DEEP GRAPH NEURAL NETWORKS WITH SHALLOW SUBGRAPH SAMPLERS

Anonymous authors
Paper under double-blind review

ABSTRACT

While Graph Neural Networks (GNNs) are powerful models for learning representations on graphs, most state-of-the-art models do not have significant accuracy gain beyond two to three layers. Deep GNNs fundamentally need to address: 1). *expressivity* challenge due to oversmoothing, and 2). *computation* challenge due to neighborhood explosion. We propose a simple "deep GNN, shallow sampler" design principle to improve both the GNN accuracy and efficiency — to generate representation of a target node, we use a deep GNN to pass messages only within a shallow, localized subgraph. A properly sampled subgraph may exclude irrelevant or even noisy nodes, and still preserve the critical neighbor features and graph structures. The deep GNN then smooths the informative local signals to enhance feature learning, rather than oversmoothing the global graph signals into just "white noise". We theoretically justify why the combination of deep GNNs with shallow samplers yields the best learning performance. We then propose various sampling algorithms and neural architecture extensions to achieve good empirical results. Experiments on five large graphs show that our models achieve significantly higher accuracy and efficiency, compared with state-of-the-art.

1 INTRODUCTION

Graph Neural Networks (GNNs) have now become the state-of-the-art models for graph mining (Wu et al., 2020; Hamilton et al., 2017b; Zhang et al., 2019), facilitating applications such as social recommendation (Monti et al., 2017; Ying et al., 2018; Pal et al., 2020), knowledge understanding (Schlichtkrull et al., 2018; Park et al., 2019; Zhang et al., 2020) and drug discovery (Stokes et al., 2020; Lo et al., 2018). With the numerous architectures proposed (Kipf & Welling, 2016; Hamilton et al., 2017a; Veličković et al., 2018), it still remains an open question how to effectively design *deep* GNNs. There are two fundamental obstacles that are intrinsic to the underlying graph structure:

- *Expressivity* challenge: deep GNNs tend to oversmooth (Li et al., 2018). They collapse embeddings of different nodes into a fixed low-dimensional subspace after repeated neighbor mixing.
- *Computation* challenge: deep GNNs recursively expand the adjacent nodes along message passing edges. The neighborhood size may grow exponentially with model depth (Chen et al., 2017).

Due to oversmoothing, one of the most popular GNN architectures, Graph Convolutional Network (GCN) (Kipf & Welling, 2016), has been theoretically proven as incapable of scaling to deep layers (Oono & Suzuki, 2020; Rong et al., 2020; Huang et al., 2020). Remedies to overcome the GCN limitations are two-folded. From the neural architecture perspective, researchers are actively seeking for more expressive neighbor aggregation operations (Veličković et al., 2018; Hamilton et al., 2017a; Xu et al., 2018a), or transferring design components (such as residual connection) from deep CNNs to GNNs (Xu et al., 2018b; Li et al., 2019; Huang et al., 2018). From the data perspective, various works (Klicpera et al., 2019a;b; Bojchevski et al., 2020) revisit classic graph analytic algorithms to reconstruct a graph with nicer topological property. The two kinds of works can also be combined to jointly improve the quality of message passing in deep GNNs.

All the above GNN variants take a "global" view on the input graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ — *i.e.*, all nodes are considered as belonging to the same $\mathcal{G}$, whose size can often be massive. To generate the node embedding, no matter how we modify the architecture and the graph structure, a deep enough GNN

would always propagate the influence from the entire node set $\mathcal{V}$ into a single target node. Intuitively, for a large graph, most nodes in $\mathcal{V}$ barely provide any useful information to the target nodes. We thus regard such "global view" on $\mathcal{G}$ as one of the root causes for both the expressivity and computation challenges discussed above. In this work, for the node embedding task, we take an alternative "local view" and interpret the GNN input as $\mathcal{V} = \bigcup_{v\in\mathcal{V}} \mathcal{V}_{[v]}$ and $\mathcal{E} = \bigcup_{v\in\mathcal{V}} \mathcal{E}_{[v]}$. In other words, each target node v belongs to some small graph $\mathcal{G}_{[v]}$ capturing the characteristics of only the node v. The entire input graph $\mathcal{G}$ is observed as the union of all such local yet latent $\mathcal{G}_{[v]}$. Such simple global-to-local switch of perspective enables us to address both the expressivity and computation challenges *without* resorting to alternative GNN architectures or reconstructing the graph.

