Bag of Tricks for Training Deeper Graph Neural Networks: A Comprehensive Benchmark Study

Tianlong Chen[1*], Kaixiong Zhou[2*], Keyu Duan[2], Wenqing Zheng[1], Peihao Wang[1], Xia Hu[2], Zhangyang Wang[1]
[1]University of Texas at Austin, [2]Rice University
{tianlong.chen,w.zheng,peihaowang,atlaswang}@utexas.edu,
{kz34,k.duan,xia.hu}@rice.edu

Abstract

Training deep graph neural networks (GNNs) is notoriously hard. Besides the standard plights in training deep architectures such as vanishing gradients and overfitting, the training of deep GNNs also uniquely suffers from over-smoothing, information squashing, and so on, which limits their potential power on large-scale graphs. Although numerous efforts are proposed to address these limitations, such as various forms of skip connections, graph normalization, and random dropping, it is difficult to disentangle the advantages brought by a deep GNN architecture from those "tricks" necessary to train such an architecture. Moreover, the lack of a standardized benchmark with fair and consistent experimental settings poses an almost insurmountable obstacle to gauging the effectiveness of new mechanisms. In view of those, we present the first fair and reproducible benchmark dedicated to assessing the "tricks" of training deep GNNs. We categorize existing approaches, investigate their hyperparameter sensitivity, and unify the basic configuration. Comprehensive evaluations are then conducted on tens of representative graph datasets including the recent large-scale Open Graph Benchmark (OGB), with diverse deep GNN backbones. Based on synergistic studies, we discover the combo of superior training tricks, that lead us to attain the new state-of-the-art results for deep GCNs, across multiple representative graph datasets. We demonstrate that an organic combo of initial connection, identity mapping, group and batch normalization has the most ideal performance on large datasets. Experiments also reveal a number of "surprises" when combining or scaling up some of the tricks. All codes are available at https://github.com/VITA-Group/Deep_GCN_Benchmarking.

1 Introduction

Graph neural networks (GNNs) [1] are powerful tools for modeling graph-structured data, and have been widely adopted in various real world scenarios, including inferring individual relations in social and academic networks [2–8], modeling proteins for drug discovery [9, 10], improving predictions of recommendation systems [11, 3], segmenting large point clouds [12, 13], among others. While many classical GNNs have no more than just a few layers at most, recent advances attempt to investigate deeper GNN architectures [13, 14], and there are massive examples of graphs on which deeper GNNs helps, such as "geometric" graphs representing structures such as molecules, point clouds [13], or meshes [15]. Deeper architectures are also found with superior performance on large-scale graph datasets such as the latest Open Graph Benchmark (OGB) [16].

However, training deep GNNs is notoriously challenging [13]. Besides the standard plights in training deep architectures such as *vanishing gradients* and *overfitting*, the training of deep GNNs suffers

[*]Equal Contribution.

Preprint. Under review.

several unique barriers that limits their potential power on large-scale graphs. One of them is *over-smoothing*, i.e., the node features tending to become indistinguishable as the result of performing several recursive neighborhood aggregation [17]. This behaviour was first observed in GCN models [18, 19], which act similarly to low-pass filters, and presents deep GNNs from effectively modeling the higher-order dependencies from multi-hop neighbors. Another phenomenon is the ***bottleneck*** phenomenon: while deep GNNs are expected to utilize longer-range interactions, the structure of the graph often results in the exponential growth of the receptive field, causing "over-squashing" of information from exponentially many neighbours into fixed-size vectors [20], and explaining why some deeper GNNs do not improve in performance compared to their shallower peers.

To address the aforementioned roadblocks, typical approaches could be categorized as *architectural modifications*, and *regularization & normalization*: both we view as "training tricks". The formal includes various types of residual connections [13, 21–23, 18, 22–25]; and the latter include random edge dropping [26, 27], pairwise distance normalisation between node features (PairNorm) [28], node-wise mean and variance normalisation (NodeNorm) [29], among many other normalization options [30–34]. While these techniques in general contribute to effective training of deep GNNs with tens of layers, their gains are not always significant nor consistent [29]. Furthermore, it is often non-straightforward to disentangle the gain by deepening the GNN architecture, from the "tricks" necessary to train such a deeper architecture. In some extreme situations, contrary to the initial belief, newly proposed training techniques could improve shallower GNNs even to outperform deeper GNNs [29], making our pursuit of depth unpersuasive. Those observations have shed light on a **missing key knob** on studying deeper GNNs: we lack a standardized benchmark that could offer fair and consistent comparison on the effectiveness of deep GCN training techniques. Without isolating the effects of deeper architectures from their training "tricks", one might never reach convincing answers whether deeper GNNs with *ceteris paribus* should perform better.

1.1 Our Contributions

Aiming to establish such a fair benchmark, our first step is to thoroughly investigate the design philosophy and implementation details on dozens of poplular deep GNN training techniques, including various residual connections, graph normalization, and random dropping. The summarization could be founded in Tables 1, A9, A10, and A11. Somehow unfortunately, we find that even sticking to the same dataset and GNN backbone, the hyperparameter configurations (e.g., hidden dimension, learning rate, weight decay, dropout rate, training epochs, early stopping patience) are highly inconsistently implemented, often varying case-to-case, which make it troubling to draw any fair conclusion.

To this end, in this paper we carefully examine those sensitive hyperparameters and unify them into one "sweetpoint" hyperparameter set, to be adopted by all experiments. Such lays the foundation for a fair and reproducible benchmark of training deep GNN. Then, we comprehensively explore the diverse combinations of the available training techniques, over *tens of* classical graph datasets with commonly used deep GNN backbones. Our comprehensive study turns out to be worthy. We conclude the baseline training configurations with the superior combo of training tricks, that lead us to attaining the new state-of-the-art results across multiple representative graph datasets including OGB. Specifically, we show that an organic combination of initial connection, identity mapping, group and batch normalizations has preferably strong performance on large datasets. All implementations and benchmarks are publicly available, and our maintenance plan is presented in Appendix A1.

Our experiments also reveals a number of "surprises". For example, ❶ as we empirically show, while initial connection [21] and jumping connection [24] are both "beneficial" training tricks when applied alone, combining them together deteriorates deep GNN performance. ❷ Although dense connection brings considerable improvement on large-scale graphs with deep GNNs, it sacrifices the training stability to a severe extent. ❸ As another example, the gain from NodeNorm [29] become diminishing when applied to large-scale datasets or deeper GNN backbones. ❹ Moreover, using random dropping techniques alone often yield unsatisfactory performance. ❺ Lastly, we observe that adopting initial connection and group normalization is universally effective across tens of classical graph datasets. Those findings urge more synergistic rethinking of those seminal works.

2 Related Works

Graph Neural Networks (GNNs) and Training Deep GNNs. GNNs [35–41, 7, 42–44] have established state-of-the-art results on various tasks [36, 38, 45–47, 6, 8]. There are also numerous GNN variants [48–50, 36, 51, 52, 11, 53–55, 37, 56]. For example, Graph Convolutional Networks

(GCNs) are widely adopted, which can be divided into spectral domain based methods [57, 36] and spatial domain bases methods [58, 51].

Unlike other deep architectures, not every useful GNN has to be deep. For example, many graphs, such as social networks and citation networks, are "small-world" [59], i.e., one node can reach any other node in a few hops. Hereby, adding more layers does not help model over those graphs since the receptive fields of just stacking a few layers would already suffice to provide global coverage. Many predictions in practice, such as in social networks, also mainly rely on short-range information from the local neighbourhood of a node. On the other hand, when the graph data is large in scale, does not have small-world properties, or its task requires long-range information to make prediction, then *deeper GNNs do become necessary*. Examples include molecular graphs, as chemical properties of a molecule can depend on the atom combination at its opposite sides [60]. Point clouds or meshes also benefit from GNN depth to capture a whole object and its context in the visual scene [13, 15].

With the prosperity of GNNs, understanding their training mechanism and limitation is of remarkable interest. As GNNs stack spatial aggregations recursively [1, 51], the node representations will collapse to indistinguishable vectors [19, 17, 61]. Such over-smoothing phenomenon hinders the training of deep GNNs and the dependency modeling to high-order neighbors. Recently, there has been a series of techniques developed to relieve the over-smoothing issue, including *skip connection*, *graph normalization*, *random dropping*. They will be detailed next.

Skip Connection. Motivated by ResNets [62], the skip connection was applied to GNNs to exploit node embeddings from the preceding layers, to relieve the over-smoothing issue. There are several skip connection patterns in deep GNNs, including: (1) residual connection connecting to the last layer [13, 18], (2) initial connection connecting to the initial layer [21–23], (3) dense connection connecting to all the preceding layers [13, 18, 63, 64], and (4) jumping connection combining all all the preceding layers only at the final graph convolutional layer [24, 25]. Note that the last one of jumping connection is a simplified version of dense connection by omitting the complex skip connections at the intermediate layers.

Graph Normalization. A series of normalization techniques has been developed for deep GNNs, including batch normalization (BatchNorm) [30], pair normalization (PairNorm) [28], node normalization (NodeNorm) [29], mean normalization (MeanNorm) [33], differentiable group normalization (GroupNorm) [34], and more. Their common mechanism is to re-scale node embeddings over an input graph to constrain pairwise node distance and thus alleviate over-smoothing. While BatchNorm and PairNorm normalize the whole input graph, GroupNorm first clusters node into groups and then normalizes each group independently. NodeNorm and MeanNorm operate on each node by re-scaling the associated node embedding with its standard deviation and mean values, respectively.

Random Dropping. Dropout [65] refers to randomly dropping hidden units in a neural network with a pre-fixed probability, which effectively prevents over-fitting. Similar ideas have been inspired for GNNs. DropEdge [26] and DropNode [27] have been proposed to randomly remove a certain number of edges or nodes from the input graph at each training epoch. Alternatively, they could be regarded as data augmentation methods, which helps relieve both the over-fitting and over-smoothing issues in training very deep GNNs.

Table 1: Configurations of basic hyperparameters adopted to implement different approaches for training deep GNNs on Cora [36]. Other graph datasets are refer to Section A3.

