



School of
Computing



RICE UNIVERSITY
Department of Computer Science



The University of Texas at Austin
Chandra Department of Electrical
and Computer Engineering
Cockrell School of Engineering

A Comprehensive Study on Large-Scale Graph Training: Benchmarking and Rethinking

NeurIPS 2022 Dataset and Benchmark Track

Keyu Duan ¹, Zirui Liu ², Peihang Wang ³, Wenqing Zheng ³, Kaixiong Zhou ², Tianlong Chen ³, Xia Hu ², Zhangyang Wang ³

¹National University of Singapore, ²Rice University, ³University of Texas at Austin



Intro

The Bottleneck of Scaling Up GNNs

A fact: training a GNN-based recommendation system over 7.4 billion items requires three days on a 16-GPU cluster (384 GB memory in total) [9]

Recap A k -layer message passing (MP) of Vanilla GCNs:

$$\mathbf{X}^{(K)} = \mathbf{A}^{(K-1)} \sigma \left(\mathbf{A}^{(K-2)} \sigma \left(\dots \sigma \left(\mathbf{A}^{(0)} \mathbf{X}^{(0)} \mathbf{W}^{(0)} \right) \dots \right) \mathbf{W}^{(K-2)} \right) \mathbf{W}^{(K-1)},$$

Bottleneck

For the memory usage, the entire sparse adjacency matrix is supposed to be stored in one GPU. As the number of nodes grows, it is not affordable.

Formulations for Large-Scale Graph Training Paradigms

Research Question

Since the entire sparse adjacency matrix can not be stored into one GPU, how could we do the training utilizing the graph structure?

SAMPLING-BASED METHODS

Approximate batch training with sampled graphs

$$\mathbf{x}_{\mathcal{B}_0}^{(k)} = \tilde{\mathbf{A}}_{\mathcal{B}_1}^{(k-1)} \sigma \left(\tilde{\mathbf{A}}_{\mathcal{B}_2}^{(k-2)} \sigma \left(\dots \sigma \left(\tilde{\mathbf{A}}_{\mathcal{B}_K}^{(0)} \mathbf{x}_{\mathcal{B}_K}^{(0)} \mathbf{W}^{(0)} \right) \dots \right) \mathbf{W}^{(K-2)} \right) \mathbf{W}^{(K-1)},$$

Category:

- Node-wise Sampling: $\mathcal{B}_{l+1} = \bigcup_{v \in \mathcal{B}_l} \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{N}(v)}\}$
- Layer-wise Sampling: $\mathcal{B}_{l+1} = \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{N}(\mathcal{B}_l)}\}$
- Subgraph-wise Sampling: $\mathcal{B}_K = \mathcal{B}_{K-1} = \dots = \mathcal{B}_0 = \{u \mid u \sim Q \cdot \mathbb{P}_{\mathcal{G}}\}$

DECOUPLING-BASED METHODS

Move all message passing (AX operation) to CPU

- Precomputing-based:

$$\underbrace{\mathbf{X}^l = \mathbf{A}^l \mathbf{X}}_{\text{precomputing}}, \quad \underbrace{\bar{\mathbf{X}} = \rho(\mathbf{X}, \mathbf{X}^1, \dots, \mathbf{X}^K), \quad \mathbf{Y} = f_\theta(\bar{\mathbf{X}})}_{\text{end-to-end training on a GPU}}$$

- Postprocessing-based: Label Propagation

$$\mathbf{Y}^{(l)} = \alpha \mathbf{A} \mathbf{Y}^{(l-1)} + (1 - \alpha) \mathbf{G}.$$

Benchmarking Scalable GNNs

Benchmarking over Effectiveness: Greedy Hyperparameter Search

Table 1: The search space of hyperparameters for benchmarked methods.

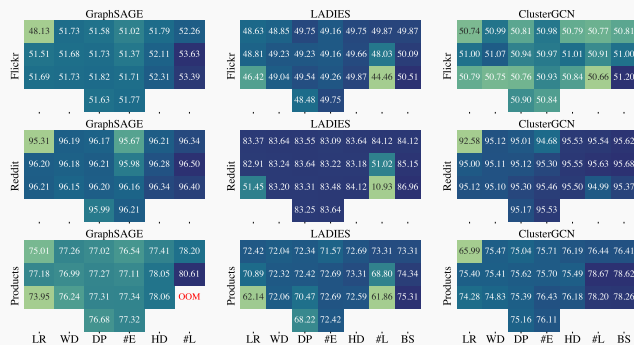
Category	Hyperparameter (Abbr.)	Candidates
Sampling & Precomputing	Learning rate (LR)	$\{1e-2^*, 1e-3, 1e-4\}$
	Weight Decay (WD)	$\{1e-4^*, 2e-4, 4e-4\}$
	Dropout Rate (DP)	$\{0.1, 0.2^*, 0.5, 0.7\}$
	Training Epochs (#E)	$\{20, 30, 40, 50^*\}$
	Hidden Dimension (HD)	$128^*, 256, 512$
	# layers (#L)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