Present work: SHADOW-GNN. We propose a "Deep GNN, shallow sampler" design principle that helps improve the expressive power and inference efficiency of various GNN architectures. We break the conventional thinking that an L-layer (deep) GNN has to aggregate L-hop (faraway) neighbors. We argue that the GNN receptive field for a target node should be *shallower* than the GNN depth. In other words, an L-layer GNN should only operate on a small subgraph $\mathcal{G}_{[v]}$ surrounding the target node v, where $\mathcal{G}_{[v]}$ consists of (part of) the L_0-hop neighborhood. The deep *vs.* shallow comparison is reflected by setting $L_0 < L$. We name such a GNN on $\mathcal{G}_{[v]}$ as a SHADOW-GNN. We justify our design principle from two aspects. Firstly, *why do we need the neighborhood to be shallow*? As a motivating example, the average number of 4-hop neighbors for the `ogbn-products` graph (Hu et al., 2020) is 0.6M, corresponding to 25% of the full graph size. Blindly encoding the 0.6M node features into a single embedding vector can create the "information bottleneck" (Alon & Yahav, 2020). The irrelevant information from the majority of the 0.6M nodes may also "dilute" the truly useful signals from a small set of close neighbors. A simple solution to the above issues is to manually create a shallow neighborhood by subgraph sampling. The second question regarding SHADOW-GNN is: *why do we still need deep GNNs*? Using more number of layers than the number of hops means the same pair of nodes may exchange messages with each other multiple times. Intuitively, this helps the GNN better absorb the subgraph information. Theoretically, we prove that a GNN deeper than the hops of the subgraph can be more powerful than the 1-dimensional Weisfeiler-Lehman test (Shervashidze et al., 2011). A shallow GNN, on the contrary, cannot accurately learn certain simple functions such as unweighted mean of the shallow neighborhood features. Note that with GCN as the backbone, a SHADOW-GCN still performs signal smoothing in each layer. However, the important distinction is that a deep GCN smooths the full $\mathcal{G}$ regardless of the target node, while a SHADOW-GCN constructs a customized smoothing domain $\mathcal{G}_{[v]}$ for each target v. The variance in those smoothing domains created by SHADOW-GCN encourages variances in the node embedding vectors. With such intuition, our analysis shows that SHADOW-GNN does not oversmooth. Finally, since the sizes of the shallow neighborhoods are independent of the GNN depth, the computation challenge due to neighbor explosion is automatically addressed.

We propose various subgraph samplers for SHADOW-GNN, including the simplest k-hop sampler and a sampler based on personalized PageRank, to improve the inference accuracy and computation efficiency. By experiments on five standard benchmarks, our SHADOW-SAGE and SHADOW-GAT models achieve significant accuracy gains compared with the original GraphSAGE and GAT models. In the meantime, the inference cost is reduced by orders of magnitude.

2 Related Work and Preliminaries

Deep GNNs. Recently, numerous GNN models (Kipf & Welling, 2016; Defferrard et al., 2016; Hamilton et al., 2017a; Veličković et al., 2018; Xu et al., 2018b;a) have been proposed. In general, the input to a GNN is the graph $\mathcal{G}$, and the outputs are representation vectors for each node, capturing both the feature and structural information of the neighborhood. Most state-of-the-art GNNs use shallow models (*i.e.*, 2 to 3 layers). As first proposed by Li et al. (2018) and further elaborated by Luan et al. (2019); Oono & Suzuki (2020); Zhao & Akoglu (2020); Huang et al. (2020), one of the major challenges to deepen GNNs is the "oversmoothing" of node features — each layer aggregation pushes the neighbor features towards similar values. Repeated aggregation over many layers results in node features being averaged over the full graph. A deep GNN may thus generate indistinguishable embeddings for different nodes. Viewing oversmoothing as a limitation of the layer aggregation, researchers develop alternative architectures. AS-GCN (Huang et al., 2018), DeepGCN (Li et al., 2019) and JK-net (Xu et al., 2018b) use skip-connection across layers. MixHop (Abu-El-Haija et al., 2019), Snowball (Luan et al., 2019) and DAGNN (Liu et al., 2020) enable multi-hop message

passing within a single layer. GraphSAGE (Hamilton et al., 2017a) and GCNII (Ming Chen et al., 2020) encourage self-to-self message passing which effectively form an implicit skip-connection. GIN (Xu et al., 2018a) and DeeperGCN (Li et al., 2020a) propose more expressive neighbor aggregation operations. All the above focus on *architectural exploration*, which is a research direction orthogonal to ours. We can construct the SHADOW version of these GNNs in a plug-and-play fashion. Lastly, DropEdge (Rong et al., 2020) and Bayesian-GDC (Hasanzadeh et al., 2020) propose regularization techniques by adapting dropout (Srivastava et al., 2014) to graphs. Such techniques are only applied during training, and so oversmoothing during inference may not be alleviated.