Methods	Total epoch	Learning rate & Decay	Weight decay	Dropout	Hidden dimension
Chen et al. (2020) [21]	100	0.01	5e-4	0.6	64
Xu et al. (2018) [24]	-	0.005	5e-4	0.5	{16, 32}
Klicpera et al. (2018) [22]	10000	0.01	5e-3	0.5	64
Zhang et al. (2020) [23]	1500	{0.001, 0.005, 0.01}	-	{0.1, 0.2, 0.3, 0.4, 0.5}	64
Luan et al. (2019) [64]	3000	1.66e-4	1.86e-2	0.65277	1024
Liu et al. (2020) [25]	100	0.01	0.005	0.8	64
Zhao et al. (2019) [28]	1500	0.005	5e-4	0.6	32
Min et al. (2020) [66]	200	0.005	0	0.9	-
Zhou et al. (2020) [34]	1000	0.005	5e-4	0.6	-
Zhou et al. (2020) [29]	50	0.005	1e-5	0	-
Rong et al. (2020) [26]	400	0.01	5e-3	0.8	128
Zou et al. (2020) [67]	100	0.001	-	-	256
Hasanzadeh et al. (2020) [68]	2000	0.005	5e-3	0	128

3 A Fair and Scalable Study of Individual Tricks

3.1 Prerequisite: Unifying the Hyperparameter Configuration

We carefully examine previous implementations of deep GNNs [21, 22, 25, 64, 24, 66, 28, 34, 26, 29], and list all their basic hyperparameters in Table 1, A9, A10, and A11. Those hyperparameters play a significant role in those methods' achievable performance, but their inconsistency challenges fair comparison of training techniques, which has been traditionally somehow overlooked in literature.

We tune them all with an exhaustive grid search, and identify the most common and effective setting as in Table 2, by choosing the configuration that performs the best across diverse backbones with different layers on the corresponding dataset. That configuration includes learning rate, weight decay, dropout rate, and the hidden dimension of multilayer perceptrons (MLP) in GNN. We recommend it as a "sweet point" hyperparameter configuration, and strictly follow it in our experiments.

For all experiments, deep GNNs are trained with a maximum of 1000 epochs with an early stop patience of 100 epochs. *One hundred independent repetitions* are conduct for each experiment, and average performance with the standard deviations of the node classification are reported in Table 4, A14, 3, A13, 5, A15, A16, and A17. We perform a comprehensive benchmark study of dozens of training approaches on four classical datasets, i.e., Cora, Citeseer, PubMed [36], and OGBN-ArXiv [16] with $2/16/32$ layers GCN [36] and simple graph convolutional (SGC) [69] backbones. More details of adopted datasets are collected in Appendix 7. The Pytorch framework and Pytorch Geometric [70] are used for all of our implementations. More specific setups for each group of training techniques are included in Section 3.3, 3.2, and 3.4.

Table 2: The "sweet point" hyperparameter configuration we used on representative datasets.

Settings	Cora	Citeseer	PubMed	OGBN-ArXiv
{Learning rate, Weight decay, Dropout, Hidden dimension}	$\{0.005, 5\times10^{-4}, 0.6, 64\}$	$\{0.005, 5\times10^{-4}, 0.6, 256\}$	$\{0.01, 5\times10^{-4}, 0.5, 256\}$	$\{0.005, 0, 0.1, 256\}$

3.2 Skip Connection

Table 3: Test accuracy (%) under different skip connection mechanisms. Experiments are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Performance are averaged from 100 independent repetitions, and their standard deviation are presented in Table A13.

Backbone	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	Residual	74.73	20.05	19.57	66.83	20.77	20.90	75.27	38.84	38.74	70.19	69.34	65.09
	Initial	79.00	**78.61**	**78.74**	70.15	**68.41**	**68.36**	77.92	**77.52**	**78.18**	70.16	70.50	70.23
	Jumping	80.98	76.04	75.57	69.33	58.38	55.03	77.83	75.62	75.36	**70.24**	**71.83**	**71.87**
	Dense	77.86	69.61	67.26	66.18	49.33	41.48	72.53	69.91	62.99	70.08	71.29	70.94
	None	**82.38**	21.49	21.22	**71.46**	19.59	20.29	**79.76**	39.14	38.77	69.46	67.96	45.48
SGC	Residual	**81.77**	82.55	80.14	71.68	71.31	71.00	78.87	79.86	79.07	69.09	66.52	61.83
	Initial	81.40	**83.66**	83.77	71.60	**72.16**	**72.25**	**79.11**	79.73	79.74	68.93	69.24	69.15
	Jumping	77.75	83.42	**83.88**	69.96	71.89	71.88	77.42	**79.99**	**80.07**	68.76	70.61	70.65
	Dense	77.31	81.24	77.66	70.99	67.75	66.35	77.12	72.77	74.84	**69.39**	**71.42**	**71.52**
	None	79.31	75.98	68.45	**72.31**	71.03	61.92	78.06	69.18	66.61	61.98	41.58	34.22

Formulations. Skip connections [13, 21–23, 18, 22–25] alleviate the vanishing gradient and over-smoothing, and substantially improve the accuracy and stability of training deep GCNs.

Let $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ denotes the graph data, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ represents the adjacent matrix, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n] \in \mathbb{R}^{n\times d}$ is the corresponding input node features, n is the number of nodes in the graph $\mathcal{G}$, and d is the dimension of each node feature $\mathbf{x}_i$. In the adjacent matrix $\mathbf{A}$, the values of entries can be weighted, i.e., $\mathbf{A}_{i,j} \in (0, 1)$. For a L layers GCN, we can apply various types of skip connections after certain graph convolutional layers with the current and preceding embeddings $\mathbf{X}^l$, $0 \le l \le L$. Four representative types of skip connections are investigated:

⋆ *Residual Connection.* [13, 18] $\mathbf{X}^l = (1-\alpha)\cdot\mathbf{X}^l + \alpha\cdot\mathbf{X}^{l-1}$.

⋆ *Initial Connection.* [21–23] $\mathbf{X}^l = (1-\alpha)\cdot\mathbf{X}^l + \alpha\cdot\mathbf{X}^0$.

⋆ *Dense Connection.* [13, 18, 63, 64] $\mathbf{X}^l = \mathrm{COM}(\{\mathbf{X}^k, 0 \le k \le l\})$.

⋆ *Jumping Connection.* [24, 25] $\mathbf{X}^L = \mathrm{COM}(\{\mathbf{X}^k, 0 \le k \le L\})$.

where α in *residual* and *initial connections* is a hyperparameter to weigh the contribution of node features from the current layer l and previous layers. In our case, the best performing α values are

searched from $\{0.1, 0.2, 0.4, 0.6, 0.8\}$ for each experiment. *Jumping connection* as a simplified case of *dense connection* is only applied at the end of whole forward propagation process to combine the node features from all previous layers.

We lastly introduce the combination functions COM used in dense and jumping connections: ❶ *Concat*, $\mathbf{X}^l = \text{MLP}([\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^l])$; ❷ *Maxpool*, $\mathbf{X}^l = max(stack([\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^l]))$, the operation max return the maximum value of each column, i.e., each dimension in node features; ❸ *Attention*, $\mathbf{X}^l = squeeze(\mathbf{S}\hat{\mathbf{X}})$, $\mathbf{S} = \sigma(\text{MLP}(\mathbf{X}))$, $\hat{\mathbf{X}} = stack([\mathbf{X}^0, \mathbf{X}^1, \ldots, \mathbf{X}^l])$. $\sigma(\cdot)$ is the activation function where $sigmoid$ is usually adopted [24, 25]. $stack$ and $squeeze$ are utilized to rearrange the data dimension so that dimension can be matched throughout the computation. Note that, for every dense connection and jumping connection experiment, we examine all three options for COM in the set {*Concat*, *Maxpool*, *Attention*} and report the one attaining the best performance.

Experimental observations. From Table 3 and A13, we can draw the following observations: (i) In general, initial and jumping connection perform relatively better than the others, especially for training very deep GNNs (i.e., ≥ 16 layers). It demonstrates that these two skip connections more effectively mitigate the over-smoothing issue, assisting graph neural networks going deeper. (ii) For shallow GCN backbones (e.g., 2 layers), using skip connections can incur performance degradation, yet ot becomes beneficial on the large scale OGBN-ArXiv dataset. Meanwhile, SGC backbones benefit from skip connections in most of the case. (iii) Residual connection only works for 2-layer SGC on Cora, implying that although it also brings in shortcuts, their effects to overcome over-smoothening are diminished by increasing depth or growing dataset size, presumably due to its more "local" information flow (directly from only the preceding one layer), compared to initial and dense connections whose information flow can be directly imported from much earlier layers. (iv) Although dense connection on average brings significant accuracy improvements on OGBN-ArXiv with SGC, it sacrifices the training stability and leads to considerable performance variance, as consistently shown by Table A13. (v) Figure 1 reveals that skip connections substantially accelerate the training of deep GNNs, which is aligned with the analysis by concurrent work [71].

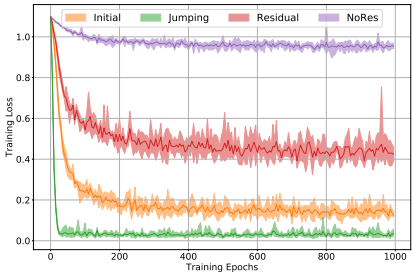

Figure 1: Training loss under different skip connections on PubMed with 16 layers SGC. Dense connection is omitted due to different scales of loss magnitude. More results are included in Section A5.

3.3 Graph Normalization

Table 4: Test accuracy (%) under different graph normalizations. Experiments are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Performance are averaged from 100 independent repetitions, and their standard deviation are presented in Table A14.

