GRAPH-LESS NEURAL NETWORKS: TEACHING OLD MLPS NEW TRICKS VIA DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have recently become popular for graph machine learning and have shown great results on wide node classification tasks. Yet, GNNs are less popular for practical deployments in the industry owing to their scalability challenges incurred by data dependency. Namely, GNN inference depends on neighbor nodes multiple hops away from the target, and fetching these nodes burdens latency-constrained applications. Existing inference acceleration methods like pruning and quantization can speed up GNNs to some extent by reducing Multiplication-and-ACcumulation (MAC) operations. However, their improvements are limited given the data dependency is not resolved. Conversely, multi-layer perceptrons (MLPs) have no dependency on graph data and infer much faster than GNNs, even though they are less accurate than GNNs for node classification in general. Motivated by these complementary strengths and weaknesses, we bring GNNs and MLPs together via knowledge distillation (KD). Our work shows that the performance of MLPs can be improved by large margins with GNN KD. We call the distilled MLPs Graph-less Neural Networks (GLNNs) as they have no inference graph dependency. We show that GLNNs with competitive performance infer faster than GNNs by 146×-273× and faster than other acceleration methods by 14×-27×. Meanwhile, under a production setting involving both transductive and inductive predictions across 7 datasets, GLNN accuracies improve over stand-alone MLPs by 12.36% on average and match GNNs on 6/7 datasets. A comprehensive analysis of GLNN shows when and why GLNNs can achieve competitive results to GNNs and suggests GLNN as a handy choice for latency-constrained applications.

1 INTRODUCTION

Graph Neural Networks (GNNs) have recently become very popular for graph machine learning (GML) research and have shown great results on node classification tasks (Kipf & Welling, 2016; Hamilton et al., 2017; Veličković et al., 2017) like product prediction on co-purchasing graphs and paper category prediction on citation graphs. However, for large-scale industrial applications, MLPs remain the major workhorse, despite common (implicit) underlying graphs and suitability for GML formalisms. One reason for this academic-industrial gap is the challenges in scalability and deployment brought by data dependency in GNNs (Zhang et al., 2020; Jia et al., 2020), which makes GNNs hard to deploy for latency-constraint applications that require fast inference.

Neighborhood fetching caused by graph dependency is one of the major sources of GNN latency. Inference on a target node necessitates fetching topology and features of many neighbor nodes, especially on small-world graphs (detailed discussion in Section 4). Common inference acceleration techniques like pruning (Zhou et al., 2021) and quantization (Tailor et al., 2021; Zhao et al., 2020) can speed up GNNs to some extent by reducing Multiplication-and-ACcumulation (MAC) operations. However, their improvements are limited given the graph dependency is not resolved. Unlike GNNs, MLPs have no dependency on graph data and are easier to deploy than GNNs. They also enjoy the auxiliary benefit of sidestepping the cold-start problem that often happens during the online prediction of relational data (Wei et al., 2020), meaning MLPs can infer reasonably even when neighbor information of a new encountered node is not immediately available. On the other hand, this lack of graph dependency typically hurts for relational learning tasks, limiting MLP performance on GML tasks compared to GNNs. We thus ask: can we bridge the two worlds, enjoy the low-latency and dependency-free nature of MLPs and the graph context awareness of GNNs at the same time?

Present work. Our key finding is that it is possible to distill knowledge from GNNs to MLPs without losing significant performance, but reducing the inference time *drastically* for node classification. The knowledge distillation (KD) can be done offline, coupled with model training. In other words, we can shift considerable work from the latency-constrained inference step, where time reduction in milliseconds makes a huge difference, to the less time-sensitive training step, where time cost in hours or days is often tolerable. We call our approach Graph-less Neural Network (GLNN). Specifically, GLNN is a modeling paradigm involving KD from a GNN teacher to a student MLP; the resulting GLNN is an MLP optimized through KD, so it enjoys the benefits of graph context-awareness in training but has no graph dependency in inference. Regarding speed, GLNNs have superior efficiency and are 146×-273× faster than GNNs and 14×-27× faster than other inference acceleration methods. Regarding performance, under a production setting involving both transductive and inductive predictions on 7 datasets, GLNN accuracies improve over MLPs by 12.36% on average and match GNNs on 6/7 datasets. We comprehensively study when and why GLNNs can achieve competitive results as GNNs. Our analysis suggests the critical factors for such great performance are large MLP sizes and high mutual information between node features and labels. Our observations align with recent results in vision and language, which posit that large enough (or slightly modified) MLPs can achieve similar results as CNNs and Transformers (Liu et al., 2021; Tolstikhin et al., 2021; Melas-Kyriazi, 2021; Touvron et al., 2021; Ding et al., 2021). Our core contributions are as follows:

- We propose GLNNs, which eliminate the neighborhood-fetching latency in GNN inference via cross-model KD to MLPs.
- We show GLNNs has competitive performance as GNNs, while enjoying 239×-758× faster inference than vanilla GNNs and 27×-32× faster inference than other inference acceleration methods.
- We study GLNN properties comprehensively by investigating their performance under different settings, how they work as regularizers, their inductive bias, expressiveness, and limitations.

2 Related Work

Graph Neural Networks. The idea of GNNs started with generalizing convolution to graphs (Bruna et al., 2014; Defferrard et al., 2017) and later simplified the to a message-passing neural network (MPNN) framework by GCN (Kipf & Welling, 2016). The majority of GNNs after are build on GCN and can be put under MPNN. For example, GAT employs attention (Veličković et al., 2017), PPNP employs personalized PageRank (Klicpera et al., 2019), GCNII and DeeperGCN employ residual connections and dense connections (Chen et al., 2020; Li et al., 2019).

Inference Acceleration. Inference acceleration schemes have been proposed by hardware improvements (Chen et al., 2016; Judd et al., 2016) and algorithmic improvements through pruning (Han et al., 2015), quantization (Gupta et al., 2015), and KD (Hinton et al., 2015). Specifically for GNNs, pruning (Zhou et al., 2021) and quantizing GNN parameters (Tailor et al., 2021; Zhao et al., 2020), or distilling to smaller GNNs (Yang et al., 2021b; Yan et al., 2020; Yang et al., 2021a) have been studied. These approaches speed up GNN inference to a certain extent but do not eliminate the neighborhood-fetching latency. In contrast, our cross-model KD approach solves this issue. Another line of research focuses on fast GNN training via sampling (Hamilton et al., 2017; Zou et al., 2019; Chen et al., 2018). They are different and complementary to our goal on inference acceleration.

3 Preliminaries

Notations. For GML tasks, the input is usually a graph and its node features, which we write as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ stands for all nodes, and $\mathcal{E}$ stands for all edges. Let N denote the total number of nodes. We use $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ to represent node features, with row $\boldsymbol{x}_v$ being the D-dimensional feature of node $v \in \mathcal{V}$. We represent edges with an adjacency matrix $\boldsymbol{A}$, with $A_{u,v} = 1$ if edge $(u, v) \in \mathcal{E}$, and 0 otherwise. For node classification, one of the most important GML applications, the prediction targets are $\boldsymbol{Y} \in \mathbb{R}^{N \times K}$, where row $\boldsymbol{y}_v$ is a K-dim one-hot vector for node v. For a given $\mathcal{G}$, usually a small portion of nodes will be labeled, which we mark using superscript ^{L}, i.e. $\mathcal{V}^L$, $\boldsymbol{X}^L$, and $\boldsymbol{Y}^L$. The majority of nodes will be unlabeled, and we mark using the superscript ^{U}, i.e. $\mathcal{V}^U$, $\boldsymbol{X}^U$, and $\boldsymbol{Y}^U$.

