Benchmarking Large-Scale Graph Training Over Effectiveness And Efficiency

Keyu Duan
k.duan@rice.edu
Rice University

Zirui Liu
zl105@rice.edu
Rice University

Wenqing Zheng
w.zheng@utexas.edu
University of Texas at Austin

Peihao Wang
peihaowang@utexas.edu
University of Texas at Austin

Kaixiong Zhou
Kaixiong.Zhou@rice.edu
Rice University

Tianlong Chen
tianlong.chen@utexas.edu
University of Texas at Austin

Zhangyang Wang
atlaswang@utexas.edu
University of Texas at Austin

Xia Hu
xia.hu@rice.edu
Rice University

ABSTRACT

Large-scale graph learning is a notoriously challenging problem in the community of network analytics and graph neural networks (GNNs). Due to the nature of evolving graph structures (a sparse matrix) into the training process, vanilla message-passing-based GNNs always failed to scale up, limited by training speed and memory occupation. Up to now, many state-of-the-art scalable GNNs have been proposed. However, we still lack a systematic study and fair benchmark of this reservoir to find the rationale for designing scalable GNNs. To this end, we conduct a meticulous and thorough study on large-scale graph learning from the perspective of *effectiveness* and *efficiency.* Firstly, we uniformly formulate the representative methods of large-scale graph training and further establish a fair and consistent benchmark regarding *effectiveness* for them by unifying the hyperparameter configuration. Secondly, benchmarking over *efficiency*, we theoretically and empirically evaluate the time and space complexity of representative paradigms for large-scale graph training. Best to our knowledge, we are the first to provide a comprehensive investigation of the efficiency of scalable GNNs, which is a key factor for the success of large-scale graph learning. Our code is available at https://github.com/VITA-Group/Large_Scale_GCN_Benchmarking.

KEYWORDS

Graph Neural Networks, Large-Scale Graph Training, Benchmark

ACM Reference Format:
Keyu Duan, Zirui Liu, Wenqing Zheng, Peihao Wang, Kaixiong Zhou, Tianlong Chen, Zhangyang Wang, and Xia Hu. 2018. Benchmarking Large-Scale Graph Training Over Effectiveness And Efficiency. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The Graph Neural Networks (GNNs) have shown great prosperity in recent years [13, 22, 35, 41], and have dominated a variety of applications, including recommender systems [14, 43], social network analysis [11, 19, 34], scientific topological structure prediction (e.g. cellular function prediction [15, 50], molecular structure prediction [16, 44], and chemical compound retrieval [36]), and scalable point cloud segmentation [25, 39], etc. However, though the message passing (MP) strategy ensures GNNs' superior performance, the nature of evolving massive topological structures prevents MP-based GNNs [10, 22, 23, 26, 35, 41, 42, 48] from scaling to industrial-grade graph applications. Specifically, as MP requires nodes aggregating information from their neighbors, the relevant graph structures inevitably need preservation during forward and backward propagation, thus occupying considerable running memory and time. For example [43], training a GNN-based recommendation system over 7.5 billion items requires three days on a 16-GPU cluster (384 GB memory in total).

To facilitate understanding, a unified formulation of MP with k layers is formulated as follows:

$$\boldsymbol{X}^{(k)} = \boldsymbol{A}^{(k-1)}\sigma\left(\boldsymbol{A}^{(k-2)}\sigma\left(\cdots\sigma(\boldsymbol{A}^{(0)}\boldsymbol{X}^{(0)}\boldsymbol{W}^{(0)})\cdots\right)\boldsymbol{W}^{(k-2)}\right)\boldsymbol{W}^{(k-1)}, \quad (1)$$

where σ is an activation function (e.g. ReLU) and $\boldsymbol{A}^{(i)}$ is the weighted adjacency matrix at i-th layer. As in Equ. 1, the key bottleneck of vanilla MP lies on the computation of $\boldsymbol{A}^{(i)}\boldsymbol{X}^{(i)}$ whose space complexity is $O(E+N)$, where E and N are the number of edges and nodes. Obviously, as the number of nodes grows, it is quite challenging for a single GPU to afford such scale of consumption.