Backbone	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	BatchNorm	69.91	61.20	**29.05**	46.27	26.25	21.82	67.15	58.00	55.98	70.44	70.52	68.74
	PairNorm	74.43	**55.75**	17.67	63.26	**27.45**	20.67	75.67	71.30	61.54	65.74	65.37	63.32
	NodeNorm	79.87	21.46	21.48	68.96	18.81	19.03	78.14	40.92	40.93	70.62	70.75	29.94
	MeanNorm	**82.49**	13.51	13.03	70.86	16.09	7.70	78.68	18.92	18.00	69.54	70.40	56.94
	GroupNorm	82.41	41.76	27.20	71.30	26.77	**25.82**	**79.78**	**70.86**	**63.91**	69.70	70.50	68.14
	CombNorm	80.00	55.64	21.44	68.59	18.90	18.53	78.11	40.93	40.90	**70.71**	**71.77**	**69.91**
	NoNorm	82.43	21.78	21.21	**71.40**	19.78	19.85	79.75	39.18	39.00	69.45	67.99	46.38
SGC	BatchNorm	79.32	15.86	14.40	61.60	17.34	17.82	76.34	54.22	29.49	**68.58**	65.54	62.33
	PairNorm	80.78	71.26	51.03	69.76	60.14	50.94	75.81	68.89	62.14	60.72	39.69	26.67
	NodeNorm	78.09	**78.77**	73.93	63.42	61.81	60.22	71.64	71.50	73.30	63.21	26.81	16.18
	MeanNorm	80.22	48.29	30.07	70.78	38.27	28.27	75.07	47.29	41.32	54.86	21.74	18.97
	GroupNorm	**82.81**	75.81	**74.94**	72.32	67.54	61.75	**78.87**	**76.43**	**74.62**	66.12	**67.29**	**66.11**
	CombNorm	77.65	75.16	74.45	63.66	59.97	54.52	71.67	71.50	72.23	65.73	54.37	47.52
	NoNorm	79.38	75.93	68.75	**72.36**	**71.06**	**62.64**	78.01	69.06	66.55	61.96	41.43	34.24

Formulations. Graph normalization mechanisms [30–32, 28, 29, 33, 34] are also designed to tackle over-smoothing. Nearly all graph normalization techniques are applied upon the intermediate node embedding matrix, after passing through several graph convolutions. We use $\mathbf{X}^l = [\mathbf{x}_1^l, \mathbf{x}_2^l, ..., \mathbf{x}_n^l] \in \mathbb{R}^{n \times F}$ to denote the aggregated node embedding matrix in layer l, where n is the number of nodes and F is the dimension of each intermediate node representation i.e, $\mathbf{x}_i^l \in \mathbb{R}^{1 \times F}$ (we omit l in later descriptions for notation's simplicity). Correspondingly, $\mathbf{x}_{(k)} \in \mathbb{R}^{n \times 1}$ represents the k-th column of $\mathbf{X}$, recording values of the k-th dimension of each feature embedding. Our investigated normalization mechanisms are formally depicted as follows.

$\star$ *PairNorm.* [28] $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$, $\text{PairNorm}(\mathbf{x}_i; s) = \frac{s\cdot\tilde{\mathbf{x}}_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i\|_2^2}}$.

$\star$ *NodeNorm.* [29] $\text{NodeNorm}(\mathbf{x}_i; p) = \frac{\mathbf{x}_i}{std(\mathbf{x}_i)^{\frac{1}{p}}}$.

$\star$ *MeanNorm.* [33] $\text{MeanNorm}(\mathbf{x}_{(k)}) = \mathbf{x}_{(k)} - \mathbb{E}[\mathbf{x}_{(k)}]$.

$\star$ *BatchNorm.* [30] $\text{BatchNorm}(\mathbf{x}_{(k)}) = \gamma \cdot \frac{\mathbf{x}_{(k)} - \mathbb{E}[\mathbf{x}_{(k)}]}{std(\mathbf{x}_{(k)})} + \beta$.

$\star$ *GroupNorm.* [34] $\text{GroupNorm}(\mathbf{X}; G, \lambda) = \mathbf{X} + \lambda \cdot \sum_{g=1}^{G} \text{BN}(\tilde{\mathbf{X}}_g), \tilde{\mathbf{X}}_g = \text{softmax}(\mathbf{X}\cdot\mathbf{U})[:, g] \circ \mathbf{X}$

where s in *PairNorm* is a hyperparameter controlling the average pair-wise variance and we choose $s = 1$ in our case. p in *NodeNorm* denotes the normalization order and our paper uses $p = 2$. $std(\cdot)$ and $\mathbb{E}[\cdot]$ are functions that calculate the standard deviation and mean value, respectively. $\circ$ in *GroupNorm* is the operation of a row-wise multiplication, $\mathbf{U} \in \mathbb{R}^{F\times G}$ represents a learned transformation, G means the number of groups, and λ is the skip connection coefficient used in *GroupNorm*. Specifically, in our implementation, we adopt $(G, \lambda) = (10, 0.03)$ for Cora, $(5, 0.01)$ for Pubmed, and $(10, 0.005)$ for Citeceer and OGBN-ArXiv. *CombNorm* normalizes layer-wise graph embeddings by successively applying the two top-performing *GroupNorm* and *NodeNorm*.

Experimental observations. Extensive results are collected in Table 4 and A14. We conclude in the following: (i) *Dataset dependent observations.* On small datasets such as Cora and Citeceer, non-parametric normalization methods such as NodeNorm and PairNorm often present on-par performance with parametric GroupNorm, and GroupNorm displays better performance than CombNorm (i.e., GroupNorm + NodeNorm). On the contrary for large graphs such as OGBN-ArXiv, GroupNorm and CombNorm show a superior generalization ability than non-parametric normalizations, and adding NodeNorm on top of GroupNorm leads to extra improvements on deep GCN backbones. (ii) *Backbone dependent observations.* We observe that even the same normalization technique usually appears diverse behaviors on different GNN backbones. For example, PairNorm is nearly always better than NodeNorm on 16 and 32 layers of GCN on three small datasets, and worse than NodeNorm on OGBN-ArXiv. However the case is almost reversed when we experiment with the SGC backbone, where NodeNorm is outperforming PairNorm on small graphs and inferior on OGBN-ArXiv, for $16/32$ layers GNNs. Meanwhile, we notice that SGC with normalization tricks achieve similar performance to GCN backbone on three small graphs, while consistently fall behind on large-scale OGBN graphs. (iii) *Stability observations.* In general, training deep GNNs with normalization incurs more instability (e.g., large performance variance) along with the increasing depth of GNNs. BatchNorm demonstrates degraded performance with large variance on small graphs, and yet it is one of the most effective mechanism on the large graph with outstanding generalization and improved stability. (iv) Overall, across the five investigated norms, GroupNorm has witnessed most of top-performing scenarios, and PairNorm/NodeNorm occupied second best performance in GCN/SGC respectively, dependent on the scale of datasets. In contrast, MeanNorm performs the worst in training deep GCN/SGC, in terms of both achievable accuracy and performance stability.

3.4 Random Dropping

Table 5: Test accuracy (%) under different random dropping. Experiments are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Performance are averaged from 100 repetitions, and standard deviations are reported in Table A15,A16,A17. Dropout rates are tuned for best performance.

Dataset	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	No Dropout	80.68	**28.56**	**29.36**	71.36	23.19	**23.03**	79.56	39.85	40.00	**69.53**	66.14	41.96
	Dropout	**82.39**	21.60	21.17	**71.43**	19.37	20.15	**79.79**	39.09	39.17	69.40	67.79	45.41
	DropNode	77.10	27.61	27.65	69.38	21.83	22.18	77.39	40.31	40.38	66.67	67.17	43.81
	DropEdge	79.16	28.00	27.87	70.26	22.92	22.92	78.58	40.61	40.50	68.67	66.50	**51.70**
	LADIES	77.12	28.07	27.54	68.87	22.52	22.60	78.31	40.07	40.11	66.43	62.05	40.41
	DropNode+Dropout	81.02	22.24	18.81	70.59	24.49	18.23	78.85	40.44	40.37	68.66	68.27	44.18
	DropEdge+Dropout	79.71	20.45	21.10	69.64	19.77	18.49	77.77	40.71	40.51	66.55	**68.81**	49.82
	LADIES+Dropout	78.88	19.49	16.92	69.02	**27.17**	18.54	78.53	**41.43**	**40.70**	66.35	65.13	39.99
SGC	No Dropout	77.55	73.99	66.80	71.80	**72.69**	70.50	77.59	69.74	67.81	**62.34**	**42.54**	**34.76**
	Dropout	79.37	75.91	68.40	**72.35**	71.21	62.35	78.04	69.12	66.53	61.96	41.47	34.22
	DropNode	78.57	76.99	**72.93**	71.87	72.50	**70.60**	77.63	72.51	68.16	61.21	40.52	34.64
	DropEdge	78.68	70.65	44.00	71.94	69.43	45.13	**78.26**	68.39	52.08	62.06	41.03	33.61
	LADIES	78.50	**78.35**	72.71	71.88	71.69	69.80	77.65	**74.86**	**72.27**	61.49	38.96	33.17
	DropNode+Dropout	**80.60**	74.83	55.04	72.33	70.30	65.85	78.10	67.98	52.01	61.54	39.48	32.63
	DropEdge+Dropout	80.27	76.19	66.08	72.09	66.48	35.55	77.63	69.65	67.55	60.21	39.12	33.81
	LADIES+Dropout	79.81	74.72	66.62	71.85	69.24	50.81	77.46	70.54	67.94	60.27	31.41	24.86

Formulations. Random dropping [65, 26, 27, 72] is another group of approaches to address the over-smoothing issue by randomly removing or sampling a certain number of edges or nodes at each training epoch. Theoretically, it is effective in decelerating the over-smoothing and relieving the information loss caused by dimension collapse [26, 27].

As described above, $\mathbf{X}^l \in \mathbb{R}^{n \times F_l}$ is the node embeddings at the l-th layer, and F_l is the corresponding feature dimension. Let $\Re(A) = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ denotes a normalization operator, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix. Then, a vanilla GCN layer can be written as $\mathbf{X}^{l+1} = \Re(\mathbf{A})\mathbf{X}^l\mathbf{W}^l$, where $\mathbf{W}^l \in \mathbb{R}^{F_l \times F_{l+1}}$ denotes the weight matrix of the l-th layer.

$\star$ *Dropout*. [65] $\mathbf{X}^{l+1} = \Re(\mathbf{A})(\mathbf{X}^l \odot \mathbf{Z}^l)\mathbf{W}^l$, where $\mathbf{Z}^l \in \{0,1\}^{F_l \times F_{l+1}}$ is a binary random matrix, and each element $\mathbf{Z}^l_{ij} \sim \text{Bernoulli}(\pi)$ is drawn from Bernoulli distribution.

$\star$ *DropEdge*. [26] $\mathbf{X}^{l+1} = \Re(\mathbf{A} \odot \mathbf{Z}^l)\mathbf{X}^l\mathbf{W}^l$, where $\mathbf{Z}^l \in \{0,1\}^{n \times n}$ is a binary random matrix, and each element $\mathbf{Z}^l_{ij} \sim \text{Bernoulli}(\pi)$ is drawn from Bernoulli distribution.