Graph Neural Networks. Most GNNs fit under the message-passing framework, where the representation $\boldsymbol{h}_v$ of each node v is updated iteratively in each layer by collecting messages from its neighbors denoted as $\mathcal{N}(v)$. For the l-th layer, $\boldsymbol{h}_v^{(l)}$ is obtained from the previous layer representation

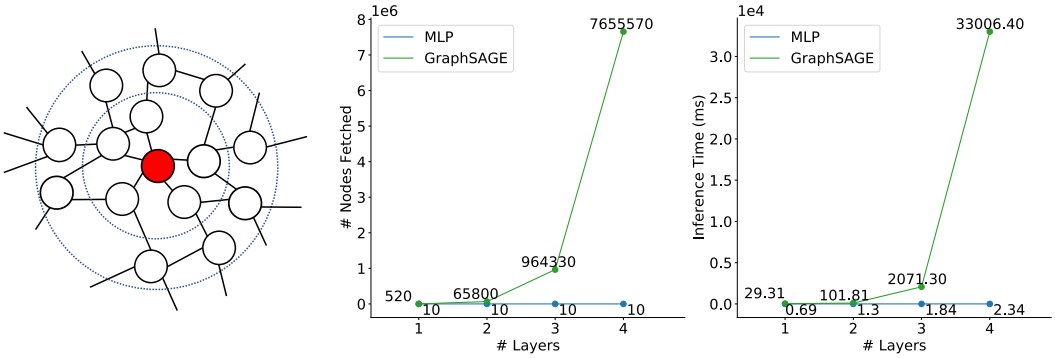

Figure 1: The number of fetches and the inference time of GNNs are both magnitudes more than MLPs and grow exponentially as functions of the number of layers. **Left**: neighbors need to be fetched for two GNN layers. **Middle**: the total number of nodes fetched during inference step. **Right**: the total inference time. Numbers are inductive inference on 10 randomly chosen nodes on the OGB `Products` dataset (Hu et al., 2020) with the DGL implementation (Wang et al., 2019).

$\boldsymbol{h}_u^{(l-1)}$ ($\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$) via an aggregation operation AGGR followed by an UPDATE operation as

$$\boldsymbol{h}_{N(v)}^{(l)} = \text{AGGR}(\{\boldsymbol{h}_u^{(l-1)} : u \in \mathcal{N}(v)\}) \qquad \text{and} \qquad \boldsymbol{h}_v^{(l)} = \text{UPDATE}(\boldsymbol{h}_{N(v)}^{(l)}, \boldsymbol{h}_v^{(l-1)})$$

4 Motivation

As mentioned in Sec 1, GNNs can have considerable inference latency due to their graph dependency. An additional GNN layer involves fetching one more hop of neighbor nodes. To infer for a single node with a L-layer GNN on a graph with average degree R will require $\mathcal{O}(R^L)$ fetchings. R can be large for real-world graphs, e.g. 208 for the Twitter social graph (Ching et al., 2015). Also, as fetching for successive layers must be done sequentially, the total latency explodes quickly as L increases. Figure 1 illustrates the successive dependence added by each GNN layer, and the exponential explosion in the number of nodes and inference time induced by multiple GNN layers. In contrast, the inference time of an MLP with the same number of layers is much smaller and only grows linearly. This marked gap contributes greatly to the practicality of MLPs in industrial applications over GNNs.

The node-fetching latency is exacerbated by two factors: firstly, newer GNN architectures are getting deeper, e.g. 64 layers (Chen et al., 2020), 112 layers (Li et al., 2019), and even 1001 layers (Li et al., 2021). Secondly, industrial-scale graphs are frequently too large to fit into the memory of a single machine, necessitating sharding of the graph out of the main memory. For example, Twitter has 288M monthly active users (nodes) and an estimated 60B followers (edges) as of 3/2015. Facebook has 1.39B active users with more than 400B edges as of 12/2014 (Ching et al., 2015). Even when stored in a sparse-matrix-friendly format (often COO or CSR), these graphs are on the order of TBs and are constantly growing. Moving away from in-memory storage results in even slower neighbor-fetching.

MLPs, on the other hand, lack the means to exploit graph topology, which hurts their performance for node classification. For example, test accuracy on `Products` is 78.61 for GraphSAGE compared to 62.47 for an equal-sized MLP. Nonetheless, recent results in vision and language posit that large (or slightly modified) MLPs can achieve similar results as CNNs and Transformers (Liu et al., 2021). We thus also ask: Can we bridge the best of GNNs and MLPs to get high-accuracy and low-latency models? This motivates us to do cross-model KD from GNNs to MLPs.

5 Graph-less Neural Networks

In this section, we introduce GLNNs in details, clarify experimental settings for our investigations, and answer the following exploration questions regarding GLNN properties: **1)** How do GLNNs compare to MLPs and GNNs? **2)** Can GLNNs work well under both transductive and inductive settings? **3)** How do GLNNs compare to other inference acceleration methods? **4)** How do GLNNs benefit from KD? **5)** Do GLNNs have sufficient model expressiveness? **6)** When will GLNNs fail to work?

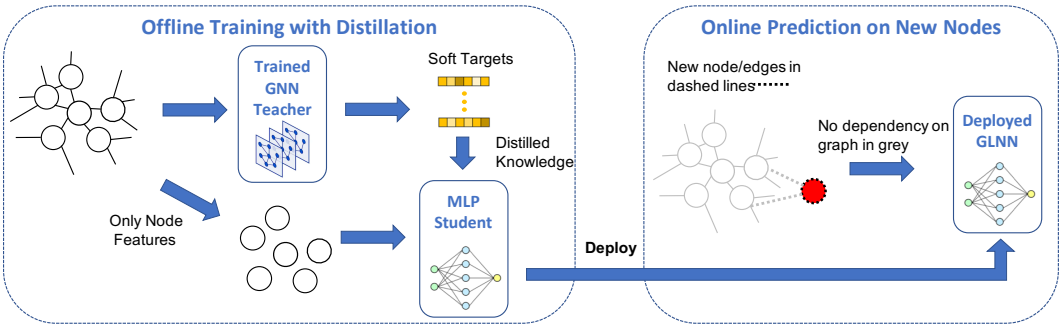

Figure 2: The GLNN framework: In offline training, a trained GNN teacher is applied on the graph for soft targets. Then, a student MLP is trained on node features guided by the soft targets. The distilled MLP, now GLNN, is deployed for online predictions. Since graph dependency is eliminated for inference, GLNNs infer much faster than GNNs, and hence the name "Graph-less Neural Network."

5.1 The GLNN Framework

The idea of GLNN is straightforward, yet as we will see, extremely effective. In short, we train a "boosted" MLP via KD from a teacher GNN. KD was introduced in Hinton et al. (2015), with the main idea being transferring knowledge from a cumbersome teacher to a simpler student. In our case, we generate soft targets $\boldsymbol{z}_v$ for each node v with a teacher GNN. Then we train the student MLP with both the true label $\boldsymbol{y}_v$ and $\boldsymbol{z}_v$. The objective is as Equation 1, with λ being a weight parameter, $\mathcal{L}_{label}$ being the cross-entropy loss between $\boldsymbol{y}_v$ and student predictions $\hat{\boldsymbol{y}}_v$, $\mathcal{L}_{teacher}$ being the KL-divergence.

$$\mathcal{L} = \lambda \Sigma_{v \in \mathcal{V}^L} \mathcal{L}_{label}(\hat{\boldsymbol{y}}_v, \boldsymbol{y}_v) + (1-\lambda) \Sigma_{v \in \mathcal{V}} \mathcal{L}_{teacher}(\hat{\boldsymbol{y}}_v, \boldsymbol{z}_v) \tag{1}$$

The model after KD, i.e. GLNN, is essentially a MLP. Therefore, GLNNs have no graph dependency during inference and are as fast as MLPs. On the other hand, through offline KD, GLNN parameters are optimized to predict and generalize as well as GNNs, with the added benefit of faster inference and easier deployment. In Figure 2, we show the offline KD and online inference steps of GLNNs.

5.2 Experiment Settings

Datasets. We consider all five datasets used in the CPF paper (Yang et al., 2021a), i.e. `Cora`, `Citeseer`, `Pubmed`, `A-computer`, and `A-photo`. To fully evaluate our method, we also include two more larger OGB datasets (Hu et al., 2020), i.e. `Arxiv` and `Products`.