Up to now, massive efforts have been made to mitigate the aforementioned issue of MP and scale up GNNs [2, 6, 9, 13, 33, 40, 45, 47, 51]. Most of them focus on approximating the iterative full-batch MP to reduce the memory consumption for training within a single GPU. It is worth noting that we target algorithmic scope and do not extend to general scalability topics like distributed training with multiple GPUs [1, 31] and quantization [30]. Briefly, previous works encompass two branches: *Sampling-based* and *Decoupling-based.* Namely, the former methods [2, 4, 6, 8, 13, 18, 45] perform *batch (sample)-training* that utilizes sampled adjacency matrix to approximate the full-batch MP such that the memory consumption is considerably reduced. The latter follows the principle of performing *propagation* ($A^{(k)}X^{(k)}$) and *prediction* ($X^{(k)}W^{(k)}$) separately, either precomputing the propagation [1, 9, 23, 28, 40] or post-processing with label propagation [17, 33]. Although the various branches follow different principles, they complement each other. Notably, the mixtures of these algorithmic components [7, 33, 46] have achieved the state-of-the-art (SOTA) performance on prestigious scalable graph learning benchmarks [15]. Despite the prosperity of scalable GNNs, there are still plights under-explored: consisting of a large amount of techniques, we lack a systematic study of the reservoir from the perspective of *effectiveness* and *efficiency*, without which it is unachievable to tell the rationale of the designing philosophy for large-scale graph learning in practice.

Present Work. To this end, from the perspective of *effectiveness*, we first establish a fair benchmark and provide a systematic study for large-scale graph training for both *Sampling-based* methods (§ 2.1) and *Decoupling-based* methods (§ 2.2). For each branch, we conduct a thorough investigation on the design strategy and implementation details of typical methods. Then, we carefully examine the sensitive hyperparameters and unify them in one "sweet point" set by a linear greedy search, i.e., iteratively searching the optimal value for a hyperparameter while fixing the others. For all selected methods, the hyperparameter search was performed on representative datasets of different scales, varying from about 80, 000 nodes to 2, 400, 000, including Flickr [45], Reddit [13], and ogb-products [15]. This step is a crucial precondition on our way to the ultimate as the configuration inconsistency significantly prohibits a fair comparison as well as the following analysis. Nevertheless, this burdensome work was overlooked by previous works. In addition, from the point of *efficiency* — a pivotal criterion of large-scale graph learning — we theoretically and empirically evaluate the time and space complexity of representative methods. Best to our knowledge, we are the first to provide a comprehensive benchmark of scalable GNNs regarding speed and memory usage.

2 FORMULATIONS

2.1 Sampling-based Methods

Given the formulation of Equ. 1, *sampling-based* paradigm seeks the optimal way to perform batch-training. Each batch will meet the memory constraint of a single GPU for message passing, i.e. $\widetilde{A}^{(k)}X^{(k)}$, where $\widetilde{A}$ is the adjacency matrix for the k-th layer sampled from the full graph. For clarity and completeness, we restate the unified formulation of sampling-based methods as follows:

$$X^{(k)}_{\mathcal{B}_0} = \widetilde{A}^{(k-1)}_{\mathcal{B}_1}\sigma\left(\widetilde{A}^{(k-2)}_{\mathcal{B}_2}\sigma(\cdots\sigma(\widetilde{A}^{(0)}_{\mathcal{B}_k}X^{(0)}_{\mathcal{B}_k}W^{(0)})\cdots)W^{(k-2)}\right)W^{(k-1)}, \quad (2)$$

where $\mathcal{B}_i$ is the set of sampled nodes for the i-th layer. The key difference among *sampling-based* methods is how $\{\mathcal{B}_0, \ldots, \mathcal{B}_{k-1}, \mathcal{B}_k\}$ are sampled. Given a large-scale graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, they generally encompass three paradigms:

2.1.1 *Node-wise Sampling.*

$$\mathcal{B}_{i+1} = \bigcup_{v\in\mathcal{B}_i} \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{N}(v)}\} \quad (3)$$

$\mathbb{P}$ is a sampling distribution; $\mathcal{N}(v)$ is the sampling space, i.e., the 1-hop neighbors of v; and Q denotes the number of samples. At the very beginning, $\mathcal{B}_0$ is uniformly sampled from the entire graph. Typically, $\mathbb{P}$ is implemented as the uniform distribution in GraphSAGE [13]. Generally, the node-wise sampling usually suffers from the "*Node Explosion*" problem. Namely, the number of nodes grows exponentially with layers, causing significant memory overhead. Please find detailed analysis for its time and space complexity in § 4.