$\star$ *DropNode*. [27] $\mathbf{X}^{l+1} = \Re(\mathbf{Q}^{l+1}\mathbf{A}\,\text{diag}(\mathbf{Z}^l)\mathbf{Q}^{l\top})\mathbf{X}^l\mathbf{W}^l$, where $\mathbf{Z}^l \in \{0,1\}^n$ is a binary random matrix, and each element $\mathbf{Z}^l_j \sim \text{Bernoulli}(\pi)$ is drawn from Bernoulli distribution. Suppose the index of selected nodes are $i_1, i_2, \cdots, i_{s_l}$, $\mathbf{Q}^l \in \{0,1\}^{s_l \times n}$ is a row selection matrix, where $\mathbf{Q}^l_{k,m} = 1$ if and only if the $m = i_k$, and $1 \le k \le s_l$, $1 \le m \le n$.

$\star$ *LADIES*. [72] $\mathbf{X}^{l+1} = \Re(\mathbf{Q}^{l+1}\mathbf{A}\,\text{diag}(\mathbf{Z}^l)\mathbf{Q}^{l\top})\mathbf{X}^l\mathbf{W}^l$, where $\mathbf{Z}^l \in \{0,1\}^n$ is a binary random matrix with $\|\mathbf{Z}^l\|_0 = s_l$, and each element $\mathbf{Z}^l_j \sim \text{Bernoulli}\left(\frac{\|\mathbf{Q}^l\mathbf{A}_{:,j}\|_2^2}{\|\mathbf{Q}^l\mathbf{A}\|_F^2}\right)$ is drawn from Bernoulli distribution. $\mathbf{Q}^l \in \{0,1\}^{s_l \times n}$ is the row selection matrix sharing the same definition in *DropNode*.

For these dropout techniques, the neuron dropout rate follows the "sweet point" configuration in Table 2, and the graph dropout rates are picked from $\{0.2, 0.5, 0.7\}$ to attain best results. All presented results adopt the layer-wise dropout scheme consistent with the one in [67].

Table 6: Average test accuracy (%) and its standard deviations from 100 independent repetitions are reported with or without the identity mapping.

Dataset	*Identity Mapping*	GCNs		
		2	16	32
Cora	with	**82.98±0.75**	**67.23±3.61**	**40.57±11.7**
	without	82.38±0.33	21.49±3.84	21.22±3.71
Citeseer	with	68.25±0.71	**56.39±4.36**	**35.28±5.85**
	without	**71.46±0.44**	19.59±1.96	20.29±1.79
PubMed	with	79.09±0.48	**79.55±0.41**	**73.74±0.71**
	without	**79.76±0.39**	39.14±1.38	38.77±1.20
OGBN-ArXiv	with	**71.08±0.28**	**69.22±0.84**	**62.85±2.22**
	without	69.46±0.22	67.96±0.38	45.48±4.50

Experimental observations. Main evaluation results of random dropping mechanisms are presented in Table 5 and A15, and additional results are referred to Table A16 and A17. Several interesting findings can be summarized as follows: (i) The overall performance of random dropping is not as significant as skip connection and graph normalization. Such random dropping tricks at most retard the performance drop, but can not boost deep GNNs to be stronger than shallow GNNs. For examples, on Cora dataset, all dropouts fail to improve deep GCNs; on OGBN-ArXiv dataset, all dropouts also fail to benefits deep SGCs. On other graph datasets, only a small subset of dropping approaches can relieve performance degradation, while others tricks still suffer accuracy deterioration. (ii) The effects of random dropping vary drastically across models, datasets, and layer number. It is hard to find a consistently improving technique. To be specific, we observe that, *Sensitive to models:* DropNode on SGC has been shown as an effective training trick for Cora dataset, but turns out to be useless when cooperating with GCN; *Sensitive to datasets:* On Pubmed dataset, the LADIES is the best trick, while it becomes the worst one on OGBN-ArXiv; *Sensitive to layer number:* Deepening GCN from 16 layers to 32 layers, the performance of DropEdge declines significantly from $65\% \sim 70\%$ to $40\% \sim 50\%$. A noteworthy exception is LADIES trick on PubMed, which consistently achieves the best result for both $16/32$ layers GCN and SGC. (iii) Dropout (with common rate setting 0.6) is demonstrated to be beneficial and stronger than node or edge dropping for shallow GCNs (i.e., 2 layers). However, this dropout rate downgrades the performance when GCN goes deeper. Combining Dropout with graph dropping may not always be helpful. For instance, incorporating dropout into LADIES downgrades the accuracy of 32 layers SGC from 69.8% to 50.8%. (iv) SGC with random dropping shows better tolerance for deep architectures, and more techniques presented in Table 5 can improve the generalization ability compared to the plain GCN. Meanwhile, with the same random dropping, SGC steadily surpasses GCN by a substantial performance margin

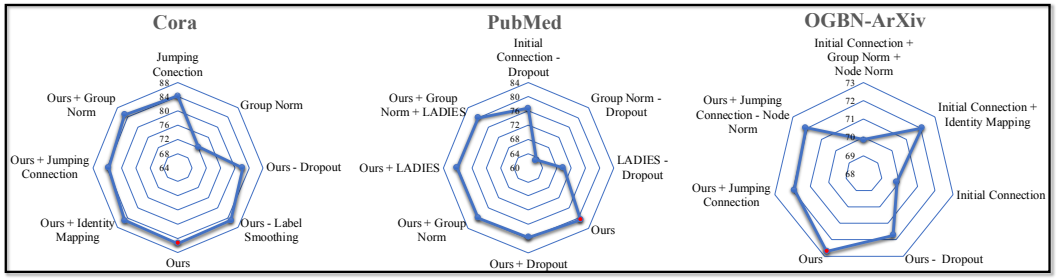

Figure 2: Average test accuracy (%) from 100 independent repetitions on Cora, PubMed, and OGBN-ArXiv graphs respectively. *Ours* indicates our best combo of tricks on the corresponding dataset. $-/+$ means add/remove certain tricks from the trick combos. The top-performing setups are highlighted with red fonts.

for training deep GCN (e.g., ≤ 16 layers) on three small graph datasets, while GCN shows a superior accuracy on large-scale OGBN graphs. (v) From the stability perspective, we notice that SGC consistently has smaller performance variance than GCN of which instability is significantly enlarged by the increasing depth.

3.5 Other Tricks

Although there are multiple other tricks in literature, such as shallow sub-graph sampling [73] and geometric scattering transform of adjacency matrix [66], most of them are intractable to be incorporated into the existing deep GNN frameworks. To be specific, the shallow sub-graph sampling has to sample the shallow neighborhood for each node iteratively and the geometric scattering requires multiscale wavelet transforms on matrix $\mathbf{A}$. Both of them do not directly propagate message based upon $\mathbf{A}$, which is deviate from the common fashion of deep GNNs. Fortunately, we find that the *Identity Mapping* proposed in [21] is effective to augment existing frameworks and relieves the over-smoothing and overfitting issues. It can be depicted as follows:

$\star$ *Identity Mapping*. [21] $\mathbf{X}^{l+1} = \sigma(\mathbf{A}\mathbf{X}^{l}(\beta_l \cdot \mathbf{W}^{l} + (1-\beta_l)\cdot \mathbf{I}))$, β_l is computed by $\beta_l = \log(\frac{\lambda}{l}+1)$, where λ is a positive hyperparameter, and β_l decreases with layers to avoid the overfitting. In our case, λ is grid searched for different graph datasets: $\lambda = 0.1$ for Cora, Citeser and PubMed, and $\lambda = 0.5$ for OGBN-ArXiv.

Experimental observations. As shown in Table 6, we conduct experiments on $2/16/32$ layers GCNs with or without the identity mapping. From the results on {Cora, Citeseer, PubMed, OGBN-ArXiv} datasets, (i) we find the identity mapping consistently brings deep GCNs (e.g., $16/32$ layers) significant accuracy improvements up to $\{45.74\%, 36.80\%, 40.41\%, 17.37\%\}$, respectively. It empirically evidenced the effectiveness of identity mapping technique in mitigating over-smoothing issue, particularly for small graphs. However, on shallow GCNs (e.g., 2 layers), identity mapping does not work very well, which either obtains a marginal gain or even incurs some degradation. (ii) Although the generalization for deep GCNs on small-scale graphs (e.g., Cora and Citeseer) is largely enhanced, it also amplify the performance instability.

4 Combining Individual Tricks into the Best Combo

The best trick combos: Provded that the "sweet point" hyperparameter configurations in Table 2 are adopted: ❶ On Cora, 64 layers SGC with *Initial Connection*, *Dropout*, and *Label Smoothing*; ❷ On PubMed, 32 layers SGC with *Jumping Connection* and *No Drpout*; ❸ On Citeseer, 16 layers GCN with *Initial Connections*, *Identity Mapping*, and *Label Smoothing*; ❹ On OGBN-ArXiv, 16 layer GCNII with *Initial Connection*, *Identity Mapping*, *GroupNorm*, *NodeNorm*, and *Dropout*.

Our best combos of tricks and ablation study. We summarize our best trick combos for Cora, Citeseer, PubMed, and OGBN-ArXiv as above. Meanwhile, comprehensive ablation studies are conducted to support the superiority of our trick combos in Figure 2. For each datasets, we examine $7 \sim 8$ combination variants by adding, removing or replacing certain tricks from our best trick combos. Results extensively endorse our carefully selected combinations.

Comparison to previous state-of-the-art frameworks. To further validate the effectiveness of our explored best combo of tricks, we perform comparisons with other previous state-of-the-art frame-

works, including SGC [69], DAGNN [25], GCNII [21], JKNet [24], APPNP [22], GPRGNN [74]. As shown in Table 7, we find that organically combining existing training tricks consistently outperforms previous elaborately design deep GCNs frameworks, on both small- and large-scale graph datasets.

Table 7: Test accuracy (%) comparison with other previous state-of-the-art frameworks. Experiments are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ GNNs. Performance are averaged from 100 repetitions, and standard deviations are reported in Table A18. The superior performance achieved by **our best tricks combo** with deep GCNs (i.e., $\geq$ 16 layers) are highlighted in the first raw.