Model Architectures. For consistent results, we use GraphSAGE (Hamilton et al., 2017) with GCN aggregation as the teacher. We conduct ablation studies of other GNN teachers like GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017) and, APPNP (Klicpera et al., 2019) in Section 6.

Evaluation Protocol. For all experiments in this section, we report the average and standard deviation over ten runs with different random seeds. Model performance is measured as accuracy, and results are reported on test data with the best model selected using validation data.

Transductive vs. Inductive. Given $\mathcal{G}$, $\boldsymbol{X}$, and $\boldsymbol{Y}^L$, we divide the node classification goal into two settings: transductive (*tran*) and inductive (*ind*). For the inductive setting, we hold out part of the test data for inductive evaluation only. We first select inductive nodes $\mathcal{V}^U_{ind} \subset \mathcal{V}^U$ to hold out, which partitions $\mathcal{V}^U$ into the disjoint inductive subset and observed subset, i.e. $\mathcal{V}^U = \mathcal{V}^U_{obs} \sqcup \mathcal{V}^U_{ind}$. Then we hold out all the edges connected to nodes in $\mathcal{V}^U_{ind}$ as well. Therefore, we end up with two disjoint graphs $\mathcal{G} = \mathcal{G}_{obs} \sqcup \mathcal{G}_{ind}$ with shared nodes or edges. Node features and labels are partitioned into three disjoint sets, i.e. $\boldsymbol{X} = \boldsymbol{X}^L \sqcup \boldsymbol{X}^U_{obs} \sqcup \boldsymbol{X}^U_{ind}$, and $\boldsymbol{Y} = \boldsymbol{Y}^L \sqcup \boldsymbol{Y}^U_{obs} \sqcup \boldsymbol{Y}^U_{ind}$. Concretely, the input/output of both settings becomes

- *tran*: train on $\mathcal{G}$, $\boldsymbol{X}$, and $\boldsymbol{Y}^L$; evaluate on $(\boldsymbol{X}^U, \boldsymbol{Y}^U)$; KD uses $\boldsymbol{z}_v$ for $v \in \mathcal{V}$.
- *ind*: train on $\mathcal{G}_{obs}$, $\boldsymbol{X}^L$, $\boldsymbol{X}^U_{obs}$, and $\boldsymbol{Y}^L$; evaluate on $(\boldsymbol{X}^U_{ind}, \boldsymbol{Y}^U_{ind})$; KD uses $\boldsymbol{z}_v$ for $v \in \mathcal{V}^L \sqcup \mathcal{V}^U_{obs}$.

Further details of data split, hyperparameters, and model settings are given in Appendix A.

Table 1: GLNNs outperform MLPs by large margins and match GNNs on 5 of 7 datasets under the **transductive** setting. Δ_{MLP} (Δ_{GNN}) represents difference between the GLNN and a trained MLP (GNN). Results show accuracy (higher is better); $\Delta_{GNN} \geq 0$ indicates GLNN outperforms GNN.

Datasets	SAGE	MLP	GLNN	Δ_{MLP}	Δ_{GNN}
Cora	80.52 ± 1.77	59.22 ± 1.31	**80.54 ± 1.35**	21.32 (36.00%)	0.02 (0.02%)
Citeseer	70.33 ± 1.97	59.61 ± 2.88	**71.77 ± 2.01**	12.16 (20.40%)	1.44 (2.05%)
Pubmed	75.39 ± 2.09	67.55 ± 2.31	**75.42 ± 2.31**	7.87 (11.65%)	0.03 (0.04%)
A-computer	82.97 ± 2.16	67.80 ± 1.06	**83.03 ± 1.87**	15.23 (22.46%)	0.06 (0.07%)
A-photo	90.90 ± 0.84	78.77 ± 1.74	**92.11 ± 1.08**	13.34 (16.94%)	1.21 (1.33%)
Arxiv	**70.92 ± 0.17**	56.05 ± 0.46	63.46 ± 0.45	7.41 (13.24%)	-7.46 (-10.52%)
Products	**78.61 ± 0.49**	62.47 ± 0.10	68.86 ± 0.46	6.39 (10.23%)	-9.75 (-12.4%)

Table 2: Enlarged GLNNs match the performance of GNNs on the OGB datasets. For Arxiv, we use MLPw4 (GLNNw4). For Products, we use MLPw8 (GLNNw8).

Datasets	SAGE	MLP+	GLNN+	Δ_{MLP}	Δ_{GNN}
Arxiv	70.92 ± 0.17	55.31 ± 0.47	**72.15 ± 0.27**	16.85 (30.46%)	0.51 (0.71%)
Products	**78.61 ± 0.49**	64.50 ± 0.45	77.65 ± 0.48	13.14 (20.38%)	-0.97 (-1.23%)

5.3 How do GLNNs compare to MLPs and GNNs?

We start by comparing GLNNs to equal-sized MLPs and GNNs. Specifically, the same number of layers and hidden dimensions. We first study the transductive setting, i.e. the standard setting for all used datasets in past works, so our experiment results in Table 1 can be directly compared to the results reported in previous literature like Yang et al. (2021a) and Hu et al. (2020).

As shown in Table 1, the performance of all GLNNs improve over MLPs by large margins. On smaller datasets (first 5 rows), GLNNs can even outperform the teacher GNNs. In other words, for each of these tasks, with the same parameter budget, there exists a set of MLP parameters that has GNN-competitive performance. We investigate the rationale in Sections 5.6 and 5.7. For the larger OGB datasets (last 2 rows), the GLNN performance is improved over MLPs but still worse than the teacher GNNs. However, as we show in Table 2, this gap can be mitigated by increasing MLP size to MLPwi[1]. In Figure 3 (right), we visualize the trade-off between prediction accuracy and model inference time with different model sizes. We show that gradually increasing GLNN size pushes its performance to be close to SAGE. On the other hand, when we reduce the number of layers of SAGE[2], the accuracy quickly drops to be worse than GLNNs. A detailed discussion of the rationale for increasing MLP sizes is in Appendix B.

5.4 Can GLNNs work well under both transductive and inductive settings?

Although transductive is the commonly studied setting for node classification, it does not encompass prediction on unseen nodes. Therefore, it may not be the best way to evaluate a deployed model, which must often generate predictions for new data points as well as reliably maintain performance on old ones. Thus, to better understand the effectiveness of GLNN, we also consider their performance under a realistic production setting, which contains both transductive and inductive predictions.

To evaluate a model inductively, we hold out some unlabeled test nodes from training to form an inductive subset, i.e. $\mathcal{V}^U = \mathcal{V}^U_{obs} \sqcup \mathcal{V}^U_{ind}$. In production, a model might be re-trained periodically, e.g. weekly. The hold-out nodes in V^U_{ind} represent new nodes that entered the graph between two times of training. V^U_{ind} is usually small compared to the size of V^U_{obs} – e.g. Graham (2012) estimates 5-7% for the fastest-growing tech startups. In our case, to mitigate randomness and better evaluate generalizability, we consider a larger V^U_{ind} containing 20% of the test data. We also evaluate on V^U_{obs} containing the other 80% of the test data, representing the standard transductive prediction on existing unlabeled nodes, since inference is commonly redone on existing nodes in real-world cases.

[1]Suffix -wi is used to note hidden layers are enlarged i times, e.g. MLPw4 has 4-times wider hidden layers than the MLP given in the context.

[2]Suffix -Li is used to explicitly note a model with i layers, e.g. SAGE-L2 represents a 2-layer SAGE.