Benchmarking over Effectiveness: Experiment Results

SAMPLING-BASED



DECOUPLING-BASED

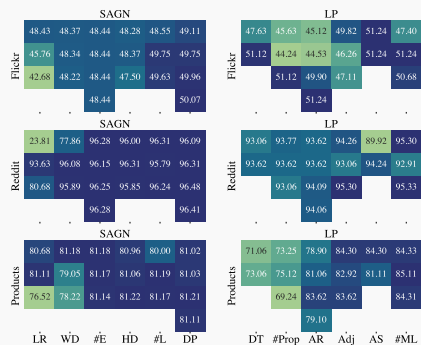


Figure 1: The greedy hyperparameter searching results for selected representative methods. The x-axis denotes the searched HPs, where the abbreviations are consistent with Table 1.

- **Sampling-based methods are more sensitive to the hyperparameters related to MP** such as `batch_size`, `num_of_layers`.
- **Sampling-based methods' performance is nearly positive-correlated with the training batch size.** Particularly, in our experiment, we set the number of sampled neighbors of *node-wise sampling* to a large threshold such that the performance of GraphSAGE can be regarded as *full-batch training*'s.
- **Precomputing-based methods generally perform better on larger datasets.** Remarkably, our searched results for GraphSAGE and LP on ogbn-products also reached better performance, compared with the ones on the OGB leaderboard ¹.

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

Benchmarking over Efficiency: Complexity Analysis

Table 2: 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 since it can be done in CPUs.

Category	Time Complexity	Space Complexity
Node-wise Sampling [4]	$\mathcal{O}(r^L ND^2)$	$\mathcal{O}(br^L D)$
Layer-wise Sampling [3, 13]	$\mathcal{O}(rLND^2)$	$\mathcal{O}(brLD)$
Subgraph-wise Sampling [2, 10]	$\mathcal{O}(L\ \mathbf{A}\ _0 D + LND^2)$	$\mathcal{O}(bLD)$
Precomputing [8, 3, 6]	$\mathcal{O}(LND^2)$	$\mathcal{O}(bLD)$

Benchmarking over Efficiency: Experiment Results

Table 3: The memory usage of activations and the hardware throughput (higher is better). The hardware here is an 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	230.63	65.96	687.21	27.62	415.94	37.69
ClusterGCN	18.45	171.46	20.84	79.91	10.62	156.01
GraphSAINT	16.51	151.77	21.25	70.68	10.95	143.51
FastGCN	19.77	226.93	22.53	87.94	11.54	93.05
LADIES	33.26	195.34	43.21	116.46	20.33	93.47
SGC	0.01	115.02	0.02	89.91	0.01	267.31
SIGN	16.99	96.20	16.38	75.33	16.21	208.52
SAGN	72.94	55.28	72.37	43.45	71.81	80.04

Benchmarking over Efficiency: Convergence Analysis

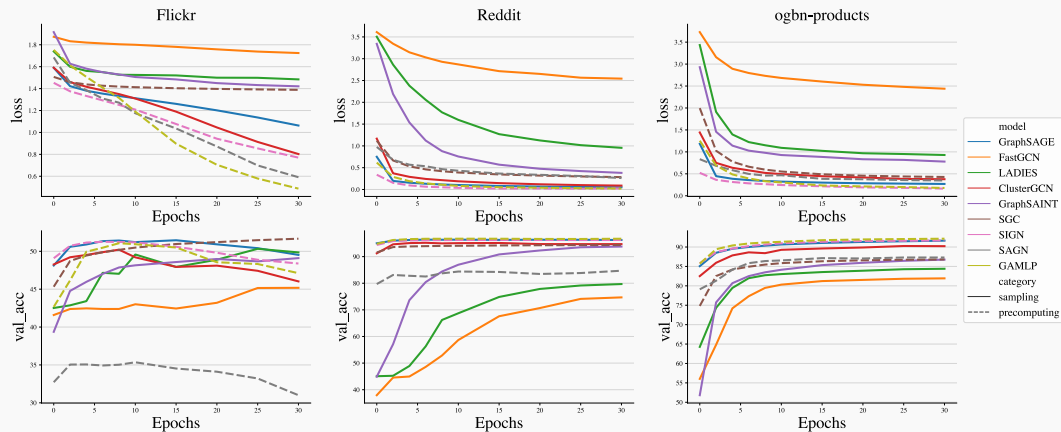


Figure 2: The empirical results of convergence for *sampling-based* methods (real line) and *precomputing-based* methods (dash line).

