kim, Y. Cancergpt: Few-shot drug pair synergy prediction using large pre-trained language models. *arXiv preprint arXiv:2304.10946*, 2023.

Lin, Q., Yu, S., Sun, K., Zhao, W., Alfarraj, O., Tolba, A., and Xia, F. Robust graph neural networks via ensemble learning. *Mathematics*, 10(8), 2022. doi: 10.3390/math10081300.

Liu, H., Yang, Y., and Wang, X. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 8653–8661, 2021.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Liu, S., Chen, T., Zhang, Z., Chen, X., Huang, T., Jaiswal, A., and Wang, Z. Sparsity may cry: Let us fail (current) sparse neural networks together! *arXiv preprint arXiv:2303.02141*, 2023.

Mao, Z., Jaiswal, A., Wang, Z., and Chan, S. H. Single frame atmospheric turbulence mitigation: A benchmark study and a new physics-inspired transformer model. *ArXiv*, abs/2207.10040, 2022.

NT, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *ArXiv*, abs/1905.09550, 2019.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Thekumparampil, K. K., Wang, C., Oh, S., and Li, L.-J. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wortsman, M., Gururangan, S., Li, S., Farhadi, A., Schmidt, L., Rabbat, M., and Morcos, A. S. lo-fi: distributed fine-tuning without communication. *arXiv preprint arXiv:2210.11948*, 2022a.

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022b.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Yang, L., Luo, L., Xin, L., Zhang, X., and Zhang, X. Dagnn: Demand-aware graph neural networks for session-based recommendation. *arXiv preprint arXiv:2105.14428*, 2021.

You, Y., Chen, T., Wang, Z., and Shen, Y. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2124–2132, 2020.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. *ArXiv*, abs/1909.12223, 2020.

Zheng, M., Gao, P., Zhang, R., Wang, X., Li, H., and Dong, H. End-to-end object detection with adaptive clustering transformer. *ArXiv*, abs/2011.09315, 2021a.

Zheng, W., Huang, E. W., Rao, N., Katariya, S., Wang, Z., and Subbian, K. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. *arXiv preprint arXiv:2111.04840*, 2021b.

Zheng, W., Sharan, S., Jaiswal, A. K., Wang, K., Xi, Y., Xu, D., and Wang, Z. Outline, then details: Syntactically guided coarse-to-fine code generation. *arXiv preprint arXiv:2305.00909*, 2023.

Zhou, K., Dong, Y., Wang, K., Lee, W. S., Hooi, B., Xu, H., and Feng, J. Understanding and resolving performance degradation in deep graph convolutional networks. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021a.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34:21834–21846, 2021b.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *NeurIPS*, 2019.

A. Dataset Details

Table 10 provided provides the detailed properties and download links for all adopted datasets. We adopt the following benchmark datasets since i) they are widely applied to develop and evaluate GNN models, especially for deep GNNs studied in this paper; ii) they contain diverse graphs from small-scale to large-scale or from homogeneous to heterogeneous; iii) they are collected from different applications including citation network, social network, etc.

Table 8. Graph datasets statistics and download links.

Dataset	Nodes	Edges	Classes	Download Links
Cora	2,708	5,429	7	https://github.com/kimiyoung/planetoid/raw/master/data
Citeseer	3,327	4,732	6	https://github.com/kimiyoung/planetoid/raw/master/data
PubMed	19,717	44,338	3	https://github.com/kimiyoung/planetoid/raw/master/data
OGBN-ArXiv	169,343	1,166,243	40	https://ogb.stanford.edu/
Flickr	89,250	899,756	7	PyTorchGeometic:https://arxiv.org/abs/1907.04931
Reddit	232,965	11,606,919	41	PyTorchGeometic:https://arxiv.org/abs/1706.02216
OGBN-products	2,449,029	61,859,140	47	https://ogb.stanford.edu/

B. Experimental setting of our large-scale datasets

Table 9. The searched optimal hyperparameters for all tested methods for data-centric soup.

Method	Flickr Split: 0.50/0.25/0.25	Reddit Split: 0.66 / 0.10 / 0.24	OGBN-products Split: 0.10 / 0.02 / 0.88
GraphSAGE(Hamilton et al., 2017)	LR: 0.0001, WD: 0.0001, DP: 0.5, EP: 50, HD: 512, L: 4, BS: 1000	LR: 0.0001, WD: 0.0 DP: 0.2, EP: 50, HD: 512, L: 4, BS: 1000	LR: 0.001, WD: 0.0 DP: 0.5, EP: 50, HD: 512, L: 4, BS: 1000
FastGCN(Chen et al., 2018)	LR: 0.001, WD: 0.0002, DP: 0.1 EP: 50, HD: 512, L: 2, BS: 5000	LR: 0.01, WD: 0.0 DP: 0.5, EP: 50, HD: 256, L: 2, BS: 5000	LR: 0.01, WD: 0.0 DP: 0.2 EP: 50, HD: 256, L: 2, BS: 5000
LADIES(Zou et al., 2019)	LR: 0.001, WD: 0.0002, DP: 0.1, EP: 50, HD: 512, L: 2, BS: 5000	LR: 0.01, WD: 0.0001 DP: 0.2, EP: 50, HD: 512, L: 2, BS: 5000	LR: 0.01, WD: 0.0 DP: 0.2 EP: 30, HD: 256, L: 2, BS: 5000
ClusterGCN(Chiang et al., 2019)	LR: 0.001, WD: 0.0002, DP: 0.2, EP: 30, HD: 256, L: 2, BS: 5000	LR: 0.0001, WD: 0.0 DP: 0.5 EP: 50, HD: 256, L: 4, BS: 2000	LR: 0.001, WD: 0.0001 DP: 0.2, EP: 40, HD: 128, L: 4, BS: 2000
GraphSAINT(Zeng et al., 2019)	LR: 0.001, WD: 0.0004, DP: 0.2 EP: 50, HD: 512, L: 4, BS: 5000	LR: 0.01, WD: 0.0002 DP: 0.7 EP: 30, HD: 128, L: 2, BS: 5000	LR: 0.01, WD: 0.0 DP: 0.2, EP: 40, HD: 128, L: 2, BS: 5000

C. Code adaptation URL for our baselines

Table 10. Method and their official implementation used in our work.

Method	Download URL
JKNet(Xu et al., 2018)	https://github.com/mori97/JKNet-dgl
DAGNN(Yang et al., 2021)	https://github.com/vthost/DAGNN
APPNP(Klicpera et al., 2019)	https://github.com/gasteigerjo/ppnp
GCNII(Chen et al., 2020)	https://github.com/chennnM/GCNII
SGC(Wu et al., 2019)	https://github.com/Tiiiger/SGC
ClusterGCN(Chiang et al., 2019)	https://github.com/benedekrozemberczki/ClusterGCN
GraphSAINT(Zeng et al., 2019)	https://github.com/GraphSAINT/GraphSAINT