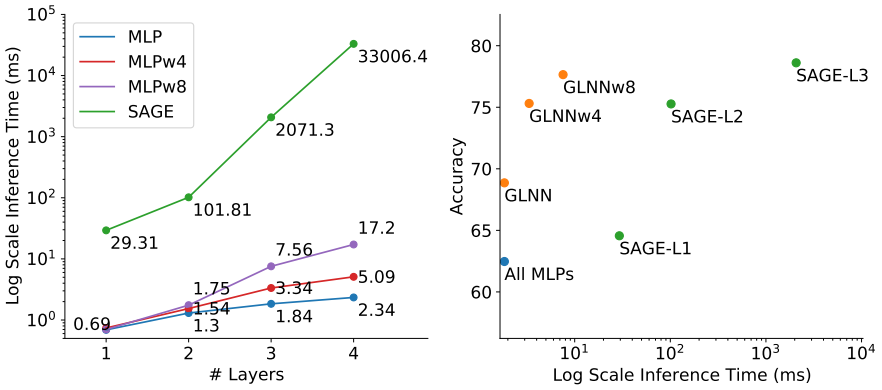

Figure 3: Enlarged MLPs (GLNNs) can match GNN accuracy, but with dramatically faster inference. Both plots are under the same setting as Figure 1. **Left** shows the inference time of MLPs vs. GNN (SAGE) for different model sizes. **Right** shows the model accuracy vs. inference time. Note: time axes are log-scale, since GNNs are much slower than GLNNs.

Table 3: GLNNs match GNN performance on a production setting with both **inductive** and **transductive** predictions. We use MLP for the 5 CPF datasets, MLPw4 for `Arxiv`, and MLPw8 for `Products`. *ind* results on V_{ind}^{U}, *tran* results on V_{obs}^{U}, and the interpolated *prod* results are reported.

Datasets	Eval	SAGE	MLP/MLP+	GLNN/GLNN+	Δ_{MLP}	Δ_{GNN}
`Cora`	*prod*	**79.29**	58.98	78.28	19.30 (32.72%)	-1.01 (-1.28%)
	ind	81.33 ± 2.19	59.09 ± 2.96	73.82 ± 1.93		
	tran	78.78 ± 1.92	58.95 ± 1.66	79.39 ± 1.64		
`Citeseer`	*prod*	68.38	59.81	**69.27**	9.46 (15.82%)	0.89 (1.30%)
	ind	69.75 ± 3.59	60.06 ± 5.00	69.25 ± 2.25		
	tran	68.04 ± 3.34	59.75 ± 2.48	69.28 ± 3.12		
`Pubmed`	*prod*	**74.88**	66.80	74.71	7.91 (11.83%)	-0.17 (-0.22%)
	ind	75.26 ± 2.57	66.85 ± 2.96	74.30 ± 2.61		
	tran	74.78 ± 2.22	66.79 ± 2.90	74.81 ± 2.39		
`A-computer`	*prod*	82.14	67.38	**82.29**	14.90 (22.12%)	0.15 (0.19%)
	ind	82.08 ± 1.79	67.84 ± 1.78	80.92 ± 1.36		
	tran	82.15 ± 1.55	67.27 ± 1.36	82.63 ± 1.40		
`A-photo`	*prod*	91.08	79.25	**92.38**	13.13 (16.57%)	1.30 (1.42%)
	ind	91.50 ± 0.79	79.44 ± 1.72	91.18 ± 0.81		
	tran	90.80 ± 0.77	79.20 ± 1.64	92.68 ± 0.56		
`Arxiv`	*prod*	**70.73**	55.30	65.09	9.79 (17.70%)	-5.64 (-7.97%)
	ind	70.64 ± 0.67	55.40 ± 0.56	60.48 ± 0.46		
	tran	70.75 ± 0.27	55.28 ± 0.49	71.46 ± 0.33		
`Products`	*prod*	**76.60**	63.72	75.77	12.05 (18.91%)	-0.83 (-1.09%)
	ind	76.89 ± 0.53	63.70 ± 0.66	75.16 ± 0.34		
	tran	76.53 ±0.55	63.73 ± 0.69	75.92 ± 0.61		

We report both results and a interpolated production (*prod*) results per dataset in Table 3. The *prod* results paint a clearer picture of model generalization as well as accuracy in production settings. We also do an ablation study of different inductive split ratios other than 20-80 in Section 6.

In Table 3, we see that GLNNs can still improve over MLP by large margins for inductive predictions. On 6/7 datasets, the GLNN *prod* performance are competitive to GNNs, which supports deploying GLNN as a much faster model with no or only slight performance loss. On the `Arxiv` dataset, the GLNN performance is notably less than GNNs – we hypothesize this is due to `Arxiv` having a particularly challenging data split which causes distribution shift between test nodes and training nodes, which is hard for GLNNs to capture without utilizing neighbor information like GNNs. However, we note that GLNN performance is substantially improved over MLP.

Table 4: Common inference acceleration methods speed up SAGE, but still considerably slower than GLNNs. Numbers (in *ms*) are inductive inference on 10 randomly chosen nodes.

Datasets	SAGE	QSAGE	PSAGE	Neighbor Sample	GLNN+
`Arxiv`	489.49	433.90 (1.13×)	465.43 (1.05×)	91.03 (5.37×)	**3.34 (146.55×)**
`Products`	2071.30	1946.49 (1.06×)	2001.46 (1.04×)	107.71 (19.23×)	**7.56 (273.98×)**

5.5 How do GLNNs compare to other inference acceleration methods?

Common techniques of inference acceleration include pruning and quantization. These approaches can reduce model parameters and Multiplication-and-ACcumulation (MACs) operations. Still, they don't eliminate neighborhood-fetching latency. Therefore, their speed gain on GNNs is less significant than on NNs. For GNNs specifically, we can also perform neighbor sampling to reduce the fetching latency. We show an explicit speed comparison between vanilla SAGE, quantized SAGE from FP32 to INT8 (QSAGE), SAGE with 50% weights pruned (PSAGE), inference neighbor sampling with fan-out 15, and GLNN in Table 4. All numbers are on `Products`, with setting the same as Figure 1. We see that GLNN is considerably faster.

Two other kinds of methods considered as inference acceleration are GNN-to-GNN KD like TinyGNN (Yan et al., 2020) and Graph Augmented-MLPs (GA-MLPs) like SGC (Wu et al., 2019) or SIGN (Frasca et al., 2020). Inference of GNN-to-GNN KD is likely to be slower than a GNN-Li with the same i as the student, since there will usually be overheads introduced by some extra modules like the Peer-Aware Module (PAM) in TinyGNN. GA-MLPs precompute augmented node features and apply MLPs to them. With precomputation, their inference time will be the same as MLPs for dimension-preserving augmentation (SGC) and the same as enlarged MLPwi for augmentation involves concatenation (SIGN). Thus, for both kinds of approaches, it is sufficient to compare GLNN with GNN-Li and MLPwi, which we have already shown in Figure 3 (left). We see that GNN-Lis are much slower than MLPs. For GA-MLPs, since full pre-computation cannot be done for inductive nodes, GA-MLPs still need to fetch neighbor nodes. This makes them much slower than MLPwi in the inductive setting, and even slower than pruned GNNs and TinyGNN as shown in Zhou et al. (2021).

5.6 How does GLNN benefit from distillation?

We showed that GNNs are markedly better than MLPs on node classification tasks. But, with KD, GLNNs can often become competitive to GNNs. This indicates that there exist suitable MLP parameters which can well approximate the ideal prediction function from node features to labels. However, these parameters can be difficult to learn through standard stochastic gradient descent. We hypothesize that KD helps to find them through regularization and transfer of inductive bias.