Model	Cora (**Ours: 85.48**)			Citeseer (**Ours: 73.35**)			PubMed (**Ours: 80.76**)			ArXiv (**Ours: 72.70**)		
	2	16	32	2	16	32	2	16	32	2	16	32
SGC [69]	79.31	75.98	68.45	72.31	71.03	61.92	78.06	69.18	66.61	61.98	41.58	34.22
DAGNN [25]	80.30	84.14	83.39	18.22	73.05	72.59	77.74	80.32	80.58	67.65	71.82	71.46
GCNII [21]	82.19	84.69	85.29	67.81	72.97	73.24	78.05	80.03	79.91	71.24	72.61	72.60
JKNet [24]	79.06	72.97	73.23	66.98	54.33	50.68	77.24	64.37	63.77	63.73	66.41	66.31
APPNP [22]	82.06	83.64	83.68	71.67	72.13	72.13	79.46	80.30	80.24	65.31	66.95	66.94
GPRGNN [74]	82.53	83.69	83.13	70.49	71.39	71.01	78.73	78.78	78.46	69.31	70.30	70.18

Transferring trick combos across datasets. The last sanity check is whether there exist certain trick combos can be effective across multiple different graph datasets. Therefore, we pick *Initial Connection + GroupNorm* as the trick combos since these two techniques achieve optimal performance under most of scenarios in Section 3.2, 3.3, and 3.4, and this trick combo also performs on-par with the best trick combo on Cora and OGBN-ArXiv. Specially, we evaluate it on **eight** other open-source graph datasets: (i) two Co-author datasets [75] (CS and Physics), (ii) two Amazon datasets [75] (Computers and Photo), (iii) three WebKB datasets [76] (Texas, Wisconsin, Cornell), and (iv) the Actor dataset [76]. In these transfer investigations, we follow the exact "sweet point" settings from Cora in Table 2, except that for two Coauthor datasets, the weight decay is set to 0 and the dropout rate is set to 0.8, and in two Amazon datasets the dropout rate is set to 0.5, as adopted by [25].

Table 8: Transfer studies of our trick combos based on SGC (*Ours*). Comparison are conducted on eight hold out datasets with two widely adopted deep GNN baselines, i.e., GCNII and DAGNN. Performance are averaged from 100 independent repetitions. All trained GNNs have 32 layers.

Category	CS	Physics	Computers	Photo	Texas	Wisconsin	Cornell	Actor
SGC	70.52±3.96	91.46±0.48	37.53±0.20	26.60±4.64	56.41±4.25	51.29±6.44	58.57±3.44	26.17±1.15
DAGNN	89.60±0.71	93.31±0.60	**79.73±3.63**	89.96±1.16	57.68±5.07	50.84±6.62	58.43±3.93	27.73±1.08
GCNII	71.67±2.68	93.15±0.92	37.56±0.43	62.95±9.41	69.19±6.56	70.31±4.75	74.16±6.48	34.28±1.12
JKNet	81.82±3.32	90.92±1.61	67.99±5.07	78.42±6.95	61.08±6.23	52.76±5.69	57.30±4.95	28.80±0.97
APPNP	**91.61±0.49**	93.75±0.61	43.02±10.16	59.62±23.27	60.68±4.50	54.24±5.94	58.43±3.74	28.65±1.28
GPRGNN	89.56±0.47	93.49±0.59	41.94±9.95	**91.74±0.81**	62.27±4.97	71.35±5.56	58.27±3.96	29.88±1.82
Ours on SGC	90.74±0.66	**94.12±0.56**	75.40±1.88c	88.29±1.73	**79.68±3.77**	**83.59±3.32**	**81.24±5.77**	**36.84±0.70**

As shown in Table 8, our chosen trick combo ***universally* encourages the worst-performing SGC to become one of the top-performing candidates**. It consistently surpasses SGC, DAGNN, GCNII, JKNet, APPNP, and GPRGNN by a substantial accuracy margin, expected the comparable performance on CS, Computers, and Photo graph datasets. The superior transfer performance of our trick combo suggests it as a stronger baseline for future research in the deep GNN community.

5 Conclusion and Discussion of Broad Impact

Deep graph neural networks (GNNs) is a promising field that has so far been a bit held back by finding out "truly" effective training ticks to alleviate the notorious over-smoothing issue. This work provides a standardized benchmark with fair and consistent experimental configurations to push this field forward. We broadly investigate dozens of existing approaches on tens of representative graphs. Based on extensive experiment results, we identify the combo of most powerful training tricks, and establish new sate-of-the-art performance for deep GNNs. We hope them to lay a solid, fair, and practical evaluation foundations by providing strong baselines and superior trick combos for deep GNN research community. All our codes are provides in the supplement for reproducible research.

We do not believe that this research place any significant risk of societal harm, since it is scientific in nature. However, if deep GNNs are utilized by vicious users, our proposal may amplify the damage since it boosts the power of deep GNNs with superior training trick combinations.

References

[1] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[2] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826, 2009.

[3] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[4] Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu. Graph recurrent networks with attributed random walks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 732–740, 2019.

[5] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1416–1424, 2018.

[6] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020.

[7] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33, 2020.

[8] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10871–10880. PMLR, 13–18 Jul 2020.

[9] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

[10] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[11] Federico Monti, Michael M Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *arXiv preprint arXiv:1704.06803*, 2017.

[12] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[13] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276, 2019.

[14] Guohao Li, Matthias Müller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[15] Shunwang Gong, Mehdi Bahri, Michael M Bronstein, and Stefanos Zafeiriou. Geometrically principled connections in graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11415–11424, 2020.

[16] 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.

[17] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[20] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[21] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020.

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[23] Hongwei Zhang, Tijin Yan, Zenjun Xie, Yuanqing Xia, and Yuan Zhang. Revisiting graph convolutional network on semi-supervised node classification from an optimization perspective. *arXiv preprint arXiv:2009.11469*, 2020.

[24] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[25] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 338–348, 2020.

[26] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.

[27] Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv e-prints*, pages arXiv–2008, 2020.

[28] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[29] Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and resolving performance degradation in graph convolutional networks. *arXiv preprint arXiv:2006.07107*, 2020.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

[31] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[32] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[33] Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting "over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

[34] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems*, 33, 2020.

[35] 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.

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

[37] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, 2019.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[39] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018.

[40] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[42] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. *arXiv preprint arXiv:2106.07594*, 2021.

[43] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1695–1706. PMLR, 18–24 Jul 2021.

[44] Ting-Kuei Hu, Fernando Gama, Tianlong Chen, Zhangyang Wang, Alejandro Ribeiro, and Brian M Sadler. Vgai: End-to-end learning of vision-based decentralized controllers for robot swarms. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4900–4904. IEEE, 2021.

[45] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. *arXiv preprint arXiv:1905.06214*, 2019.

[46] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv preprint arXiv:1909.11715*, 2019.

[47] Mostafa Karimi, Di Wu, Zhangyang Wang, and Yang Shen. Explainable deep relational networks for predicting compound-protein affinities and contacts. *arXiv preprint arXiv:1912.12553*, 2019.

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

[49] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[50] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv*, 2013.

[51] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeuIPS*, pages 1024–1034, 2017.

[52] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.

[53] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[54] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[55] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[56] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673, 2019.

[57] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeuIPS*, pages 3844–3852, 2016.

[58] Martin Simonovsky and Nikos Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

[59] Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

[60] Matthew K Matlock, Arghya Datta, Na Le Dang, Kevin Jiang, and S Joshua Swamidass. Deep learning long-range information in undirected graphs with wave networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[61] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[63] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[64] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1906.02174*, 2019.

[65] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[66] Yimeng Min, Frederik Wenkel, and Guy Wolf. Scattering gcn: Overcoming oversmoothness in graph convolutional networks. *arXiv preprint arXiv:2003.08414*, 2020.

[67] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS*, 2019.

[68] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling. In *International Conference on Machine Learning*, pages 4094–4104. PMLR, 2020.

[69] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[70] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop*, 2019.

[71] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. *arXiv preprint arXiv:2105.04550*, 2021.

[72] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[73] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. *arXiv preprint arXiv:2012.01380*, 2020.

[74] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations. https://openreview. net/forum*, 2021.

[75] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[76] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

[77] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Lariviere, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research. *Journal of Machine Learning Research*, 22:1–20, 2021.

[78] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[79] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

[80] Weiwei Sun, Wei Jiang, Eduard Trulls, Andrea Tagliasacchi, and Kwang Moo Yi. Acne: Attentive context normalization for robust permutation-equivariant learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11286–11295, 2020.

[81] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[82] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[83] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816, 2009.

A1 Intended Use and Maintenance Plan

Our codes are available at: https://github.com/VITA-Group/Deep_GCN_Benchmarking.

Licensing. The license of our repository is MIT license. For more information, please see https://github.com/VITA-Group/Deep_GCN_Benchmarking/LICENSE.

Intended use. Our benchmark platform is intended for data mining scientists and machine learning researchers to innovate novel methods to tackle problems (e.g., over-smoothing, overfitting) towards deep GNNs. We implement a series of popular tricks necessary to build up deep GNNs, include a set of promising deep models, give the interfaces to access a variety of graph data, and define the standard evaluation process. Based upon our code framework, it is simple for users to incorporate promising tricks and formulate deep GNNs, which could be further tested fairly and comprehensively. Please see more in our open repository for how to use the benchmark platform.

Maintenance plan and contribution policy. Our deep GNNs benchmark is a community-driven platform. We host leaderboard in our Github to compare the outperforming models up to date, and welcome contributions from the machine learning community to pull the model incorporation request. We are committed to actively maintain the repository, enrich our model library, and update the learderboard to motivate more advanced GNNs in the future.

A2 Reproducibility Checklist

To ensure reproducibility, we use the Machine Learning Reproducibility Checklist v2.0, Apr. 7 2020 [77]. An earlier version of this checklist (v1.2) was used for NeurIPS 2019 [77].

- For all **models** and **algorithms** presented,
  - **A clear description of the mathematical settings, algorithm, and/or model.** We clearly describe all of the settings, formulations, and algorithms in Section 3.
  - **A clear explanation of any assumptions.** We do not make assumptions.
  - **An analysis of the complexity (time, space, sample size) of any algorithm.** We do not make the analysis.
- For any **theoretical claim**,
  - **A clear statement of the claim.** We do not make theoretical claims.
  - **A complete proof of the claim.** We do not make theoretical claims.
- For all **datasets** used, check if you include:
  - **The relevant statistics, such as number of examples.** We use widely adopted public graph datasets in Section 3. We provide all the related statistics in Appendix A4.
  - **The details of train/validation/test splits.** We give this information in our repository.
  - **An explanation of any data that were excluded, and all pre-processing step.** We did not exclude any data or perform any pre-processing.
  - **A link to a downloadable version of the dataset or simulation environment.** Our repository contains all of the instructions to download and run experiments on the datasets in this work. See https://github.com/VITA-Group/Deep_GCN_Benchmarking.
  - **For new data collected,a complete description of the data collection process, such as instructions to annotators and methods for quality control.** We do not collect or release new datasets.
- For all shared **code** related to this work, check if you include:
  - **Specification of dependencies.** We give installation instructions in the README of our repository.
  - **Training code.** The training code is available in our repository.
  - **Evaluation code.** The evaluation code is available in our repository.