Learning from structural information. Another line of research is to go beyond the layer-wise message passing and more explicitly utilize the graph structural information (Wu et al., 2019; Klicpera et al., 2019a; Bojchevski et al., 2020; Liu et al., 2020; Frasca et al., 2020; You et al., 2019; Li et al., 2020b). In particular, APPNP (Klicpera et al., 2019a) and PPRGo (Bojchevski et al., 2020) utilize the personalized PageRank (Page et al., 1999) algorithm to re-define neighbor connections — instead of propagating features along the (noisy) graph edges, any nodes of structural significance can directly propagate to the target node. Other related methods such as GDC (Klicpera et al., 2019b) and AM-GCN (Wang et al., 2020) reconstructs the adjacency matrix in each GNN layer to short-cut important multi-hop neighbors. Note that all the above methods takes a global view on $\mathcal{G}$ and operate the neural networks on the *full graph*. On the other hand, the idea of using subgraph samples to improve the GNN efficiency has also been explored. For example, SEAL (Zhang & Chen, 2018) extracts local k-hop enclosing subgraphs around links to perform link prediction. GraphSAINT (Zeng et al., 2020) propose random walk-based samplers to construct minibatches during training. Note that the training sampler of GraphSAINT aims at approximating the feature aggregation in the full L-hop neighborhood. So the sampling of GraphSAINT cannot be applied to the inference phase, leaving the issue of oversmoothing and neighbor exploration unaddressed.

Notations. We focus on the node classification task, although our design principle can be naturally extended to other tasks. Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ be an undirected graph, with node set $\mathcal{V}$, edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$. The u-th row of $\boldsymbol{X}$ corresponds to the length-d feature of node u. Let $\boldsymbol{A}$ be the adjacency matrix of $\mathcal{G}$ where $A_{u,v} = 1$ if edge $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. Denote $\widetilde{\boldsymbol{A}}$ as the adjacency matrix after symmetric normalization (used by GCN), and $\widehat{\boldsymbol{A}}$ as the one after random walk normalization (used by GraphSAGE). Let subscript "$[u]$" mark the quantities corresponding to a small subgraph surrounding node u. For example, the subgraph itself is $\mathcal{G}_{[u]}$. For an L-layer GNN, let superscript "(ℓ)" denote the layer-ℓ quantities ($1 \leq \ell \leq L$). Let $d^{(\ell)}$ be the number of channels for layer ℓ; $\boldsymbol{H}^{(\ell-1)} \in \mathbb{R}^{|\mathcal{V}| \times d^{(\ell-1)}}$ and $\boldsymbol{H}^{(\ell)} \in \mathbb{R}^{|\mathcal{V}| \times d^{(\ell)}}$ be the input and output feature matrices. Thus, $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ and $d^{(0)} = d$. Further, let $\boldsymbol{Y} = \boldsymbol{H}^{(L)}$. The operation of a layer can be abstracted as $\boldsymbol{H}^{(\ell)} = f\left(\boldsymbol{H}^{(\ell-1)}, \boldsymbol{A}; \boldsymbol{W}^{(\ell)}\right)$, where $\boldsymbol{W}^{(\ell)}$ are the learnable weights.

3 DEEP GNN, SHALLOW SAMPLER

Algorithm 1 SHADOW-GNN inference algorithm

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$; Target nodes $\mathcal{V}_t$; GNN model;
Output: Node embedding matrix $\boldsymbol{Y}$ for $\mathcal{V}_t$;

1: **for** $v \in \mathcal{V}_t$ **do**
2: Get $\mathcal{G}_{[v]}\left(\mathcal{V}_{[v]}, \mathcal{E}_{[v]}, \boldsymbol{X}_{[v]}\right)$ by SAMPLE on $\mathcal{G}$
3: Build L-layer GNN with layer operation f
4: $\boldsymbol{y}_v \leftarrow \left[f^L\left(\boldsymbol{X}_{[v]}, \boldsymbol{A}_{[v]}\right)\right]_{v,:}$

"Deep GNN, shallow sampler" is a design principle to improve the expressivity and efficiency of GNNs without modifying the layer architecture. Based on this principle, we construct SHADOW-GNN by subgraph sampling. SHADOW-GNN uses the same sampling procedure during both training and inference. The inference algorithm is shown in Algorithm 1. In line 4, we use f^L to denote the composition of the function f for the L layers. Operation $[\cdot]_{v,:}$ slices the v-th row of the matrix. We discuss in Section 3.2 how to design SAMPLE to return informative shallow neighborhood.

Notice that the normal GNN is a special kind of SHADOW-GNN. Under the normal setup, the GNN operates on the full graph $\mathcal{G}$ and the L layers propagate the influence from *all* the neighbors up to L hops away from the target node. Such a GNN is equivalent to a SHADOW-GNN when SAMPLE returns the full L-hop subgraph. However, following our principle, a good SAMPLE should encourage most of $\mathcal{V}_{[v]}$ to concentrate within the L_0-hop neighborhood, where $L_0 < L$. Figures 5, 6

in Appendix show the difference in the neighborhood composition for normal GNN and SHADOW-GNN. Section 3.1 discusses the benefits of making the subgraph shallow and the GNN deep.