First, we show that KD can help to regularize the student model and prevent overfitting. By comparing the loss curves of a directly trained MLP and the GLNN in Figure 4, the gap between training and validation loss is visibly larger for MLPs than GLNNs, and MLPs show obvious overfitting trends. Second, we analyze the inductive bias that making GNNs powerful on node classification, which suggests that node inferences should be influenced by the graph topology, especially by neighbor nodes. In contrast, MLPs are more flexible but have less inductive bias. Similar difference exists between Transformers (Vaswani et al., 2017) and MLPs. In Liu et al. (2021), it is shown that the inductive bias in Transformers can be mitigated by a simple gate on large MLPs. For node classification, we hypothesize that KD helps to mitigate the inductive bias, so GLNNs can perform competitively. Soft labels from GNN teachers are heavily influenced by the graph topology due to inductive bias. They maintain nonzero probabilities on classes other than the ground truth provided by one-hot labels, which can be useful for the student to learn to complement the missing inductive bias in MLPs. To evaluate this hypothesis quantitatively, we define the cut loss $\mathcal{L}_{cut} \in [0, 1]$ in Equation 2 to measure the consistency between model predictions and graph topology (details in Appendix C):

$$\mathcal{L}_{cut} = \frac{Tr(\hat{\boldsymbol{Y}}^T \boldsymbol{A} \hat{\boldsymbol{Y}})}{Tr(\hat{\boldsymbol{Y}}^T \boldsymbol{D} \hat{\boldsymbol{Y}})} \tag{2}$$

Here $\hat{\boldsymbol{Y}} \in [0, 1]^{N \times K}$ is the soft classification probability output by the model, $\boldsymbol{A}$ and $\boldsymbol{D}$ are the adjacency and degree matrices. When $\mathcal{L}_{cut}$ is close to 1, it means the predictions and the graph

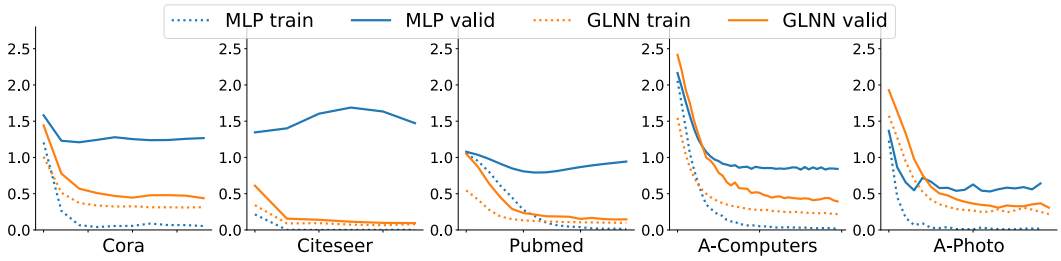

Figure 4: Loss curves on CPF datasets show GLNN distillation can help to regularize the training.

topology are very consistent. In our experiment, we observe that the average $\mathcal{L}_{cut}$ for SAGE over five CPF datasets is 0.9221, which means high consistency. The same $\mathcal{L}_{cut}$ for MLPs is only 0.7644, but for GLNNs it is 0.8986. This shows that the GLNN predictions indeed benefit from the graph topology knowledge contained in the teacher outputs (the full table of $\mathcal{L}_{cut}$ values in Appendix C).

5.7 Do GLNNs have enough model expressiveness?

Intuitively, the addition of neighbor information makes GNNs more powerful than MLPs when classifying nodes. Thus, a natural question regarding KD from GNNs to MLPs is whether MLPs are expressive enough to represent graph data as well as GNNs. Many recent works studied GNN model expressiveness (Xu et al., 2018; Chen et al., 2021). The latter analyzed GNNs and GA-MLPs for node classification and characterized expressiveness as the number of equivalence classes of rooted graphs induced by the model (formal definitions in Appendix D). The conclusion is that GNNs are more powerful than GA-MLPs, but in most real-world cases their expressiveness is indistinguishable.

We adopt the analysis framework from Chen et al. (2021) and show in Appendix D that the number of equivalence classes induced by GNNs and MLPs are $\binom{|\mathcal{X}|+m-2}{m-1}^{2^L-1}$ and $|\mathcal{X}|$ respectively. Here m denotes the max node degree, L denotes the number of GNN layers, and $\mathcal{X}$ denotes the set of all possible node features. The former is apparently larger which concludes that GNNs are more expressive. Empirically, however, the gap makes little difference when $|\mathcal{X}|$ is large. In real applications, node features can be high dimensional like bag-of-words, or even word embeddings, thus making $|\mathcal{X}|$ enormous. Like for bag-of-words, $|\mathcal{X}|$ is in the order of $\mathcal{O}(p^D)$, where D is the vocabulary size, and p is the max word frequency. The expressiveness of a L-layer GNN is lower bounded by $\binom{|\mathcal{X}|+m-2}{m-1}^{2^L-1} = \mathcal{O}(p^{D(m-1)(2^L-1)})$, but empirically, both MLPs and GNNs should have enough expressiveness given D is usually hundreds or bigger (see Table 5).

5.8 When will GLNNs fail to work?

As discussed in Section 5.7 and Appendix D, the goal of GML node classification is to fit a function f on the rooted graph $\mathcal{G}^{[i]}$ and label $\boldsymbol{y}_i$. From the information theoretic perspective, fitting f by minimizing the commonly used cross-entropy loss is equivalent to maximizing the mutual information (MI), $I(\mathcal{G}^{[i]}; \boldsymbol{y}_i)$ as shown in Qin et al. (2020). If we consider $\mathcal{G}^{[i]}$ as a joint distribution of two random variables $\boldsymbol{X}^{[i]}$ and $\mathcal{E}^{[i]}$ representing the node features and edges in $\mathcal{G}^{[i]}$ respectively, we have

$$I(\mathcal{G}^{[i]}; \boldsymbol{y}_i) = I(\boldsymbol{X}^{[i]}, \mathcal{E}^{[i]}; \boldsymbol{y}_i) = I(\mathcal{E}^{[i]}; \boldsymbol{y}_i) + I(\boldsymbol{X}^{[i]}; \boldsymbol{y}_i | \mathcal{E}^{[i]}) \tag{3}$$

$I(\mathcal{E}^{[i]}; \boldsymbol{y}_i)$ only depends on edges and labels, thus MLPs can only maximize $I(\boldsymbol{X}^{[i]}; \boldsymbol{y}_i | \mathcal{E}^{[i]})$. In the extreme case, $I(\boldsymbol{X}^{[i]}; \boldsymbol{y}_i | \mathcal{E}^{[i]})$ can be zero when $\boldsymbol{y}^{[i]}$ is conditionally independent from $\boldsymbol{X}^{[i]}$ given $\mathcal{E}^{[i]}$. For example, when every node is labeled by its degree or whether it forms a triangle. Then MLPs won't be able to fit meaningful functions, and neither will GLNNs. However, such cases are typically rare, and unexpected in practical settings our work is mainly concerned with. For real GML tasks, node features and structural roles are often highly correlated (Lerique et al., 2020), hence MLPs can achieve reasonable results even only based on node features, and thus GLNNs can potentially achieve much better results. We study the failure case of GLNNs by creating a low MI scenario in Section 6.

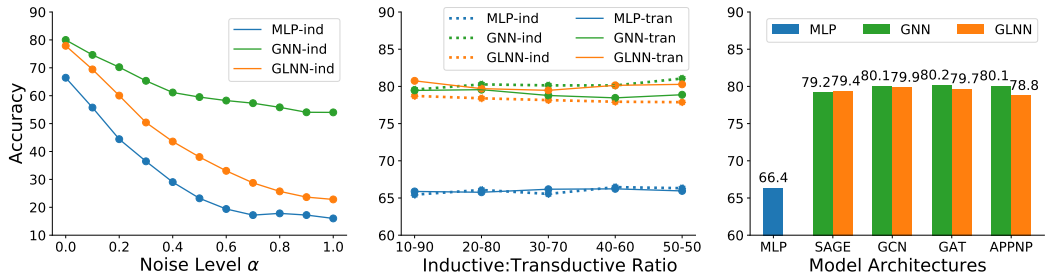

Figure 5: **Left**: Node feature noise. GLNN has comparable performance to GNNs only when nodes are less noisy. Adding more noise decreases GLNN performance faster than GNNs. **Middle**: Inductive split rate. Altering the inductive:transductive ratio in the production setting doesn't affect the accuracy much. **Right**: Teacher GNN architecture. GLNNs can learn from different GNN teachers to improve over MLPs and achieve comparable results. Accuracies are averaged over five CPF datasets.

6 Ablation Studies

In this section, we do ablation studies of GLNNs on node feature noise, inductive split rates, and teacher GNN architecture. Reported results are test accuracies averaged over five datasets in CPF.