2.1.2 *Layer-wise Sampling.*

$$\mathcal{B}_{i+1} = \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{N}(\mathcal{B}_i)}\} \quad (4)$$

$\mathcal{N}(\mathcal{B}_i) = \bigcup_{v\in\mathcal{B}_i}\mathcal{N}(v)$ denotes the 1-hop neighbors of all nodes in $\mathcal{B}_i$. In FastGCN [2], the sampling distribution $\mathbb{P}$ is designed regarding the node degree, where the probability for node u of being sampled is $p(u) \propto ||\hat{A}(u,:)||^2$. More recently, based on FastGCN, Zou et al. [51] propose LADIES that extends the sampling space from $\mathcal{N}(\mathcal{B}_i)$ to $\mathcal{N}(\mathcal{B}_i) \cup \mathcal{B}_i$ by adding self-loops. Notably, layer-wise sampling mitigates the "*Neighbor Explosion*" problem by fixing the sampled nodes to Q, but potentially suffers from the *linking sparsity* [6, 45] that prevents it from achieving SOTA performance.

2.1.3 *Subgraph-wise Sampling.*

$$\mathcal{B}_k = \mathcal{B}_{k-1} = \cdots = \mathcal{B}_0 = \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{G}}\} \quad (5)$$

For one epoch, all layers share the same subgraph that is derived from the entire graph $\mathcal{G}$ based on a specific sampling strategy $\mathcal{P}_{\mathcal{G}}$. The sampling strategy have two paradigms: (i) *GraphSAINT* [45] that samples a subset of nodes based on sampling distribution $\mathbb{P}$ and then induces the corresponding subgraph as a batch; (ii) *ClusterGCN* [6] that first partitions the entire graph into clusters based on the topological structure and then select several clusters to form a batch. We summarize representative sampling strategies in appendix A1.1.

2.2 Decoupling-based Methods

In conventional GNNs, message passing plays a computationally expensive and memory-consuming part. Training such GNNs on large-scale datasets with message passing for every pass is no more plausible. Therefore, we summarize another line of scalable GNNs which decouple the feature aggregation and transformation operations to avoid this operation. There are two typical ways to decouple these two operations: (i) *pre-processing* and (ii) *post-processing*.

2.2.1 *Pre-processing: MP precomputating.* Recalling Equ. 1, without loss of generalization, we assume that $A^{(k-1)} = A^{(k-2)} = \cdots A^{(0)} = A$, i.e. the topological structure for the entire graph remains the same during forward propagation, meeting most of the cases. To decouple the two operations, *message passing* (AX) and *feature transformation* (XW), we can first precompute the propagated node representations and then train a neural network for the downstream task based on these fused representations:

$$X^k = A^kX, \quad \bar{X} = \rho(X, X^1, \cdots, X^k), \quad Y = f_\theta(\bar{X}), \quad (6)$$

where X^k can be regarded as the node representation aggregating k-hop neighborhood information, K is the largest propagation hop, $\rho(\cdot)$ is a function that combines the aggregated features from different hops, $f_\theta(\cdot)$ is a feature mapping function parameterized by

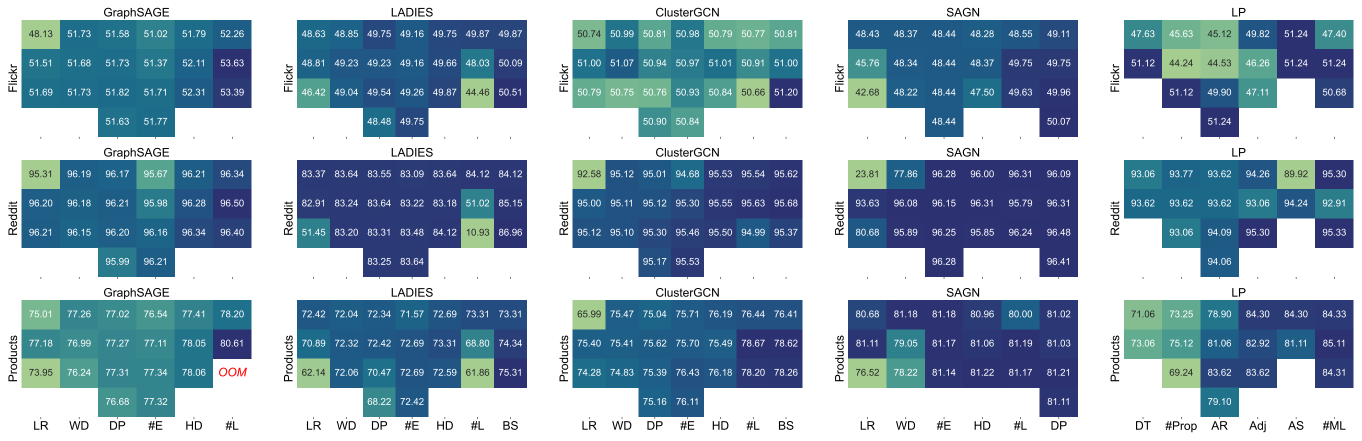

Figure 1: The greedy hyperparameter searching results for representative large-scale graph training methods.