EnGCN: Ensembling Graph Convolutional Networks

Starting Point: Cons of Precomputing-based GNNs

- In general, precomputing-based methods at least occupy a CPU memory space of $\mathcal{O}(LNd)$, where L is the number of layers; N is the number of nodes; and d is the dimension of input features.
- In comparison, it is L times as large as the others, which is not affordable for extremely large-scale graphs.

A FACT: containing about 111 million nodes, the largest ogb dataset, ogbn-papers100M, requires approximately 57 Gigabytes (GB) to store the initial feature matrix, given the data type is float and the dimension of features is 128. As the number of layers increases, the required CPU memory space will grow proportionally to an unaffordable number.

EnGCN: Ensembling Graph Neural Networks

RECAP: The message passing of k layers with a general form

$$\mathbf{X}^{(k)} = \Phi^{(k-1)} \left(\mathbf{A} \Phi^{(k-2)} \left(\dots \mathbf{A} \Phi^{(0)} (\mathbf{A} \mathbf{X}^{(0)}) \right) \right),$$

regarding those feature transformation unit Φ as weak models, we propose a layer-wise training manner:

$$\underbrace{\mathbf{X}^{(l)} = \mathbf{A} \mathbf{X}^{(l-1)}}_{\textcircled{1} \text{ Message passing on CPUs}}, \quad \underbrace{\mathbf{Z}^{(l)} = \Phi^{(l)}(\mathbf{X}^{(l)})}_{\textcircled{2} \text{ forward propagation}}, \quad \underbrace{\nabla \Phi^{(l)} = \nabla \mathcal{L}(\mathbf{Z}^{(l)}, \mathbf{Y})}_{\textcircled{3} \text{ backward propagation}}.$$

for l from 1 to k

- $\Phi^{(l)} = \text{deepcopy}(\Phi^{(l-1)})$
- do **1**; do **2** **3** til convergence.
- save model $\Phi^{(l)}$

Organically integrating SLE [6], EnGCN achieves new SOTA performance on several benchmark datasets.

Table 4: The comparison experiment results on Flickr, Reddit, and ogbn-products

Category	Baselines	Flickr	Reddit	ogbn-products
Sampling-based	GraphSAGE [4]	53.63 \pm 0.13%	96.50 \pm 0.03%	80.61 \pm 0.16%
	FastGCN [1]	50.51 \pm 0.13%	79.50 \pm 1.22%	73.46 \pm 0.20%
	LADIES [13]	50.51 \pm 0.13%	86.96 \pm 0.37%	75.31 \pm 0.56%
	ClusterGCN [2]	51.20 \pm 0.13%	95.68 \pm 0.03%	78.62 \pm 0.61%
	GraphSAINT [10]	51.81 \pm 0.17%	95.62 \pm 0.05%	75.36 \pm 0.34%
Decoupling-based	SGC [8]	50.35 \pm 0.05%	93.51 \pm 0.04%	67.48 \pm 0.11%
	SIGN [3]	51.60 \pm 0.11%	95.95 \pm 0.02%	76.85 \pm 0.56%
	SAGN [6]	50.07 \pm 0.11%	96.48 \pm 0.03%	81.21 \pm 0.07%
	GAMLP [12]	52.58 \pm 0.12%	96.73 \pm 0.03%	83.76 \pm 0.19%
	C&S [5]	51.24 \pm 0.17%	95.33 \pm 0.08%	85.11 \pm 0.07%
Other SOTA Methods	AdaGCN [7]	52.97 \pm 0.01%	96.05 \pm 0.00%	76.41 \pm 0.00%
	SAGN+SLE [6]*	54.60 \pm 0.40%	97.10 \pm 0.00%	84.28 \pm 0.14%
	GIANT-XRT+ SAGN+MCR+C&S [11]*	-	-	86.73 \pm 0.08%
Ours	EnGCN	56.43 \pm 0.21%	97.14 \pm 0.03%	87.99 \pm 0.04%