Noisy node features. Following Section 5.8, we investigate failure cases of GLNN by adding different level of Gaussian noise to the node features to decrease their mutual information with labels. Specifically, we replace $\boldsymbol{X}$ with $\tilde{\boldsymbol{X}} = (1-\alpha)\boldsymbol{X} + \alpha\epsilon$. Here ϵ is isotropic Gaussian and independent from $\boldsymbol{X}$, and $\alpha \in [0, 1]$ denotes the noise level. We show the inductive performance of MLPs, GNNs, and GLNN under different noise levels in the left plot in Figure 5. We see that as α increases, the accuracy of MLPs and GLNNs decrease faster than GNNs, while the performance of GLNN and GNNs are still comparable with small αs. When α reaches 1, $\tilde{\boldsymbol{X}}$ and $\boldsymbol{Y}$ will become independent, which corresponds to the extreme case discussed in Section 5.8.

Inductive split rate. In Section 5.4, we use a 20-80 split of the test data for inductive evaluation. In the middle plot in Figure 5, we show the results under different split rates. We see that as the inductive portion increase, GNN and MLP performance stays roughly the same, and the GLNN inductive performance drops slightly. We only consider rates up to 50-50 since having 50% or even more inductive nodes is highly atypical in practice. When a large amount of new data are encountered, practitioners can opt to retrain the model on all the data before deployment.

Teacher GNN architecture. For experiments above, we used SAGE to represent GNNs. In Figure 5 (right), we now results with other GNNs as teachers, e.g. GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017), and APPNP Klicpera et al. (2019). We see that GLNNs can learn from different GNN teachers and improve over MLPs. Their performance with all four teachers is similar, with the GLNN distilled from APPNP slightly worse than others. In fact, a similar phenomenon has been observed in Yang et al. (2021a) as well, i.e. APPNP benefits the student the least. One possible reason is that the first step of APPNP is to utilize the node's own feature for prediction (prior to propagating over the graph), which is very similar to what the student MLP is doing, and thus provides less extra information to MLPs than other teachers.

7 Conclusion and Future Work

In this paper, we explored whether we can bridge the best of GNNs and MLPs to achieve accurate and fast GML models for deployment. We found that KD from GNNs to MLPs helps to eliminate inference graph dependency, which results in GLNNs that are 146×-273× faster than GNNs while enjoying competitive performance. We do a comprehensive study of GLNN properties. The promising results on 7 datasets across different domains show that GLNNs can be a handy choice for deploying latency-constraint models. In our experiments, the current version of GLNNs on the `Arxiv` dataset doesn't show competitive inductive performance. More advanced distillation techniques can potentially improve the GLNN performance, and we leave this investigation as future work.

REFERENCES

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Mincut pooling in graph neural networks. CoRR, abs/1907.00481, 2019. URL http://arxiv.org/abs/1907.00481.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In International Conference on Learning Representations, 2018. URL https://openreview.net/forum?id=rytstxWAW.

Lei Chen, Zhengdao Chen, and Joan Bruna. On graph neural networks versus graph-augmented {mlp}s. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=tiqI7w64JG2.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh (eds.), Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pp. 1725–1735. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20v.html.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 367–379, 2016. doi: 10.1109/ISCA.2016.40.

Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. Proceedings of the VLDB Endowment, 8 (12):1804–1815, 2015.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: Spectral clustering and normalized cuts. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04, pp. 551–556, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138881. doi: 10.1145/1014052.1014118. URL https://doi.org/10.1145/1014052.1014118.

Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Repmlp: Re-parameterizing convolutions into fully-connected layers for image recognition, 2021.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks, 2020.

Paul Graham. Want to start a startup? http://www.paulgraham.com/growth.html, 2012.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In Francis Bach and David Blei (eds.), Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research, pp. 1737–1746, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/gupta15.html.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, pp. 1024–1034, 2017.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, pp. 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

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. CoRR, abs/2005.00687, 2020. URL https://arxiv.org/abs/2005.00687.

Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation for graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 997–1005, 2020.

Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting numerical precision variability in deep neural networks. In Proceedings of the 2016 International Conference on Supercomputing, pp. 23, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR (Poster), 2015. URL http://arxiv.org/abs/1412.6980.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank, 2019.

Sébastien Lerique, Jacob Levy Abitbol, and Márton Karsai. Joint embedding of structure and features via graph convolutional networks. Applied Network Science, 5(1):1–24, 2020.

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

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In International Conference on Machine Learning (ICML), 2021.

Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps, 2021.

Luke Melas-Kyriazi. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet, 2021.

Zhenyue Qin, Dongwoo Kim, and Tom Gedeon. Rethinking softmax with cross-entropy: Neural network classifier as mutual information estimator, 2020.

Shyam Anil Tailor, Javier Fernandez-Marques, and Nicholas Donald Lane. Degree-quant: Quantization-aware training for graph neural networks. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=NSBrFgJAHg.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601, 2021.

Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. arXiv preprint arXiv:2105.03404, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

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

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315, 2019.

Tianxin Wei, Ziwei Wu, Ruirui Li, Ziniu Hu, Fuli Feng, Xiangnan He, Yizhou Sun, and Wei Wang. Fast adaptation for cold-start collaborative filtering with meta-learning. In 2020 IEEE International Conference on Data Mining (ICDM), pp. 661–670. IEEE, 2020.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

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

Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. Tinygnn: Learning efficient graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pp. 1848–1856, 2020.

Cheng Yang, Jiawei Liu, and Chuan Shi. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework, 2021a.

Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks, 2021b.

Dalong Zhang, Xin Huang, Ziqi Liu, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Zhiqiang Zhang, Lin Wang, Jun Zhou, Yang Shuang, et al. Agl: a scalable system for industrial-purpose graph machine learning. arXiv preprint arXiv:2003.02454, 2020.

Yiren Zhao, Duo Wang, Daniel Bates, Robert Mullins, Mateja Jamnik, and Pietro Lio. Learned low precision graph neural networks, 2020.

Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor K. Prasanna. Accelerating large scale real-time GNN inference using channel pruning. CoRR, abs/2105.04528, 2021. URL https://arxiv.org/abs/2105.04528.

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. arXiv preprint arXiv:1911.07323, 2019.

A Detailed Experiment Settings

A.1 Datasets

Here we provide a detailed description of the datasets we used to support our argument. Out of these datasets, 4 of them are citation graphs. Cora, Citeseer, Pubmed, ogbn-arxiv with the node features being descriptions of the papers, either bag-of-word vector, TF-IDF vector, or word embedding vectors.

In Table 5, we provided the basic statistics of these datasets.

Table 5: Dataset Statistics.

Dataset	# Nodes	# Edges	# Features	# Classes
`Cora`	2,485	5,069	1,433	7
`Citeseer`	2,110	3,668	3,703	6
`Pubmed`	19,717	44,324	500	3
`A-computer`	13,381	245,778	767	10
`A-photo`	7,487	119,043	745	8
`Arxiv`	169,343	1,166,243	128	40
`Products`	2,449,029	61,859,140	100	47

For all datasets, we follow the setting in the original paper to split the data. Specifically, for the five smaller datasets from the CPF paper, we use the CPF splitting strategy and each random seed corresponds to a different split. For the OGB datasets, we follow the OGB official splits based on time and popularity for `Arxiv` and `Products` respectively.

A.2 Model Hyperparameters

The hyperparameters of GNN models on each dataset are taken from the best hyperparameters provided by the CPF paper and the OGB official examples. For the student MLPs and GLNN s, unless otherwise specified with -wi or -Li, we set the number of layers and the hidden dimension of each layer to be the same as the teacher GNN, so their total number of parameters stays the same as the teacher GNN.

Table 6: Hyperparameters for GNNs on five datasets from the CPF paper.

	SAGE	GAT	APPNP
# layers	2	2	2
hidden dim	128	64	64
learning rate	0.01	0.01	0.01
weight decay	0.0005	0.01	0.01
dropout	0	0.6	0.5
fan out	5,5	-	-
attention heads	-	8	-
power iterations	-	-	10

Table 7: Hyperparameters for GraphSAGE on OGB datasets.