θ. We summarize three existing pre-computing schemes [9, 33, 40] in appendix A1.2

2.2.2 Post-processing: Label Propagation. The label propagation algorithm [12, 17, 20, 29, 37, 38, 49] diffuse labels in the graph and make predictions based on the diffused labels. It is a classical family of graph algorithms for *transductive learning*, where the nodes for testing are used in the training procedure. The label propagation can be written in a unified form as follows:

$$\boldsymbol{Y}^{(k+1)} = \alpha \boldsymbol{A}\boldsymbol{Y}^{(k)} + (1-\alpha)\boldsymbol{G}. \tag{7}$$

The diffusion procedure iterates the formula above with k for multiple times. It requires two sets of inputs: 1) the stack of the *initial embedding* of all nodes, denoted as $\boldsymbol{Y}^{(0)} \in \mathbb{R}^{N\times d}$; 2) the *diffusion source embedding*, denoted as $\boldsymbol{G} \in \mathbb{R}^{N\times d}$ that propagate themselves across the edges in the graph. For the two methods in our benchmarks, Huang et al. [17] uses residual error correlation and is denoted as *residual*, and Zhu [49] set zero embeddings on test set and is denoted as *zeors*.

3 BENCHMARKING OVER EFFECTIVENESS

3.1 Implementation Details

We test numerous large-scale graph training methods with a greedy hyperparameter (HP) search to find their *sweet point* and the best performance for a fair comparison. The search space is defined in Table 1. Particularly, for label propagation, we select two representative algorithms: Huang et al. [17], the *residual* diffusion type, and Zhu [49], the *zeros* type. The number of propagation is the maximum iteration k. The aggregation ratio is α as in Equ. 7, and the number of MLP layers is the number of MLP layers that precedes the label propagation module following Huang et al. [17].

Limited by space, we select five representative approaches that covers all branches as we introduced, including GraphSAGE [13], LADIES [51], ClusterGCN [6], SAGN [33], and C&S [17]. We illustrate the selected results in Fig. 1 and place the other approaches' results in Fig. A2. For each subplot of Fig. 1 and Fig. A2, from left to right, each column denotes the searching results for one HP. Once one HP was searched, its value will be fixed to the best results for the rest HP searching. Iteratively, we obtain the best performance at the last column. For convenience and clarity, we list the searched optimal hyperparameter settings of all test methods in Table A4.

Table 1: The search space of hyperparameters for sampling based methods.

Category	Hyperparameter (Abbr.)	Candidates
Sampling & Precomputing	Learning rate (LR)	$\{1e-2^{*}, 1e-3, 1e-4\}$
	Weight Decay (WD)	$\{1e-4^{*}, 2e-4, 4e-4\}$
	Dropout Rate (DP)	$\{0.1, 0.2^{*}, 0.5, 0.7\}$
	Training Epochs[b] (#E)	$\{20, 30, 40, 50^{*}\}$
	Hidden Dimension (HD)	$128^{*}, 256, 512$
	# layers (#L)	$\{2^{*}, 4, 6\}$
	Batch size[a] (BS)	$\{1000^{*}, 2000, 5000\}$
LP	Diffusion Type (DT)	{ residual*, zeros }
	# Propagations (#Prop)	$\{2, 20^{*}, 50\}$
	Aggregation Ratio (AR)	$\{0.5, 0.75^{*}, 0.9, 0.99\}$
	Adj. Norm (Adj.)	$\{\boldsymbol{D}^{-1}\boldsymbol{A}, \boldsymbol{A}\boldsymbol{D}^{-1}, \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2*}\}$
	Auto Scale (AS)	{ *True**, *False* }
	# MLP Layers (#ML)	$\{2^{*}, 3, 4\}$

* marks the default value
[a] we do not search batch size for precomputing based methods since they do not follow a batch-training style.
[b] on ogb-products, we expand the training epoch to {500, 1000, 1500} for Precomputing-based methods to guarantee convergence.