- **(Pre-)trained model(s).** We do not release any pre-trained models. The code to run all experiments in our work can be found in the GitHub repository.
- **README file includes table of results accompanied by precise command to run to produce those results.** We include a README with detailed instructions to reproduce all of our experimental results.

- For all reported **experimental results**, check if you include:
  - **The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.** We provide all details of the hyper-parameter tuning in Section 3.
  - **The exact number of training and evaluation runs.** One hundred independent repetitions are conducted for each experiments.
  - **A clear definition of the specific measure or statistics used to report results.** We use the classification accuracy on test-set, which is defined in Section 3.
  - **A description of results with central tendency (e.g. mean) & variation (e.g. error bars).** We report mean and standard deviation for all experiments, as indicated in Section 3.
  - **The average runtime for each result, or estimated energy cost.** We do not report the running time or energy cost.
  - **A description of the computing infrastructure used.** All detailed descriptions are presented in Section A4.

A3 More Technical Details of Investigated Algorithms

Diverse hyperparameter configurations of existing deep GNNs. As shown in Table A9, A10, and A11, we can see that these highly inconsistent hyperparameter settings pose severely challenges to fairly compare the existing training tricks for deep GNNs.

Table A9: Configurations adopted to implement different approaches for training deep GNNs on Citeseer [36].

Methods	Total epoch	Learning rate & Decay	Weight decay	Dropout	Hidden dimension
Chen et al. (2020) [21]	100	0.01	5e-4	0.6	256
Xu et al. (2018) [24]	-	0.005	5e-4	0.5	{16, 32}
Klicpera et al. (2018) [22]	10000	0.01	5e-3	0.5	64
Zhang et al. (2020) [23]	1500	{0.001, 0.005, 0.01}	-	{0.1, 0.2, 0.3, 0.4, 0.5}	64
Luan et al. (2019) [64]	3000	2.8218e-03	1.9812e-02	0.98327	5000
Liu et al. (2020) [25]	100	0.01	0.005	0.8	64
Zhao et al. (2019) [28]	1500	0.005	5e-4	0.6	32
Min et al. (2020) [66]	200	0.005	0	0.9	-
Zhou et al. (2020) [34]	1000	0.005	5e-4	0.6	-
Zhou et al. (2020) [29]	50	0.005	1e-5	0	-
Rong et al. (2020) [26]	400	0.009	1e-3	0.8	128
Zou et al. (2020) [67]	100	0.001	-	-	256
Hasanzadeh et al. (2020) [68]	1700	0.005	5e-3	0	128

Table A10: Configurations adopted to implement different approaches for training deep GNNs on PubMed [36].

Methods	Total epoch	Learning rate & Decay	Weight decay	Dropout	Hidden dimension
Chen et al. (2020) [21]	100	0.01	5e-4	0.5	256
Klicpera et al. (2018) [22]	10000	0.01	5e-3	0.5	64
Zhang et al. (2020) [23]	1500	{0.001, 0.005, 0.01}	-	{0.1, 0.2, 0.3, 0.4, 0.5}	64
Luan et al. (2019) [64]	3000	0.001	0.02	0.65277	128
Liu et al. (2020) [25]	100	0.01	0.005	0.8	64
Zhao et al. (2019) [28]	1500	0.005	5e-4	0.6	{32, 64}
Zhou et al. (2020) [34]	1000	0.01	1e-3	0.6	-
Rong et al. (2020) [26]	-	0.01	1e-3	0.5	128
Zou et al. (2020) [67]	100	0.001	-	-	256
Hasanzadeh et al. (2020) [68]	2000	0.005	5e-3	0	128

A4 More Implementation Details

Datasets and computing facilities. Table A12 provides the detailed properties and download links for all adopted datasets. We adopt the following benchmark datasets since i) they are widely applied to

Table A11: Configurations adopted to implement approaches for training deep GNNs on Ogbn-ArXiv [16].

Methods	Total epoch	Learning rate & Decay	Weight decay	Dropout	Hidden dimension
Xu et al. (2018) [24]	500	0.01	-	0.5	128
Li et al. (2019, 2020) [13, 63]	500	0.01	-	0.5	128
Chen et al. (2020) [21]	1000	0.001	-	0.1	256
Zhang et al. (2020) [23]	1500	{0.001, 0.005, 0.01}	-	0.1, 0.2, 0.3, 0.4, 0.5	256
Liu et al. (2020) [25]	1000	0.005	-	0.2	256
Zhou et al. (2020) [29]	400	0.005	0.001	0.6	-
Shi et al. (2020) [78]	2000	0.001	0.0005	0.3	128
Kong et al. (2020) [79]	500	0.01	-	0.5	256
Sun et al. (2020) [80]	2000	0.002	-	0.75	256

develop and evaluate GNN models, especially for deep GNNs studied in this paper; ii) they contains 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. All experiments on large graph datasets, e.g., OGBN-ArXiv, are conducted on single 48G Quadro RTX 8000 GPU. For other experiments on small graphs such as Cora, are ran at singe 12G GTX 2080TI GPU. The detailed descriptions of these datasets are listed in the following.

- **Cora, Citeseer, Pubmed.** They are the scientific citation network datasets [81, 36], where nodes and edges represent the scientific publications and their citation relationships, respectively. Each node is described by bag-of-words representation, i.e., the 0/1-valued word vector indicating the absence/presence of the corresponding words in the scientific publication. Each node is associated with a one-hot label, where the node classification task is to predict what field the corresponding publication belongs to.

- **OGBN-Arxiv.** The OGBN-Arxiv dataset is a benchmark citation network collected in open graph benchmark (OGB) [16], which has been widely to evaluate GNN models recently. Each node represents an arXiv paper from computer science domain, and each directed edge indicates that one paper cites another one. The node is described by a 128-dimensional word embedding extracted from the title and abstract in the corresponding publication. Similar to Cora, Citeseer, and Pubmed, the node classification task is to predict the subject areas of the corresponding arXiv papers.

- **Coauthor CS, Coauthor Physics.** They are the co-authorship graph datasets [75] from the scientific fields of computer science and physics, respectively. The nodes represent the authors, and the links indicate whether the two corresponding nodes co-authored papers. Node features represent paper keywords for each author's papers. The node classification task is to predict the most active fields of study for the corresponding author.

- **Amazon Computers, Amazon Photo.** These two datasets [75] are the segments of the Amazon co-purchase graph [82]. While nodes are the items, edges indicate whether the two items re frequently bought together or not. The node features are given by bag-of-words representations extracted from the product reviews, and the class labels denote the product categories.

- **Texas, Wisconsin, Cornell.** They are the three subdatasets of WebKB[1] collected from computer science departments of various universities by Carnegie Mellon University [76]. While nodes represent webpages in the webpage datasets, edges are hyperlinks between them. The node feature vectors are given by bag-of-word representation of the corresponding webpages. Each node is associated with one-hot label to indicate one of the following five categories, i.e., student, project, course, staff, and faculty.

- **Actor.** This is an actor co-occurrence network, which is an actor-only induced subgraph of the film-director-actor-writer network [83]. In the actor co-occurrence network, nodes correspond to actors and edges denote the co-occurrence relationships on the same Wikipedia pages. Node feature vectors are described by bag-of-word representation of keywords in the actors' Wikipedia pages. The node classification task is to predict the topic of the actor's Wikipedia page.

[1] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb

Backbone Details. We implement backbones GCN [2] and SGC [3] to compare the different tricks as studied in Section 3. Specifically, we follow the official model implementations of GCN and SGC to evaluate tricks of graph normalization and random dropping. While the default dropout rate listed in Table 2 is applied for backbones (GCN/SGC+NoNorm) in benchmarking normalization techniques, the dropout rate is optimized from set $\{0.2, 0.5, 0.7\}$ for backbones (GCN/SGC+Dropout) in benchmarking dropping methods. To evaluate the skip connections and identity mapping, we further include feature transformation layers to the initial and the end of backbones, which project initial features into hidden embedding space and map hidden embeddings to node labels, respectively. The backbones are denoted by GCN/SGC+None and GCN+without in Tables 3 and 6, respectively. The slight adaptions over the backbones explain the different baseline performances for the benchmark studies of skip connections, graph normalization, random dropping, and other tricks.

Table A12: Graph datasets statistics and download links.

Dataset	Nodes	Edges	Ave. Degree	Features	Classes	Download Links
Cora	2,708	5,429	3.88	1,433	7	https://github.com/kimiyoung/planetoid/raw/master/data
Citeseer	3,327	4,732	2.84	3,703	6	https://github.com/kimiyoung/planetoid/raw/master/data
PubMed	19,717	44,338	4.50	500	3	https://github.com/kimiyoung/planetoid/raw/master/data
OGBN-ArXiv	169,343	1,166,243	13.77	128	40	https://ogb.stanford.edu/
Coauthor CS	18,333	81,894	8.93	6805	15	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/
Coauthor Physics	34,493	247,962	14.38	8,415	5	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/
Amazon Computers	13,381	245,778	36.74	767	10	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/
Amazon Photo	7,487	119,043	31.80	745	8	https://github.com/shchur/gnn-benchmark/raw/master/data/npz/
Texas	183	309	3.38	1,703	5	https://raw.githubusercontent.com/graphdml-uiuc-jlu/geom-gcn/master
Wisconsin	183	499	5.45	1,703	5	https://raw.githubusercontent.com/graphdml-uiuc-jlu/geom-gcn/master
Cornell	183	295	3.22	1,703	5	https://raw.githubusercontent.com/graphdml-uiuc-jlu/geom-gcn/master
Actor	7,600	33,544	8.83	931	5	https://raw.githubusercontent.com/graphdml-uiuc-jlu/geom-gcn/master

A5 More Experimental Results

More results of skip connections. Table A13 collects the achieved test accuracy under different skip connection mechanisms, and the standard deviations of 100 independent runs are reported.

Table A13: Test accuracy (%) under different skip connection mechanisms. Experiments with 100 independent repetitions are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC.