3.1 ANALYSIS ON EXPRESSIVITY

The SHADOW-GNN design is motivated by the following:

1). A shallow neighborhood is *sufficient* for the GNN to learn a good node representation;

2). A shallow neighborhood is *necessary* to reduce the effect of noise on the GNN;

3). A deep GNN is *necessary* to be expressive on the shallow neighborhood.

Point 1 is true in many real-world scenarios, and is also justified by the γ-decaying theorem (Zhang & Chen, 2018): to estimate various important graph metrics of a node from its L-hop neighborhood, the estimation error decays exponentially with the number of hops L. The understanding on Point 2 is two-folded: real-world graphs are likely to include noisy nodes and edges due to erroneous graph construction (Klicpera et al., 2019b). Additionally, even without errors, a node u may still be regarded as "white noise" to the target node v, simply because u is far away from v and u's information is irrelevant. The first kind of noise may be handled by filtering out the high frequency components of the node signals (Wu et al., 2019) or enforcing low rank constraints on the adjacency matrix (Grover et al., 2019; Jin et al., 2020). The second kind of noise *cannot be filtered* as long as the GNN aggregates the full L-hop neighborhood. In this regard, sampling provides an additional mechanism to deal with both kinds of noises. Also, since we customize different subgraphs for different target nodes, `SAMPLE` filters noise at the node level rather than the graph level. For Point 3, while theoretical understanding on the benefit of deep neural networks has been well established for Multi-Layer Perceptrons (MLPs) (Telgarsky, 2016), such conclusion does not directly transfer to GNNs. For SHADOW-GNN, we prove a deeper model is more expressive than a shallower one.

Expressivity comparison with "deep GNN, deep sampler". When increasing the depth of a normal GNN, the neighborhood around a target node (i.e., the full L-hop subgraph) also expands. Eventually, a deep normal GNN on a connected $\mathcal{G}$ will propagate influence of all $\mathcal{V}$ into a single target node. As discussed in Point 2 above, most $\mathcal{V}$ only act as "white noise" to a target node, thus causing difficulties in learning regardless of the layer propagation function.

We pick the GCN architecture for case study. When the GCN depth grows, we analyze how a shallow sampler can filter out the "white noise" in the neighborhood and help preserve target node specific information. Following the setup of Li et al. (2018); Luan et al. (2019); Ming Chen et al. (2020); Zhao & Akoglu (2020), we study the asymptotic behavior of repeatedly applying the neighbor aggregation operation. *i.e.*, we analyze how much information is preserved in $\boldsymbol{M} = \lim_{L\to\infty} \widetilde{\boldsymbol{A}}^L \boldsymbol{X}$.

Proposition 3.1. *An ∞-layer* SHADOW-GCN *generates the neighbor aggregation for v as:*

$$\boldsymbol{m}_{[v]} = \left[e_{[v]}\right]_v \cdot \left(\boldsymbol{e}_{[v]}^{\mathsf{T}} \boldsymbol{X}_{[v]}\right) \tag{1}$$

where $\boldsymbol{e}_{[v]}$ is defined by $\left[e_{[v]}\right]_u = \sqrt{\frac{\delta_{[v]}(u)}{\sum_{w\in\mathcal{V}_{[v]}} \delta_{[v]}(w)}}$; $\delta_{[v]}(u)$ returns the degree of u in $\mathcal{G}_{[v]}$ plus 1.

OVERSMOOTHING OF NORMAL GCN With large enough L, the full L-hop neighborhood becomes $\mathcal{V}$ (assuming connected $\mathcal{G}$). So $\forall\ u, v$, we have $\mathcal{G}_{[u]} = \mathcal{G}_{[v]} = \mathcal{G}$, implying $\boldsymbol{e}_{[u]} = \boldsymbol{e}_{[v]}$ and $\boldsymbol{X}_{[u]} = \boldsymbol{X}_{[v]} = \boldsymbol{X}$. From Proposition 3.1, the aggregation converges to a point where no feature and little structural information of the target is preserved. The only information in $\boldsymbol{m}_{[v]}$ is v's degree.