3.2 Experimental Observations

Based the HP searching results in Fig. 1, we summarize two main experimental observations as follows. Additional detailed observation and discussion could be found in

Obs. 1. ***Sampling-based methods are more sensitive to the hyperparameters related to MP.*** According to Fig. 1, in comparison with precomputing, all sampling-based methods are non-sensitive to hyperparameters (HPs) that are related to the feature transformation matrices, including weight decay, dropout, and hidden dimension; but particularly sensitive to the MP-related HPs, including the number of layers and batch size. For model depth, sampling-based methods generally achieve the *sweet points* when the number of layers is confined to shallow ($2 \sim 4$) and suffer from the *oversmoothing* problem [5, 27, 32] as the GNN models go deeper. However, this

Table 2: The memory usage of activations and the hardware throughput (higher is better). The hardware here is a RTX 3090 GPU.

	Flickr		Reddit		ogbn-products	
	Act Mem. (MB)	Throughput (iteration/s)	Act Mem. (MB)	Throughput (iteration/s)	Act Mem. (MB)	Throughput (iteration/s)
GraphSAGE [13]	230.63	65.96	687.21	27.62	415.94	37.69
FastGCN [2]	19.77	226.93	22.53	87.94	11.54	93.05
Ladies [51]	33.26	195.34	43.21	116.46	20.33	93.47
ClusterGCN [6]	18.45	171.46	20.84	79.91	10.62	156.01
GraphSAINT [45]	16.51	151.77	21.25	70.68	10.95	143.51
SGC [40]	0.01	115.02	0.02	89.91	0.01	267.31
SIGN [9]	16.99	96.20	16.38	75.33	16.21	208.52
SAGN [33]	72.94	55.28	72.37	43.45	71.81	80.04

issue is moderately mitigated in decoupling-based methods as the model depth does not align with the number of MP hop.

Obs. 2. ***Datasets of different scales are dominated by different branches.*** As show in Fig. 1, C&S (*label propagation*) outperforms the *full-batch training* (GraphSAGE as introduced in *Obs. 2*) on the largest dataset ogb-products by a large margin of 4.5%. In contrast, GraphSAGE significantly outperforms the other methods on the smallest dataset Flickr. Remarkably, our searched results for GraphSAGE and LP on ogb-products also reached better performance, compared with the ones on the OGB leaderboard [1]. Besides, our searched results for GraphSAGE on Flickr also reach the new SOTA performance 53.63%. Noticing that GraphSAGE encounters the out-of-memory (OOM) runtime error with increasing depth, the observation partially indicates that, limited by model depth and *neighbor explosion* problem, sampling-based methods is possibly not powerful for extreme large-scale graphs to learn expressive representations.

4 BENCHMARKING OVER EFFICIENCY

In this section, we present another benchmark regarding the efficiency of scalable graph training methods. Firstly, we briefly summarize a general complexity analysis in Table 3. For *sampling-based* methods, we note that the time complexity is for training GNNs by iterating over the whole graph. The time complexity $O(L||\boldsymbol{A}||_0 D + LND^2)$ consists of two parts. The first part $L||\boldsymbol{A}||_0 D$ is from the Sparse-Dense Matrix Multiplication,i.e., $\boldsymbol{AX}$. The second part LND^2 is from the normal Dense-Dense Matrix Multiplication, i.e., $(\boldsymbol{AX})\boldsymbol{W}$. Regarding the space complexity, we need to store the activations of each layer in memory, which has a $O(bLD)$ space complexity. Note that we ignore the memory usage of model weights and the optimizer here since they are negligible compared to the activations. For *decoupling-based* methods, the training paradigm is simplified as MLPs, and thus the complexity is the same as the traditional mini-batch training. We do not include *label propagation* in our analysis since it can be trained totally on CPUs. The hyperparameter settings and other implementation details for this part are included in Appendix. A3.

[1]https://ogb.stanford.edu/docs/leader_nodeprop/

Table 3: The time and space complexity for training GNNs with sampling-based and decoupling-based methods, where b is the averaged number of nodes in the sampled subgraph and r is the averaged number of neighbors of each node. Here we do not consider the complexity of pre-processing sice it can be done in CPUs.

Category	Time Complexity	Space Complexity
Node-wise Sampling [13]	$O(r^L ND^2)$	$O(br^L D)$
Layer-wise Sampling [9, 51]	$O(rLND^2)$	$O(brLD)$
Subgraph-wise Sampling [6, 45]	$O(L\|\|\boldsymbol{A}\|\|_0 D + LND^2)$	$O(bLD)$
Precomputing [9, 33, 40]	$O(LND^2)$	$O(bLD)$

4.1 Experimental Observations

Here we report the hardware throughput and activation usage in Table 2. We summarize three main observations.