Backbone	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	Residual	74.73±3.41	20.05±4.66	19.57±4.94	66.83±1.37	20.77±1.25	20.90±1.34	75.27±1.72	38.84±1.39	38.74±1.33	70.19±0.17	69.34±0.30	65.09±1.08
	Initial	79.00±1.01	**78.61±0.84**	**78.74±0.86**	70.15±0.84	**68.41±0.93**	**68.36±1.10**	77.92±1.76	**77.52±1.55**	**78.18±0.49**	70.16±0.21	70.50±0.28	70.23±0.26
	Jumping	80.98±0.85	76.04±3.04	75.57±3.74	69.33±0.92	58.38±5.53	55.03±6.27	77.83±0.88	75.62±1.95	75.36±2.48	**70.24±0.23**	**71.83±0.26**	**71.87±0.23**
	Dense	77.86±1.73	69.61±4.31	67.26±6.51	66.18±1.93	49.33±7.79	41.48±7.85	72.53±2.61	69.91±6.95	62.99±10.21	70.08±0.24	71.29±0.23	70.94±0.30
	None	**82.38±0.33**	21.49±3.84	21.22±3.71	**71.46±0.44**	19.59±1.96	20.29±1.79	**79.76±0.39**	39.14±1.38	38.77±1.20	69.46±0.22	67.96±0.38	45.48±4.50
SGC	Residual	**81.77±0.28**	82.55±0.41	80.14±0.40	71.68±0.33	71.31±0.57	71.00±0.49	78.87±0.29	79.86±0.25	79.07±0.35	69.09±0.13	66.52±0.23	61.83±0.36
	Initial	81.40±0.26	**83.66±0.38**	83.77±0.38	71.60±0.33	**72.16±0.30**	**72.25±0.38**	**79.11±0.23**	79.73±0.23	79.74±0.24	68.93±0.11	69.24±0.16	69.15±0.17
	Jumping	77.75±0.65	83.42±0.50	**83.88±0.48**	69.96±0.37	71.89±0.52	71.88±0.58	77.42±0.30	**79.99±0.46**	**80.07±0.67**	68.76±0.17	70.61±0.19	70.65±0.23
	Dense	77.31±0.39	81.24±1.12	77.66±2.74	70.99±0.58	67.75±1.85	66.35±5.61	77.12±0.73	72.77±5.12	74.84±1.58	**69.39±0.18**	**71.42±0.28**	**71.52±0.31**
	None	79.31±0.37	75.98±1.06	68.45±3.10	**72.31±0.38**	71.03±1.18	61.92±3.48	78.06±0.31	69.18±0.58	66.61±0.56	61.98±0.08	41.58±0.27	34.22±0.04

More results of graph normalizations. Table A14 reports the performance with different graph normalization techniques. The standard deviations are included.

More results of random droppings. Table A15 shows the performance of diverse random dropping tricks, where all dropout rates are tuned for best test accuarcies. Table A16 and A17 report the achieved performance with fixed dropout rates 0.2 and 0.5 respectively, for better comparisons. For all experiments, the standard deviations of 100 independent repetitions are provided.

More results of comparison with previous state-of-the-art frameworks. We present a more complete comparison with other previous state-of-the-art frameworks of training deep GNNs in Table A18. Error bars are also recorded in the table.

[2] https://github.com/tkipf/gcn

[3] https://github.com/Tiiiger/SGC

Table A14: Test accuracy (%) under different graph normalization mechanisms. Experiments with 100 independent repetitions are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC.

Backbone	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	BatchNorm	69.91±1.96	61.20±13.88	**29.05±10.68**	46.27±2.30	26.25±12.11	21.82±7.91	67.15±1.94	58.00±13.42	55.98±12.98	70.44±0.22	70.52±0.46	68.74±0.84
	PairNorm	74.43±1.36	**55.75±13.19**	17.67±9.23	63.26±1.32	**27.45±7.22**	20.67±6.51	75.67±0.86	71.30±3.00	61.54±13.80	65.74±0.16	65.37±0.69	63.32±0.97
	NodeNorm	79.87±0.80	21.46±9.50	21.48±9.41	68.96±1.00	18.81±2.12	19.03±2.35	78.14±0.79	40.92±0.29	40.93±0.29	70.62±0.23	70.75±0.43	29.94±2.29
	MeanNorm	**82.49±0.35**	13.51±3.14	13.03±0.15	70.86±0.36	16.09±3.40	7.70±0.00	78.68±0.41	18.92±3.20	18.00±0.00	69.54±0.15	70.40±0.25	56.94±20.26
	GroupNorm	82.41±0.33	41.76±15.07	27.20±11.61	71.30±0.46	26.77±5.68	**25.82±5.73**	**79.78±0.39**	**70.86±5.55**	**63.91±8.26**	69.70±0.19	70.50±0.31	68.14±1.10
	CombNorm	80.00±0.86	55.64±13.23	21.44±7.52	68.59±1.03	18.90±2.17	18.53±1.68	78.11±0.76	40.93±0.29	40.90±0.28	**70.71±0.21**	**71.77±0.31**	**69.91±0.64**
	NoNorm	82.43±0.33	21.78±3.32	21.21±3.48	**71.40±0.48**	19.78±1.95	19.85±2.00	79.75±0.33	39.18±1.38	39.00±1.59	69.45±0.27	67.99±0.31	46.38±3.87
SGC	BatchNorm	79.32±0.76	15.86±8.86	14.40±7.02	61.60±1.31	17.34±4.05	17.82±3.86	76.34±0.75	54.22±15.63	29.49±11.83	**68.58±0.15**	65.54±0.38	62.33±0.61
	PairNorm	80.78±0.23	71.26±2.12	51.03±3.25	69.76±0.33	60.14±2.40	50.94±4.76	75.81±0.34	68.89±0.49	62.14±4.33	60.72±0.03	39.69±2.36	26.67±10.82
	NodeNorm	78.09±0.94	**78.77±0.96**	73.93±2.40	63.42±1.45	61.81±1.94	60.22±2.22	71.64±1.42	71.50±1.72	73.30±1.52	63.21±0.06	26.81±1.05	16.18±8.56
	MeanNorm	80.22±0.43	48.29±8.55	30.07±8.07	70.78±0.75	38.27±5.79	28.27±5.77	75.07±0.44	47.29±6.02	41.32±1.26	54.36±0.03	21.74±9.95	18.97±9.52
	GroupNorm	**82.81±0.27**	75.81±1.02	**74.94±1.87**	72.32±0.43	67.54±0.78	61.75±0.76	**78.87±0.50**	**76.43±1.03**	**74.62±1.29**	66.12±0.04	**67.29±0.53**	**66.11±0.46**
	CombNorm	77.65±1.68	75.16±2.27	74.45±1.86	63.66±1.41	59.97±3.90	54.52±4.79	71.67±2.30	71.50±2.27	72.23±2.46	65.73±0.07	54.37±0.53	47.52±1.40
	NoNorm	79.38±0.39	75.93±1.12	68.75±2.59	**72.36±0.39**	**71.06±1.32**	**62.64±3.68**	78.01±0.32	69.06±0.57	66.55±0.56	61.96±0.07	41.43±0.25	34.24±0.07

Table A15: Test accuracy (%) under different random dropping mechanisms. Experiments with 100 independent repetitions are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Dropout rates are tuned for best performance.

Dataset	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	No Dropout	80.68±0.13	**28.56±2.77**	**29.36±2.76**	71.36±0.18	23.19±1.47	**23.03±1.13**	79.56±0.17	39.85±1.47	40.00±1.59	**69.53±0.19**	66.14±0.73	41.96±9.01
	Dropout	**82.39±0.36**	21.60±3.54	21.17±3.29	**71.43±0.46**	19.37±1.88	20.15±1.85	**79.79±0.37**	39.09±1.23	39.17±1.43	69.40±0.12	67.79±0.54	45.41±3.63
	DropNode	77.10±1.04	27.61±4.34	27.65±5.07	69.38±0.89	21.83±3.07	22.18±3.06	77.39±0.98	40.31±1.61	40.38±1.20	66.67±0.16	67.17±0.44	43.81±9.62
	DropEdge	79.16±0.73	28.00±3.36	27.87±3.04	70.26±0.70	22.92±1.95	22.92±2.12	78.58±0.58	40.61±1.20	40.50±1.48	68.67±0.17	66.50±0.39	**51.70±5.08**
	LADIES	77.12±0.82	28.07±5.07	27.54±4.19	68.87±0.84	22.52±2.42	22.60±2.23	78.31±0.82	40.07±1.49	40.11±1.36	66.43±0.21	62.05±0.80	40.41±6.22
	DropNode+Dropout	81.02±0.75	22.24±8.30	18.81±4.30	70.59±0.80	24.49±6.50	18.23±0.55	78.85±0.64	40.44±1.22	40.37±1.36	68.66±0.14	68.27±0.33	44.18±4.90
	DropEdge+Dropout	79.71±0.92	20.45±7.97	21.10±8.17	69.64±0.89	19.77±4.47	18.49±1.71	77.77±0.99	40.71±0.84	40.51±0.96	66.55±0.14	**68.81±0.21**	49.82±4.16
	LADIES+Dropout	78.88±0.79	19.49±8.31	16.92±6.60	69.02±0.88	**27.17±6.74**	18.54±2.38	78.53±0.77	**41.43±2.59**	**40.70±1.00**	66.35±0.17	65.13±0.33	39.99±4.74
SGC	No Dropout	77.55±0.08	73.99±0.03	66.80±0.01	71.80±0.00	**72.69±0.05**	70.50±0.02	77.59±0.03	69.74±0.07	67.81±0.03	**62.34±0.07**	**42.54±0.25**	**34.76±0.06**
	Dropout	79.37±0.36	75.91±1.07	68.40±2.57	**72.35±0.39**	71.21±0.93	62.35±3.15	78.04±0.33	69.12±0.54	66.53±0.68	61.96±0.05	41.47±0.23	34.22±0.06
	DropNode	78.57±0.27	76.99±0.31	**72.93±0.39**	71.87±0.27	72.50±0.20	**70.60±0.11**	77.63±0.32	72.51±0.29	68.16±0.33	61.21±0.08	40.52±0.11	34.64±0.05
	DropEdge	78.68±0.24	70.65±0.80	44.00±0.90	71.94±0.32	69.43±0.57	45.13±0.93	**78.26±0.32**	68.39±0.26	52.08±0.79	62.06±0.05	41.03±0.23	33.61±0.06
	LADIES	78.50±0.27	**78.35±0.34**	72.71±0.84	71.88±0.28	71.69±0.43	69.80±0.36	77.65±0.23	**74.86±0.88**	**72.27±0.55**	61.49±0.07	38.96±0.08	33.17±0.04
	DropNode+Dropout	**80.60±0.41**	74.83±1.64	55.04±4.28	72.33±0.43	70.30±1.56	65.85±1.86	78.10±0.41	67.98±0.67	52.01±1.36	61.54±0.05	39.48±0.13	32.63±0.07
	DropEdge+Dropout	80.27±0.50	76.19±1.28	66.08±3.55	72.09±0.45	66.48±3.38	35.55±2.79	77.63±0.37	69.65±0.67	67.55±0.79	60.21±0.07	39.12±0.10	33.81±0.06
	LADIES+Dropout	79.81±0.56	74.72±1.34	66.62±2.49	71.85±0.39	69.24±1.83	50.81±4.09	77.46±0.33	70.54±0.36	67.94±0.63	60.27±0.07	31.41±0.03	24.86±1.15

Table A16: Test accuracy (%) under different random dropping mechanisms. Experiments with 100 independent repetitions are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Dropout rates are unified as $p = 0.2$.