LOCAL-SMOOTHING OF SHADOW-GCN With a fixed shallow subgraph, no matter how many times we aggregate using $\widetilde{\boldsymbol{A}}_{[v]}$, the many layers will not include the faraway irrelevant nodes. We can see $\boldsymbol{m}_{[v]}$ as a linear combination of the neighbor node features $\boldsymbol{X}_{[v]}$. Increasing the number of layers only pushes the coefficients of each neighbor features to the stationary values. The domain $\boldsymbol{X}_{[v]}$ of such linear transformation is solely determined by `SAMPLE` and is independent of the layer depth. Intuitively, if `SAMPLE` picks non-identical subgraphs for two nodes u and v, the aggregations should be different due to the different domains of the linear transformation. Therefore, SHADOW-GCN preserves local feature information whereas normal GCN preserves *none*. For structural information

in $\boldsymbol{m}_{[v]}$, note that $\boldsymbol{e}_{[v]}$ is a normalized degree distribution of the subgraph around v, and $\left[e_{[v]}\right]_v$ indicates the role of the target node in the subgraph. If we let SAMPLE return the 1-hop subgraph, then $\left[e_{[v]}\right]_v$ alone already contains all the information preserved by a normal GCN, which is v's degree in $\mathcal{G}$. Since $\boldsymbol{e}_{[v]}$ additionally reflects the structure of v's ego-net, a deep SHADOW-GCN with a naïve 1-hop SAMPLE preserves more structural information than a deep GCN.

Theorem 3.2. *Let $\overline{\boldsymbol{m}}_{[v]} = \phi_{\mathcal{G}}(v) \cdot \boldsymbol{m}_{[v]}$ where $\phi_{\mathcal{G}}$ is any non-zero function only depending on the structural property of v. Let $\mathcal{M} = \{\overline{\boldsymbol{m}}_{[v]} \mid v \in \mathcal{V}\}$. Given $\mathcal{G}$, SAMPLE and some continuous probability distribution in $\mathbb{R}^{|\mathcal{V}| \times f}$ to generate $\boldsymbol{X}$, then $\overline{\boldsymbol{m}}_{[v]} \neq \overline{\boldsymbol{m}}_{[u]}$ if $\mathcal{V}_{[u]} \neq \mathcal{V}_{[v]}$, almost surely.*

Corollary 3.2.1. *Consider SAMPLE$_1$ where $\forall v \in \mathcal{V},\ \left|\mathcal{V}_{[v]}\right| \leq n$. Then $|\mathcal{M}| \geq \left\lceil \frac{|\mathcal{V}|}{n} \right\rceil$ almost surely.*

Corollary 3.2.2. *Consider SAMPLE$_2$ where $\forall\ u, v \in \mathcal{V},\ \mathcal{V}_{[v]} \neq \mathcal{V}_{[u]}$. Then $|\mathcal{M}| = |\mathcal{V}|$ almost surely.*

Theorem 3.2 proves SHADOW-GCN does not oversmooth, as 1). a normal GCN pushes the aggregation of same-degree nodes to the same point, while SHADOW-GCN with SAMPLE$_2$ ensures any two nodes (even with the same degree) can have different aggregation; 2). a normal GCN wipes out all the information in the initial node features after many times of aggregation, while SHADOW-GCN always preserves node feature information. In particular, if we set $\phi_{\mathcal{G}}(v) = \left(\delta_{[v]}(v)\right)^{-1/2}$, a normal GCN generates only one unique value of $\overline{\boldsymbol{m}}$, while SHADOW-GNN can still generate N for any $\phi_{\mathcal{G}}$.

Expressivity comparison with "shallow GNN, shallow sampler". Most state-of-the-art GNNs belong to the "shallow GNN, shallow sampler" category. However, we argue that a deep GNN is still desirable even though the neighborhood is shallow. In the following, we first give two examples showing when "shallow GNN, shallow sampler" fails to learn something that a SHADOW-GNN can. Then we prove that SHADOW-GNN is strictly more powerful than 1-dimensional Weisfeiler-Lehman test, thus more powerful than all standard "shallow GNN, shallow sampler" designs.

In Figure 1, consider the 2-hop neighborhood (black dots) of some target node v (blue star). In the first example, suppose another target node v' has a slightly different 2-hop structure as reflected by an additional edge (red dashed line). Assume all nodes have identical features. Then any 2-layer GNN cannot distinguish v and v', while a 3-layer GNN can. The red edge that distinguishes v and v' is not having any influence on v' until three times of message passing. In the second example, we consider a simple task of learning the unweighted mean of neighbor features, $\boldsymbol{\tau} = \frac{1}{\left|\mathcal{V}_{[v]}\right|} \sum_{u \in \mathcal{V}_{[v]}} \boldsymbol{x}_u$.