Obs. 3. ***GraphSAGE is significantly slower and occupies more memory compared to other baselines.*** This is partially because of the large neighbor sampling threshold we set and inherently owing to its neighborhood explosion. Namely, to compute the loss for a single node, it requires the neighbors' embeddings at the down-streaming layer recursively. Please refer to § 2.1.1 for details.

Obs. 4. ***counter-intuitively, SGC does not occupy any activation memory.*** As shown in Table 2, SGC only occupies about 0.01 MB actual memory during training. This is because for SGC, it only has one linear layer and the activation is exactly the input feature matrix, which has been stored in memory. Thus, it is not accounted towards the activation memory.

Obs. 5. ***In general, the speed of decoupling-based methods is comparable to sampling-based methods.*** Particularly, sampling-based methods have higher throughputs on Flickr, benefiting from batch training. However, as the scale grows, precomputing-based methods generally outperform. This is because they avoid computing MP on CPUs, thus significantly saving time.

REFERENCES

[1] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *Proceedings of*

the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2464–2473.

[2] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[3] Jianfei Chen, Lianmin Zheng, Zhewei Yao, Dequan Wang, Ion Stoica, Michael Mahoney, and Joseph Gonzalez. 2021. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*. PMLR, 1803–1813.

[4] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568* (2017).

[5] Tianlong Chen, Kaixiong Zhou, Keyu Duan, Wenqing Zheng, Peihao Wang, Xia Hu, and Zhangyang Wang. 2021. Bag of Tricks for Training Deeper Graph Neural Networks: A Comprehensive Benchmark Study. *arXiv preprint arXiv:2108.10521* (2021).

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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*. 257–266.

[7] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2021. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. *arXiv preprint arXiv:2111.00064* (2021).

[8] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1393–1403.

[9] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[10] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *international conference on machine learning*. PMLR, 2083–2092.

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

[12] Chen Gong, Dacheng Tao, Wei Liu, Liu Liu, and Jie Yang. 2016. Label propagation via teaching-to-learn and learning-to-teach. *IEEE transactions on neural networks and learning systems* 28, 6 (2016), 1452–1465.

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

[14] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

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

[16] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[17] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. 2020. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993* (2020).

[18] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

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

[20] Masayuki Karasuyama and Hiroshi Mamitsuka. 2013. Manifold-based similarity adaptation for label propagation. *Advances in neural information processing systems* 26 (2013), 1547–1555.

[21] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

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

[24] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 631–636.

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

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

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

[28] Meng Liu and Shuiwang Ji. 2022. Neighbor2Seq: Deep Learning on Massive Graphs by Transforming Neighbors to Sequences. *arXiv preprint arXiv:2202.03341* (2022).

[29] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. 2018. Learning to propagate labels: Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002* (2018).

[30] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. 2021. EXACT: Scalable Graph Neural Networks Training via Extreme Activation Compression. In *International Conference on Learning Representations*.

[31] Vasimuddin Md, Sanchit Misra, Guixiang Ma, Ramanarayan Mohanty, Evangelos Georganas, Alexander Heinecke, Dhiraj Kalamkar, Nesreen K Ahmed, and Sasikanth Avancha. 2021. DistGNN: Scalable Distributed Training for Large-Scale Graph Neural Networks. *arXiv preprint arXiv:2104.06700* (2021).

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

[33] Chuxiong Sun and Guoshi Wu. 2021. Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced training. *arXiv preprint arXiv:2104.09376* (2021).

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

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv* 1, 2 (2017).

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

[37] Fei Wang and Changshui Zhang. 2007. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20, 1 (2007), 55–67.

[38] Hongwei Wang and Jure Leskovec. 2020. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755* (2020).

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

[40] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

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

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

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

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

[45] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[46] Chenhui Zhang, Yufei He, Yukuo Cen, Zhenyu Hou, and Jie Tang. 2021. Improving the Training of Graph Neural Networks with Consistency Regularization. *arXiv preprint arXiv:2112.04319* (2021).

[47] Wentao Zhang, Ziqi Yin, Zeang Sheng, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2021. Graph attention multi-layer perceptron. *arXiv preprint arXiv:2108.10097* (2021).

[48] Kaixiong Zhou, Qingquan Song, Xiao Huang, Daochen Zha, Na Zou, and Xia Hu. 2019. Multi-Channel Graph Neural Networks. *arXiv preprint arXiv:1912.08306* (2019).

[49] Xiaojin Zhu. 2005. *Semi-supervised learning with graphs*. Carnegie Mellon University.

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

[51] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems* 32 (2019).