Dataset	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	DropNode ($p = 0.2$)	77.10±1.04	27.61±4.34	27.65±5.07	69.38±0.89	21.83±3.07	22.18±3.06	77.39±0.98	40.18±1.18	39.88±2.54	66.67±0.16	67.17±0.47	43.81±9.62
	DropEdge ($p = 0.2$)	79.16±0.73	28.00±3.36	27.87±3.04	70.26±0.70	22.92±1.95	22.92±2.12	78.58±0.58	40.32±1.63	40.21±1.65	68.67±0.17	66.38±0.60	45.74±5.65
	LADIES ($p = 0.2$)	77.12±0.82	28.07±5.07	27.54±4.19	68.87±0.84	22.52±2.42	22.60±2.23	78.31±0.82	39.98±1.76	39.82±1.63	66.43±0.21	62.05±0.80	40.41±6.22
	DropNode+Dropout ($p = 0.2$)	81.02±0.75	18.55±4.68	18.81±4.30	70.59±0.80	20.03±4.68	18.16±0.79	78.85±0.64	40.22±1.60	39.92±1.73	68.66±0.14	68.19±0.25	44.18±4.90
	DropEdge+Dropout ($p = 0.2$)	79.71±0.92	17.83±4.80	19.32±5.60	69.64±0.89	18.48±1.87	18.49±1.71	77.77±0.99	39.92±1.29	39.75±1.28	66.55±0.14	68.79±0.34	49.82±4.16
	LADIES+Dropout ($p = 0.2$)	78.88±0.79	19.49±8.31	16.92±6.60	69.02±0.88	27.17±6.74	18.54±2.38	78.53±0.77	40.01±2.52	39.92±1.36	66.35±0.17	65.13±0.33	39.99±4.74
SGC	DropNode ($p = 0.2$)	77.89±0.23	75.22±0.22	69.51±0.40	71.87±0.27	72.50±0.20	70.60±0.11	77.63±0.32	70.28±0.21	68.16±0.33	61.21±0.08	40.52±0.11	34.64±0.05
	DropEdge ($p = 0.2$)	78.16±0.24	70.65±0.80	44.00±0.90	71.94±0.32	69.43±0.57	45.13±0.93	78.26±0.32	68.39±0.26	52.08±0.79	62.06±0.05	41.03±0.23	33.61±0.06
	LADIES ($p = 0.2$)	77.85±0.28	76.93±0.39	72.14±0.33	71.88±0.28	71.69±0.43	69.80±0.36	77.65±0.23	72.46±0.40	69.93±0.46	61.49±0.07	38.96±0.08	33.17±0.04
	DropNode+Dropout ($p = 0.2$)	79.95±0.49	74.83±1.64	55.04±4.28	72.33±0.43	70.30±1.56	65.85±1.86	78.10±0.41	67.98±0.67	52.01±1.36	61.54±0.05	39.48±0.13	32.63±0.07
	DropEdge+Dropout ($p = 0.2$)	80.11±0.38	76.19±1.28	66.08±3.55	72.09±0.45	66.48±3.38	35.55±2.79	77.63±0.37	69.65±0.67	67.55±0.79	60.21±0.07	39.12±0.10	33.81±0.06
	LADIES+Dropout ($p = 0.2$)	79.63±0.44	74.72±1.34	66.62±2.49	71.85±0.39	69.24±1.83	50.81±4.09	77.46±0.33	69.81±0.82	67.94±0.63	60.27±0.07	31.41±0.03	24.86±1.15

Table A17: Test accuracy (%) under different random dropping mechanisms. Experiments with 100 independent repetitions are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ layers GCN and SGC. Dropout rates are unified as $p = 0.5$.

Dataset	Settings	Cora			Citeseer			PubMed			OGBN-ArXiv		
		2	16	32	2	16	32	2	16	32	2	16	32
GCN	DropNode ($p = 0.5$)	70.28±1.36	21.09±5.86	20.62±5.92	65.47±1.31	18.68±3.05	19.38±2.84	73.82±1.38	40.31±1.61	40.38±1.20	60.64±0.16	67.17±0.44	35.65±14.39
	DropEdge ($p = 0.5$)	75.94±1.19	25.38±4.20	25.27±4.37	68.06±1.08	21.06±2.68	21.34±2.53	76.33±1.03	40.61±1.20	40.50±1.48	66.86±0.23	66.50±0.39	51.70±5.08
	LADIES ($p = 0.5$)	70.59±1.15	24.57±7.84	25.77±7.43	65.19±1.39	20.64±2.98	19.98±2.67	75.32±1.39	40.07±1.49	40.11±1.36	60.46±0.19	55.86±0.48	34.11±6.28
	DropNode+Dropout ($p = 0.5$)	78.30±1.00	22.24±6.68	18.74±6.68	68.20±1.05	24.49±6.50	18.23±0.55	76.75±1.02	40.44±1.22	40.37±1.36	66.82±0.20	68.27±0.33	42.48±4.50
	DropEdge+Dropout ($p = 0.5$)	72.31±1.21	20.45±7.97	21.10±8.17	64.69±1.35	19.77±4.47	18.22±0.90	75.21±1.04	40.71±0.84	40.51±0.96	60.17±0.14	68.81±0.21	47.03±8.06
	LADIES+Dropout ($p = 0.5$)	71.65±1.13	17.82±7.84	16.79±7.79	64.28±1.36	23.63±5.84	18.05±1.48	75.89±1.34	41.43±2.59	40.70±1.00	59.96±0.22	59.21±0.40	30.21±1.94
SGC	DropNode ($p = 0.5$)	78.57±0.27	76.99±0.31	72.93±0.39	70.84±0.32	71.59±0.20	70.54±0.28	77.38±0.22	72.51±0.29	68.08±0.39	58.29±0.08	38.80±0.06	33.92±0.03
	DropEdge ($p = 0.5$)	78.68±0.24	63.92±1.59	29.50±1.66	71.83±0.26	57.23±1.45	19.97±0.48	77.95±0.31	65.97±1.24	48.38±1.69	60.93±0.08	37.01±0.18	28.30±0.70
	LADIES ($p = 0.5$)	78.50±0.27	78.35±0.34	72.71±0.84	69.98±0.29	70.66±0.44	69.24±0.39	77.27±0.34	74.86±0.88	72.27±0.55	58.89±0.09	35.78±0.05	31.35±0.02
	DropNode+Dropout ($p = 0.5$)	80.60±0.41	68.37±2.54	8.14±0.72	72.08±0.61	63.33±3.27	48.22±2.64	77.94±0.35	64.55±3.05	44.72±0.61	59.79±0.09	34.28±0.06	27.42±0.56
	DropEdge+Dropout ($p = 0.5$)	80.27±0.50	59.52±7.94	22.44±8.83	70.86±0.46	27.48±6.40	18.12±0.30	77.37±0.47	68.67±1.43	64.27±1.78	55.73±0.07	36.29±0.05	31.66±0.03
	LADIES+Dropout ($p = 0.5$)	79.81±0.56	67.00±3.14	42.08±6.53	70.25±0.49	46.27±4.73	34.90±0.58	76.88±0.45	70.54±0.36	67.86±0.99	56.02±0.08	27.71±0.01	15.65±7.59

Table A18: Test accuracy (%) comparison with other previous state-of-the-art frameworks. Experiments are conducted on Cora, Citeseer, PubMed, and OGBN-ArXiv with $2/16/32$ GNNs. Performance are averaged from 100 repetitions. The superior performance achieved by **our best tricks combination** are highlighted in the first raw.

Model	Cora (**Ours: 85.48±0.39**)			Citeseer (**Ours: 73.35±0.79**)			PubMed (**Ours: 80.76±0.26**)			OGBN-ArXiv (**Ours: 72.70±0.15**)		
	2	16	32	2	16	32	2	16	32	2	16	32
SGC [69]	79.31±0.37	75.98±1.06	68.45±3.10	72.31±0.38	71.03±1.18	61.92±3.48	78.06±0.31	69.18±0.58	66.61±0.56	61.98±0.08	41.58±0.27	34.22±0.04
DAGNN [25]	80.30±0.78	84.14±0.59	83.39±0.59	18.22±3.48	73.05±0.62	72.59±0.54	77.74±0.57	80.32±0.38	80.58±0.51	67.65±0.52	71.82±0.28	71.46±0.27
GCNII [21]	82.19±0.77	84.69±0.51	85.29±0.47	67.81±0.89	72.97±0.71	73.24±0.78	78.05±1.53	80.03±0.50	79.91±0.27	71.24±0.17	72.61±0.29	72.60±0.25
JKNet [24]	79.06±0.11	72.97±3.94	73.23±3.59	66.98±1.82	54.33±7.74	50.68±8.73	77.24±0.92	64.37±8.80	63.77±9.21	63.73±0.38	66.41±0.56	66.31±0.63
APPNP [22]	82.06±0.46	83.64±0.48	83.68±0.48	71.67±0.78	72.13±0.53	72.13±0.59	79.46±0.47	80.30±0.30	80.24±0.33	65.31±0.23	66.95±0.24	66.94±0.26
GPRGNN [74]	82.53±0.49	83.69±0.55	83.13±0.60	70.49±0.95	71.39±0.73	71.01±0.79	78.73±0.63	78.78±1.02	78.46±1.03	69.31±0.09	70.30±0.15	70.18±0.16