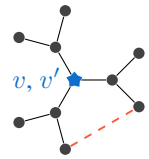

Figure 1: Example neighborhood

Suppose $\mathcal{G}_{[v]}$ consists of the star node, black nodes and black edges. Surprisingly, even with such a "regular" subgraph structure, a normal L-layer GraphSAGE cannot learn $\boldsymbol{\tau}$ accurately. The reasons are as follows. If $L > 2$, then normal GraphSAGE will include nodes outside $\mathcal{V}_{[v]}$, and the only way to always exclude the impact of these nodes is by setting the weights for that layer to 0 — this effectively reduces the number of layers to 2. Now since $\boldsymbol{\tau}$ is linear, ReLU is not desirable, and should be bypassed by shifting $\boldsymbol{X}$ with the bias parameters. The output of the neural net then simplifies to $\boldsymbol{\zeta} = \left[\sum_{\ell=1}^{L} \widehat{\boldsymbol{A}}_{[v]}^{\ell} \boldsymbol{X}_{[v]} \boldsymbol{W}_{\ell}\right]_{v,:}$, where $L = 2$ for normal GraphSAGE. For the given $\widehat{\boldsymbol{A}}_{[v]}$, there does not exist a $\boldsymbol{W}_{\ell}$ for GraphSAGE to satisfy $\boldsymbol{\tau} = \boldsymbol{\zeta}$ for any value of $\boldsymbol{X}_{[v]}$. On the other hand, with SAMPLE returning $\mathcal{G}_{[v]}$, SHADOW-SAGE can learn $\boldsymbol{\tau}$ with any precision if L is allowed to grow. The required L for SHADOW-SAGE to reach the desired precision is determined by the mixing time of $\widehat{\boldsymbol{A}}_{[v]}$. SHADOW-SAGE can learn the unweighted mean because the eigenvector corresponding to the largest eigenvalue of $\widehat{\boldsymbol{A}}_{[v]}$ is a uniform vector. In fact, a deep SHADOW-SAGE can learn the unweighted feature mean on any neighborhood.

Theorem 3.3. *Suppose we are given any GNN following the per-layer message passing design as:*

$$\boldsymbol{h}_v^{(\ell)} = f_1^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \sum_{u \sim v} f_2^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{h}_u^{(\ell-1)}\right)\right), \tag{2}$$

where $f_1^{(\ell)}$ and $f_2^{(\ell)}$ are the layer-ℓ update and message functions, respectively, implemented as MLPs; $u \sim v$ denotes u is v's 1-hop neighbor. Then, on the full L-hop subgraph, an L'-layer GNN is strictly more discriminative than an L-layer model, when $L' > L$. Moreover, with injective f_1 and f_2, the GNN model described above (using $L' > L$ layers on the L-hop subgraph) is strictly more discriminative than the 1-dimensional Weisfeiler-Lehman test.

See Appendix A for the proof. By Theorem 3.3, given a shallow sampler, a deep GNN is strictly more powerful than a shallow one. Even given a deep sampler, we can still use a "deeper" GNN to beat the "deep" GNN. Other practical concerns on deep GNNs include the difficulty in optimization and the computation cost. The optimal depth may be determined after considering such tradeoffs.

3.2 Sampler Design

A fundamental question behind any sampler is: what is the right metric to reflect neighbor importance? `SAMPLE` should be based on such metric, while at the same time be lightweight and scalable.

k-hop sampler. Starting from the target node v, the sampler traverses up to k hops. At a hop-ℓ node u, the sampler will add to its hop-$(\ell+1)$ node set either all neighbors of u, or b randomly selected neighbors of u. The subgraph $\mathcal{G}_{[v]}$ is induced from all the nodes selected by the sampler. Here k and b are the sampling parameters. Note that GraphSAGE also uses a k-hop sampler. However, the main difference is that SHADOW-GNN runs on the *induced subgraph* of the k-hop neighbors. Compared with the normal GraphSAGE, a SHADOW-SAGE is able to discover more complicated structural information from the subgraph constructed by our k-hop sampler (See Theorem 3.3).

PPR sampler. Personalized PageRank (PPR) has been recently combined with graph learning (Klicpera et al., 2019a; Bojchevski et al., 2020; Li et al., 2020b). However, none of the existing works have explored PPR in the context of subgraph sampling. Given a target v, our PPR sampler proceeds as follows: 1). Compute the approximate PPR vector $\boldsymbol{\pi}_v \in \mathbb{R}^{|\mathcal{V}|}$ for v. 2). Select the neighborhood $\mathcal{V}_{[v]}$ such that for $u \in \mathcal{V}_{[v]}$, the PPR score $[\pi_v]_u$ is large. 3). Construct the induced subgraph from $\mathcal{V}_{[v]}$. For step 1), even though vector $\boldsymbol{\pi}_v$ is of length-$|\mathcal{V}|$, most of the PPR values are close to zero. So we can use a fast algorithm (Andersen et al., 2006) to compute the approximate PPR score by traversing only the local region around v. Throughout the $\boldsymbol{\pi}_v$ computation, most of its entries remain as the initial value of 0. Therefore, the PPR sampler is scalable and efficient w.r.t. both time and space complexity. For step 2), we can either select $\mathcal{V}_{[v]}$ based on top-p values in $\boldsymbol{\pi}_v$, or based on some threshold $[\pi_v]_u > \theta$. Then p, θ are hyperparameters. For step 3), notice that our PPR sampler only uses PPR scores as a node filter. The original graph structure is still preserved among $\mathcal{V}_{[v]}$, due to the induced subgraph step. Such property differentiates our algorithm from existing ones such as APPNP (Klicpera et al., 2019a), GDC (Klicpera et al., 2019b) and PPRGo (Bojchevski et al., 2020). Specifically, the three related works all *reconstruct* the adjacency matrix by directly connecting the target v with nodes of top PPR scores. Reconstruction promotes feature propagation from direct neighbors at the cost of losing structural information, since all $\mathcal{V}_{[v]}$ would become 1-hop away from v. We show in Section 3.3 further connections between SHADOW-GNN and the prior arts.