A1 MORE DETAILS OF FORMULATIONS

A1.1 Representative Graph Sampling Schemes

⋆ **Node Sampler** [2, 45]: $\mathbb{P}(u) = ||\widetilde{A}_{:,u}||^2$, where all nodes are sampled independently based on the normalized distribution of $\mathbb{P}$. This sampling strategy is logically equivalent to layer-wise sampling [2].
⋆ **Edge Sampler** [45]: $\mathbb{P}(u, v) = \frac{1}{deg(u)} + \frac{1}{deg(v)}$, where all edges are sampled independently based the edge distribution above. In our implementation, we utilize the sampled nodes (once contained in the sampled edges) to induce the subgraph as input, which should include more edges to help boost the performance.
⋆ **Random Walk Sampler** [24, 45]: Here, we first sample a subset of root nodes uniformly, based on which we perform a random walk at a certain length to obtain the subgraph as a batch.
⋆ **Graph Partitioner** [6, 21]: We first partition the entire graph into clusters with graph clustering algorithms and then select multiple clusters to form a batch.

A1.2 Representative Precomputing Schemes

⋆ **SGC** [40]: SGC simply keeps aggregating neighborhood information for K times and feed the resultant features to a full-connected layer. We can formulate this scheme by letting $\rho(\cdot)$ select the last element X^K and $f_\theta(\cdot)$ be a linear layer with readout activation: $Y = \sigma(X^K\Theta)$.
⋆ **SIGN** [9]: SIGN concatenates features from different hops and then fuse them as the final node representation via a linear layer. To be more specific, $\rho(\cdot)$ is defined as $\bar{X} = \begin{bmatrix} X & X^1 & \cdots & X^K \end{bmatrix} \Omega$, and $f_\theta(\cdot)$ is defined as a linear readout layer $Y = \sigma(\bar{X}\Theta)$.
⋆ **SAGN** [33]: SAGN adopts attention mechanism to combine feature representations from K hops: $\bar{X} = \sum_{k=1}^{K} \boldsymbol{T}^k \boldsymbol{X}^k$, where $\boldsymbol{T}^k$ is a diagonal matrix whose diagonal corresponds to the attention weight for each node of k-hop information. The attention weight for i-th node is calculated by $T_i^k = \text{softmax}_K(\text{LeakyReLU}(\boldsymbol{u}^T \boldsymbol{X}_i + \boldsymbol{v}^T \boldsymbol{X}_j^k))$, where the subscripts slices the data matrices along the row. The feature mapping function is selected as an MLP block with a skip connection to initial features: $Y = \text{MLP}_\theta(\bar{X} + X\Theta_r)$.