Dataset	`Arxiv`	`Products`
# layers	3	3
hidden dim	256	256
learning rate	0.001	0.001
weight decay	0	0
dropout	0.2	0.5
normalization	batch	batch
fan out	[5, 10, 15]	[5, 10, 15]

For GLNN s we do a hyperparameter search of learning rate from [0.01, 0.005, 0.001], weight decay from [0, 0.001, 0.002, 0.005, 0.01], and dropout from 0.1 to 0.6.

A.3 KNOWLEDGE DISTILLATION

We use the distillation method proposed in Hinton et al. (2015) as in Equation 1, the hard labels are found to be helpful, so nonzero λs was suggested. In our case, we did a little tuning for λ but didn't find nonzero λs to be very helpful. Therefore, we report all of our results with $\lambda = 0$, i.e. only the second term involving soft labels is effective. More careful tuning of λ should further improve the results since the searching space is strictly larger. We implemented a weighted version in our code, and we leave the choice of λ as future work.

A.4 THE TRANSDUCTIVE SETTING AND THE INDUCTIVE SETTING

Given $\mathcal{G}$, $\boldsymbol{X}$, and $\boldsymbol{Y}^L$, the goal of node classification can be divided into two different settings, i.e. transductive and inductive. In real applications, the former can correspond to predict missing attributes of a user based on the user profile and other existing users, and the latter can correspond to predict labels of some new nodes that are only seen during inference time. To create the inductive setting on a given dataset, we hold out some nodes along with edges connected to these nodes during training and use them for inductive evaluation only. These nodes and edges are picked from the test data. Using notation defined above, we pick the inductive nodes $\mathcal{V}^U_{ind} \subset \mathcal{V}^U$, which partitions $\mathcal{V}^U$ into the disjoint inductive subset and observed subset, i.e. $\mathcal{V}^U = \mathcal{V}^U_{obs} \sqcup \mathcal{V}^U_{ind}$. Then we can take all the edges connected to nodes in $\mathcal{V}^U_{ind}$ to further partition the whole graph, so we end up with $\mathcal{G} = \mathcal{G}_{obs} \sqcup \mathcal{G}_{ind}$, $\boldsymbol{X} = \boldsymbol{X}^L \sqcup \boldsymbol{X}^U_{obs} \sqcup \boldsymbol{X}^U_{ind}$, and $\boldsymbol{Y} = \boldsymbol{Y}^L \sqcup \boldsymbol{Y}^U_{obs} \sqcup \boldsymbol{Y}^U_{ind}$. We show the input and output of both settings using the notations below.

We visualize the difference between the inductive setting and the transductive setting in Figure 6.

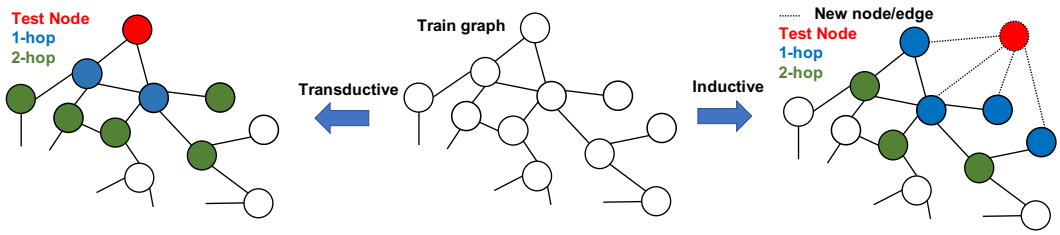

Figure 6: The transductive setting and inductive setting illustrated by a 2-layer GNN. The **middle** shows the original graph used for training. The **left** shows the transductive setting, where the test node is in red and within the graph. The **right** shows the inductive setting, where the test node is an unseen new node.

A.5 IMPLEMENTATION AND HARDWARD DETAILS

The experiments on both baselines and our approach are implemented using PyTorch, the DGL (Wang et al., 2019) library for GNN algorithms, and Adam (Kingma & Ba, 2015) for optimization. We run all experiments on a machine with 80 Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz CPUs, and a single NVIDIA V100 GPU with 16GB RAM.

B SPACE AND TIME COMPLEXITY OF GNNS VS. MLPS

In Table 1, the model comparison was between equal-sized MLPs (GLNNs) and GNNs. This is how people used to compare models for the sake of fair space complexity, but it is not a completely fair comparison for cross-model comparison, especially MLPs vs. GNNs. To do inference with GNNs, the graph needs to be loaded in the memory either entirely or batch by batch, which will take a much larger space than the model parameters. Thus, the space complexity of GNNs is actually much higher than equal-sized MLPs. From the time complexity perspective, the major inference latency of GNNs comes from the data dependency as shown in Section 4. Under the same setting as Figure

1, we show in the left part of Figure 3 that even a 5-layer MLP with 8 times wider hidden layers still runs much faster than a single-layer SAGE. Another example of cross-model comparison is Transformers vs. RNNs. Large Transformers can have more parameters than RNNs because of the attention mechanism, but they also run faster than RNNs in general.

Moreover, the comparison of GNNs and MLPs with the same number of parameters ignores the inductive bias difference. GNNs are advantaged since they get information from two sources when they do inference, i.e. the learned parameters and the graph data. For GNNs, there is an inductive bias of neighbor nodes influencing each other, so the knowledge missing from model parameters can be complemented. For MLPs, model parameters are the only source for them to remember the learned knowledge. They are being very flexible with less inductive bias, which could require more parameters for the model to learn (see Section 5.6 for a more detailed discussion of inductive bias).

In Table 1, we saw that for equal-sized comparison, GLNNs are not as good as GNNs on the OGB datasets. Following the discussion above and given the GLNNs used in Table 1 are relatively small (3 layers and 256 hidden dimensions) for millions of nodes in the OGB datasets, we ask whether this gap can be mitigated by increasing the MLP and thus GLNN sizes. Our answer is shown in Table 2.

C Consistency Measure of Model Predictions and Graph Topology Based on Min-Cut

We introduce a metric to measure the consistency between model predictions and graph topology based on the min-cut problem in Section 5.6. The K-way normalized min-cut problem, or simply min-cut, partitions N nodes in $\mathcal{V}$ into K disjoint subsets by removing the minimum volume of edges. According to Dhillon et al. (2004), the min-cut problem can be expressed as

$$\begin{aligned} &\max \frac{1}{K} \sum_{k=1}^{K} \frac{\boldsymbol{C}_k^T \boldsymbol{A} \boldsymbol{C}_k}{\boldsymbol{C}_k^T \boldsymbol{D} \boldsymbol{C}_k} \\ &s.t. \quad \boldsymbol{C} \in \{0,1\}^{N \times K}, \boldsymbol{C}\mathbf{1}_K = \mathbf{1}_N \end{aligned} \tag{4}$$

with $\boldsymbol{C}$ being the node assignment matrix that partitions $\mathcal{V}$, i.e. $\boldsymbol{C}_{i,j} = 1$ if node i is assigned to class j. $\boldsymbol{A}$ being the adjacency matrix and $\boldsymbol{D}$ being the degree matrix. This quantity we try to maximize here tells us whether the assignment is consistent with the graph topology. The bigger it is, the less edges need to be removed, and the assignment is more consistent with existing connections in the graph. In Bianchi et al. (2019), the authors show that when replacing the hard assignments $\boldsymbol{C} \in \{0,1\}^{N \times K}$ with a soft classification probability $\hat{\boldsymbol{Y}} \in [0,1]^{N \times K}$, a cut loss $\mathcal{L}_{cut}$ in Equation 2 can become a good approximation of Equation 4 and be used as the measuring metric.

Table 8: GLNN predictions are much more consistent with the graph topology than MLPs. We show the $\mathcal{L}_{cut}$ values of GNNs, MLPs, and GLNN s on five CPF datasets. GLNN $\mathcal{L}_{cut}$ values become pretty close to the high $\mathcal{L}_{cut}$ values of GNNs, which were closely related to the GNN inductive bias.