Extensions. The k-hop and PPR samplers each assume *shortest path distance* and *random walk landing probability* as important metrics for measuring neighbor relevance. Following such rationale, we can design many more sampling algorithms. For example, instead of PPR scores, we can use Katz index (Katz, 1953) as our node filtering condition. This is another way to signify the importance of shortest path distance. If the number of common neighbors is important, we can filter $\mathcal{V}_{[v]}$ by SimRank (Jeh & Widom, 2002). Apart from structural information, `SAMPLE` can also utilize node features. For example, in the k-hop sampler, instead of randomly pick b neighbors, we can pick those with highest feature similarity (e.g., by cosine similarity or heat kernel (Wang et al., 2020)).

3.3 Architectural Extension

Subgraph ensemble. From Section 3.2, each reasonable `SAMPLE` reveals some perspectives of the graph, and they altogether present the full picture. We may then combine the information from various `SAMPLE`. Assume a set of C candidates $\{\texttt{SAMPLE}_i\}$, each returning $\left(\mathcal{G}_{[v]}\right)_i$. We can either: 1). Construct a union-subgraph $\mathcal{G}^\star_{[v]}$ for SHADOW-GNN to operate on, where $\mathcal{V}^\star_{[v]} = \bigcup_{i=1}^{C} \left(\mathcal{V}_{[v]}\right)_i$ and $\mathcal{E}^\star_{[v]} = \bigcup_{i=1}^{C} \left(\mathcal{E}_{[v]}\right)_i$, or 2). Use C parallel branches of L-layer GNN, each operating on $\left(\mathcal{G}_v\right)_i$. So v's output embedding of each branch i (i.e., a vector $\boldsymbol{z}_i$) can be aggregated by attention mechanism:

$$\left(\boldsymbol{Z}_{[v]}\right)_i = f_i^L\left(\left(\boldsymbol{X}_{[v]}\right)_i, \left(\boldsymbol{A}_{[v]}\right)_i\right); \qquad w_i = \texttt{MLP}\left(\boldsymbol{z}_i\right) \cdot \boldsymbol{q}; \qquad \boldsymbol{y}_v = \sum_{i=1}^{C} \widetilde{w}_i \cdot \boldsymbol{z}_i \tag{3}$$

where f_i^L is the function by the L-layer GNN at branch i; $\boldsymbol{z}_i$ is a row vector of $\left(\boldsymbol{Z}_{[v]}\right)_i$ corresponding to v; $\boldsymbol{q}$ is a learnable vector; $\widetilde{\boldsymbol{w}}$ is normalized from $\boldsymbol{w}$ by softmax. $\boldsymbol{y}_v$ is the final embedding. We can use identical GNN structure for all the C branches or even share weights among the branches.

$\{\texttt{SAMPLE}_i\}$ can also perform the same algorithm with different hyperparameters. For example, we can let $\texttt{SAMPLE}_i$ be the PPR sampler with different threshold θ_i. A SHADOW-SAGE-ensemble can imitate similar feature aggregation behavior as PPRGo (Bojchevski et al., 2020), while still having the additional advantage of learning rich subgraph structural information. Note that PPRGo generates embedding by 1-hop aggregation on the reconstructed graph: $\boldsymbol{\tau}_v = \sum_{u \in \mathcal{V}_{[v]}} \pi_u \boldsymbol{h}_v$, where $\pi_u = [\pi_v]_u$ and $\boldsymbol{h}_v = \texttt{MLP}(\boldsymbol{x}_v)$. We can partition $\mathcal{V}_{[v]} = \bigcup_{i=1}^{C} \mathcal{V}_{[v]}^i$ such that nodes in $\mathcal{V}_{[v]}^i$ have similar PPR scores approximated by $\widetilde{\pi}_i$, and $\widetilde{\pi}_i \leq \widetilde{\pi}_{i+1}$. So $\boldsymbol{\tau}_v \approx \sum_{i=1}^{C} \left(\rho_i \sum_{u \in \mathcal{V}_i'} \boldsymbol{h}_u\right)$, where $\rho_i = \widetilde{\pi}_i - \sum_{j<i} \widetilde{\pi}_j$ and $\mathcal{V}_i' = \bigcup_{k=i}^{C} \mathcal{V}_{[v]}^k$. Recall the discussion on learning unweighted feature mean of any $\mathcal{V}_{[v]}$ (Section 3.1). Setting $\theta_i = \widetilde{\pi}_i$ and $w_i = \rho_i$, SHADOW-SAGE can approximate $\boldsymbol{\tau}_v$. As our PPR sampler also preserves local structure, our model can be more expressive than PPRGo.