A2 ADDITIONAL EXPERIMENT RESULTS

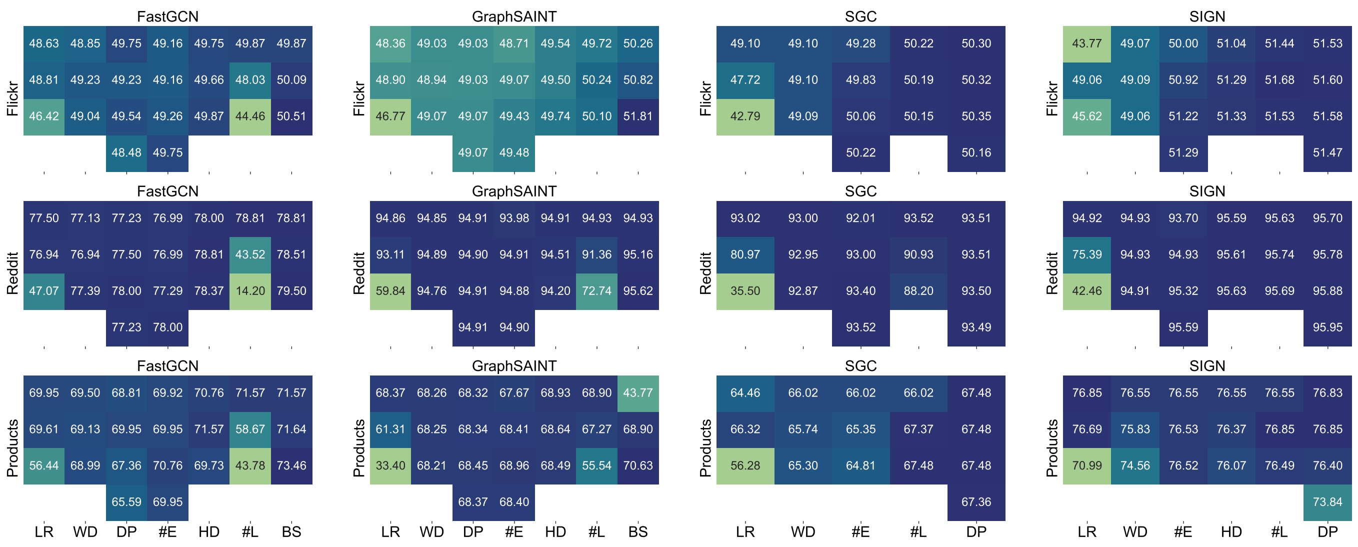

Figure A2: More greedy hyperparameter searching results for representative large-scale graph training methods, including FastGCN [2], GraphSAINT [45], SGC [40], SIGN [9].

We provide the searched optimal hyperparameters for all tested methods in Table A4 and show additional HP searching results in Fig. A2. Given Fig. 1 and Fig. A2, we provide an additional observation as follows.

Obs. 6. ***Sampling-based methods' performance is positively correlated with the training batch size.*** According to the results of the last column of all sampling-based methods, the performance of the layer-wise and subgraph-wise sampling methods is roughly proportional to the batch size. Expectedly, the model performance could further increase as the batch size grows till the upper bound of full-batch training because more links can be preserved. Particularly, in our experiment, we set the number of sampled neighbors (Q in Equ. 3) of *node-wise sampling* to a large threshold such that the performance of GraphSAGE can be regarded as *full-batch training*'s. It can be easily found that the performance of sampling-based methods is inferior to *full-batching training* (GraphSAGE), further proving our conjecture that the missing links by sampling are non-trivial.

Table A4: The searched optimal hyperparameters for all tested methods

Category	Methods	Datasets		
		Flickr	Reddit	ogbn-products
Sampling	GraphSAGE [13]	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 [2]	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 [51]	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 [6]	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 [45]	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
Decoupling	SGC [40]	LR: 0.01, WD: 0.0002, EP: 100, #L:2, DP: 0.5	LR: 0.01, WD: 0.0001, EP: 50, #L:2, DP: 0.1	LR: 0.001, WD: 0.0001, EP: 500, #L:8, DP: 0.1
	SIGN [9]	LR: 0.001, WD: 0.0002, EP: 100, HD:256, #L:4, DP: 0.2	LR: 0.01, WD: 0.0002, EP: 50, HD: 512, #L:8, DP: 0.7	LR: 0.01, WD: 0.0001, EP: 500, HD:256, #L:4, DP: 0.2
	SAGN [33]	LR: 0.01, WD: 0.0001, EP: 20, HD:64, #L:4, DP: 0.7	LR: 0.001, WD: 0.0002, EP: 50, HD: 256, #L:2, DP: 0.5	LR: 0.001, WD: 0.0, EP: 500, HD:512, #L:4, DP: 0.5
	LP [17, 49]	DT: residual, #Prop: 20, AR: 0.9, Adj: $D^{-1/2}AD^{-1/2}$, AS: True, #ML:2	DT: residual, #Prop: 50, AR: 0.9, Adj: $D^{-1}A$, AS: True, #ML:2	DT: residual, #Prop: 20, AR: 0.9, Adj: $D^{-1}A$, AS: True, #ML:3

A3 ADDITIONAL IMPLEMENTATION DETAILS

Here we provide the details of implementation and hyperparameters for the throughput and memory usage experiments. Regarding the implementation, we evaluate the hardware throughput based on Chen et al. [3]. For the activation memory, we measure it based on `torch.cuda.memory_allocated`.

Regarding the hyperparameter setting in the throughput and memory usage measurement, we set the hidden dimension to 128 across different models and datasets. We control the number of nodes whose embedding requires gradients roughly equals 5,000 across different models and datasets. Thus, our method is fair in the sense that we control the number of active nodes per batch is the same for different methods. We note that for graph-wise sampling based methods (e.g., ClusterGCN, GraphSAINT), the number of nodes whose embedding requires gradients equals the number of nodes retained in the GPU memory. However, for other sampling-based methods (e.g., GraphSAGE, FastGCN), they need to gather the neighbor embeddings to update the node embedding in current batch. These embeddings of nodes that are outside the current batch do not require gradients. We also want to clarify that the hyperparameter "batch_size" in our script has different meaning for different methods. For example, for precomputing methods, a 5,000 "batch_size" means each mini-batch contains 5,000 input samples (i.e., nodes). For GraphSAINT, "batch_size" means the number of roots in the random walk sampler. Thus, the number of nodes in each mini-batch roughly contains "batch_size" $\times$ "walk_length".