Datasets	SAGE	MLP	GLNN
Cora	0.9347	0.7026	0.8852
Citeseer	0.9485	0.7693	0.9339
Pubmed	0.9605	0.9455	0.9701
A-computer	0.9003	0.6976	0.8638
A-photo	0.8664	0.7069	0.8398
Average	0.9221	0.7644	0.8986

D Expressiveness of GNNs vs. MLPs in Terms of Equivalence Classes of Rooted Graphs

In Chen et al. (2021), the expressiveness of GNNs and GA-MLPs were theoretically quantified in terms of induced equivalence classes of rooted graphs. We adopt their framework and perform a similar analysis for GNNs vs. MLPs. We first define rooted graphs.

Definition 1 (Rooted Graph)**.** *A rooted graph, denoted as $\mathcal{G}^{[i]}$ is a graph with one node i in $\mathcal{G}^{[i]}$ designated as the root. GNNs, GA-MLPs, and MLPs can all be considered as functions on rooted graphs. The goal of a node-level task on node i with label $\boldsymbol{y}_i$ is to fit a function to the input-output pairs ($\mathcal{G}^{[i]}$, $\boldsymbol{y}_i$).*

We denote the space of rooted graphs as $\mathcal{E}$. Following Chen et al. (2021), the expressive power of a model on graph data is evaluated by its ability to approximate functions on $\mathcal{E}$. This is further characterized as the number of induced equivalence classes of rooted graphs on $\mathcal{E}$, with the equivalence relation defined as the following. Given a family of functions $\mathcal{F}$ on $\mathcal{E}$, we define an equivalence relation $\simeq_{\mathcal{E},\mathcal{F}}$ among all rooted graphs such that $\forall \mathcal{G}^{[i]}, \mathcal{G}'^{[j]} \in \mathcal{E}, \mathcal{G}^{[i]} \simeq_{\mathcal{E},\mathcal{F}} \mathcal{G}'^{[j]}$ if and only if $\forall f \in \mathcal{F}, f(\mathcal{G}^{[i]}) = f(\mathcal{G}'^{[j]})$. We now give a proposition to characterize the GNN expressive power (proof in Appendix E).

Proposition 1. *With $\mathcal{X}$ denotes the set of all possible node features and assuming $|\mathcal{X}| \geq 2$, with m denotes the maximum node degree and assuming $m \geq 3$, the total number of equivalence classes of rooted graphs induced by an L-layer GNN is lower bounded by $\binom{|\mathcal{X}|+m-2}{m-1}^{2^L-1}$.*

As shown in Proposition 1, the expressive power of GNNs grows doubly-exponentially in the number of layers L, which means it grows linearly in L after taking $\log(\log(\cdot))$. The expressive power GA-MLPs only grows exponentially in L as shown in Chen et al. (2021). Under this framework, the expressive power of MLPs, which corresponds to a 0-layer GA-MLP, is $|\mathcal{X}|$. Since the former is much larger than the latter, the conclusion will be GNNs are much more expressive than MLPs. The gap between these two numbers indeed exists, but empirically this gap will only make a difference when $|\mathcal{X}|$ is small. As in Chen et al. (2021), both the lower bound proof and the constructed examples showing GNNs are more powerful than GA-MLPs assumed $|\mathcal{X}| = 2$. In real applications and datasets considered in this work, the node features can be high dimensional vectors like bag-of-words, which makes $|\mathcal{X}|$ enormous. Thus, this gap doesn't matter much empirically.

E Proof of The Proposition 1

To prove Proposition 1, we first define rooted aggregation trees, which is similar to but different from rooted graphs.

Definition 2 (Rooted Aggregation Tree)**.** *The depth-K rooted aggregation tree of a rooted graph $\mathcal{G}^{[i]}$ is a depth-K rooted tree with a (possibly many-to-one) mapping from every node in the tree to some node in $\mathcal{G}^{[i]}$, where (i) the root of the tree is mapped to node i, and (ii) the children of every node j in the tree are mapped to the neighbors of the node in $\mathcal{G}^{[i]}$ to which j is mapped.*

A rooted aggregation tree can be obtained by unrolling the neighborhood aggregation steps in the GNNs. An illustration of rooted graphs and rooted aggregation trees can be found in Chen et al. (2021) Figure 4. We denote the set of all rooted aggregation trees of depth L using $\mathcal{T}_L$. Then we use $\mathcal{T}_{L,\mathcal{X},m}$ to denote a subset of $\mathcal{T}_L$, where the node features belong to $\mathcal{X}$, and all the nodes have exactly degree m (m children), and at least two nodes out of these m nodes have different features. In other words, a node can't have all identical children. With rooted aggregation trees defined, we are ready to prove Proposition 1. The proof is adapted from the proof of Lemma 3 in Chen et al. (2021).

Proof. Since the number of equivalence classes on $\mathcal{E}$ induced by the family of all depth-L GNNs consists of all rooted graphs that share the same rooted aggregation tree of depth-L (Chen et al., 2021), the lower bound problem in Proposition 1 can be reduced to lower bound $|\mathcal{T}_L|$, which can be further reduced to lower bound the subset $|\mathcal{T}_{L,\mathcal{X},m}|$. We now show $|\mathcal{T}_{L,\mathcal{X},m}| \geq \binom{|\mathcal{X}|+m-2}{m-1}^{2^L-1}$ inductively.

When $L = 1$, the root of the tree can have $|\mathcal{X}|$ different choices. For the children nodes, we pick m features from $|\mathcal{X}|$ and repetitions are allowed. This leads to $\binom{|\mathcal{X}|+m-1}{m}$ cases. Therefore, $\mathcal{T}_{L+1,\mathcal{X},m} = |\mathcal{X}|\binom{|\mathcal{X}|+m-1}{m} \geq \binom{|\mathcal{X}|+m-2}{m-1}$.

Assuming the statement holds for L, we show it holds for $L+1$ by constructing trees in $\mathcal{T}_{L+1,\mathcal{X},m}$ from $T, T' \in \mathcal{T}_{L,\mathcal{X},m}$. We do this by assigning node features in $\mathcal{X}$ to the m children of each leaf

node in T and T'. First note that when T and T' are two non-isomorphic trees, two depth-L+1 trees constructed from T and T' will be different no matter how the node features are assigned. Now we consider all the trees can be constructed from T by assign node features of children to leaf nodes.

We first consider all paths from the root to leaves in T. Each path consists of a sequence of nodes where the node features form a one-to-one mapping to an L-tuple $\tau \in \{(x_1, \ldots, x_L) : x_i \in \mathcal{X}\}$. Leaf nodes are called *node under* τ if the path from the root to it corresponds to τ. The children of nodes under different τs are always distinguishable, and thus any assignments lead to distinct rooted aggregation trees of depth $L+1$. The assignment of children of nodes under the same τ, on the other hand, could be overcounted. Therefore, to lower bound $\mathcal{T}_{L+1,\mathcal{X},m}$, we only consider a special way of assignments to avoid over counting, which is that children of all nodes under the same τ are assigned the same set of features.

Since we assumed that at least two nodes of T have different features, there are at least 2^L different τs corresponding to the path from the root to leaves. For a leaf node j under a fixed τ, one of its children needs to have the same feature as j's parent node. This restriction is due to the definition of rooted aggregation trees. Therefore, we only pick features for the other $m-1$ nodes, which will be $\binom{|\mathcal{X}|+m-2}{m-1}$ cases for each j. Then through this construction, the total number of depth-L+1 trees from T can be lower bounded by $\binom{|\mathcal{X}|+m-2}{m-1}^{2^L}$. Finally, we have this lower bound holds for all $T \in \mathcal{T}_{L,\mathcal{X},m}$, so we derive $\mathcal{T}_{L+1,\mathcal{X},m} \geq \binom{|\mathcal{X}|+m-2}{m-1}^{2^L} \mathcal{T}_{L,\mathcal{X},m}$, and $\mathcal{T}_{L,\mathcal{X},m} \geq \binom{|\mathcal{X}|+m-2}{m-1}^{\sum_{l=1}^{L} 2^l} = \binom{|\mathcal{X}|+m-2}{m-1}^{2^L-1}$

□