*: the results are from the original papers

Convergence Analysis

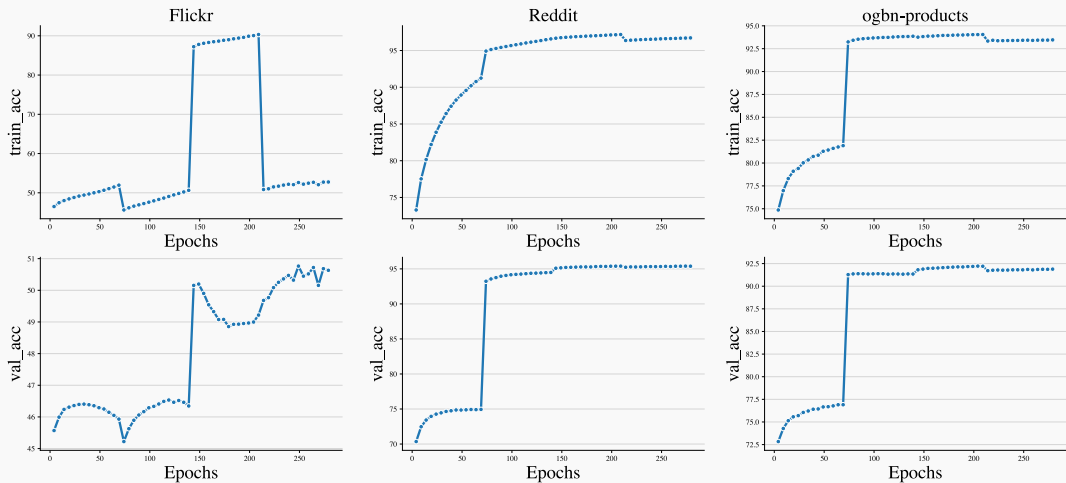


Figure 3: The convergence landscape of EnGCN. All models are trained with 4 layers' features. For each layer-wise phase, we train the model with 70 epochs.

CPU Memory Usage Comparison

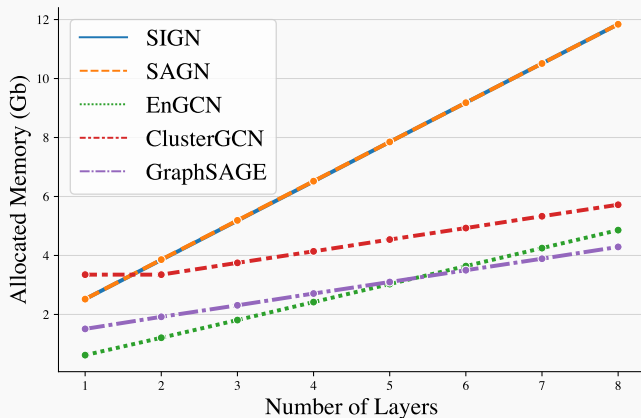












Figure 4: The allocated CPU memory of EnGCN and selected baselines on Flickr.




Questions?²

²Contact k.duan@u.nus.edu; {zl105,Kaixiong.Zhou,xia.hu}@rice.edu;
{peihaowang,w.zheng,tianlong.chen,atlaswang}@utexas.edu

-  J. Chen, T. Ma, and C. Xiao.
Fastgcn: fast learning with graph convolutional networks via importance sampling.
arXiv preprint arXiv:1801.10247, 2018.
-  W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh.
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*, pages 257–266, 2019.
-  F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti.
Sign: Scalable inception graph neural networks.
arXiv preprint arXiv:2004.11198, 2020.

-  W. Hamilton, Z. Ying, and J. Leskovec.
Inductive representation learning on large graphs.
In *NeurIPS*, pages 1024–1034, 2017.
-  Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson.
Combining label propagation and simple models out-performs graph neural networks.
arXiv preprint arXiv:2010.13993, 2020.
-  C. Sun and G. Wu.
Scalable and adaptive graph neural networks with self-label-enhanced training.
arXiv preprint arXiv:2104.09376, 2021.

-  K. Sun, Z. Zhu, and Z. Lin.
Adagcn: Adaboosting graph convolutional networks into deep models.
arXiv preprint arXiv:1908.05081, 2019.
-  F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger.
Simplifying graph convolutional networks.
In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.
-  R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. 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.
-  H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna.
Graphsaint: Graph sampling based inductive learning method.
arXiv preprint arXiv:1907.04931, 2019.

-  C. Zhang, Y. He, Y. Cen, Z. Hou, and J. Tang.
Improving the training of graph neural networks with consistency regularization.
arXiv preprint arXiv:2112.04319, 2021.
-  W. Zhang, Z. Yin, Z. Sheng, W. Ouyang, X. Li, Y. Tao, Z. Yang, and B. Cui.
Graph attention multi-layer perceptron.
arXiv preprint arXiv:2108.10097, 2021.
-  D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu.
Layer-dependent importance sampling for training deep and large graph convolutional networks.
Advances in neural information processing systems, 32, 2019.