Sampling the reconstructed graph. Another extension is to build SHADOW-GNN on the graph reconstructed by graph diffusion (Klicpera et al., 2019b; Frasca et al., 2020). In the simplest case, we can apply our L-hop sampler on the reconstructed graph, and build a L'-layer GNN on the subgraph. When $L = L'$, we recover GDC (Klicpera et al., 2019b). When $L < L'$, we have SHADOW-GDC.

4 EXPERIMENTS

Setup. We perform node classification tasks on five large graphs: `Flickr` (Zeng et al., 2020), `Reddit` (Hamilton et al., 2017a), `Yelp` (Zeng et al., 2020), `ogbn-arxiv` (Hu et al., 2020) and `ogbn-products` (Hu et al., 2020). The `Flickr`, `Reddit` and `Yelp` graphs are under the inductive learning setting, while `ogbn-arxiv` and `ogbn-products` are transductive. The `ogbn-arxiv` and `ogbn-products` belong to the Open Graph Benchmark (Hu et al., 2020). The sizes of the five graphs ranges from around 9K nodes (`Flickr`) to around 2.5M nodes (`ogbn-products`). Consistent with their original paper, we use the metric of "accuracy" for `Flickr`, `Reddit`, `ogbn-arxiv` and `ogbn-products`, and "F1-micro score" for `Yelp`. See also Appendix for further details.

We consider six baselines and construct the corresponding SHADOW models: GCN (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017a), GAT (Veličković et al., 2018), JK-Net (Xu et al., 2018b), GIN (Xu et al., 2018a) and GraphSAINT (Zeng et al., 2020). The first five baselines are representatives of the state-of-the-art GNN architectures. They jointly cover various message aggregation functions as well as the skip connection design for facilitating deep GNN training. GraphSAINT is the state-of-the-art minibatch training algorithm for large graphs. The GraphSAINT sampling algorithm has been shown to work well with the GraphSAGE and GAT architectures. In Table 1, for the first four methods (*i.e.*, the rows of GCN, GIN, GraphSAGE and GAT), we train 3-layer and 5-layer models in the full batch fashion. However, for the large graphs (*e.g.*, `ogbn-products`), the GPU memory (15GB) is too small to support full batch training. We therefore further run experiments using the minibatch training algorithm of GraphSAINT. Note that "GraphSAINT-RW" means the random walk sampler as suggested by the GraphSAINT paper.

For all the experiments, we fix the number of channels (*i.e.*, the hidden dimension) for each layer as $d^{(\ell)} = 256$. All accuracy results are measured by five runs without fixing random seeds. Detailed hyperparameter tuning procedure and architecture configuration are described in Appendix C.3.

Comparison with State-of-the-Art. Table 1 shows the performance comparison of SHADOW-GNN with state-of-the-art GNNs. We use two metrics: test set *accuracy* and inference *cost*. The inference cost – defined as the average amount of computation to generate prediction for one test node – is purely a measure of computation complexity and is independent of hardware / implementation factors such as parallelization strategy, batch processing, distributed storage, *etc*. Thus, the metric reflects the feasibility of deploying the particular GNN on real-world workloads, where the graph sizes and hardware specifications can be different from our setup. For SHADOW-GNN, the inference cost does not consider the sampling overhead. See Figure 2 for the sampling time measurement and Appendix B for the equations for calculating the inference cost. For the 3-layer and 5-layer models in Table 1, we simply stack the corresponding GNN layers. During training, we

further apply the DropEdge technique (Rong et al., 2020) to both the baseline and SHADOW models. We observe that DropEdge helps improve the baseline accuracy by alleviating oversmoothing, and helps improves the SHADOW accuracy due to its regularization effects. We further compare the effectiveness of skip-connection (*i.e.*, JK-Net) and shallow sampling in Table 2. .

ACCURACY Comparing GCN, GraphSAGE and GAT with their SHADOW versions, SHADOW-GNN achieves significantly higher accuracy across all the five datasets. Since such comparison is under the identical backbone GNN architectures, the accuracy gains validate the effectiveness of our design principle of "deep GNN, shallow sampler". Note that that our subgraphs are truly shallow, since the the subgraphs generated by our 2-hop and PPR samplers contain no more than 200 nodes. Comparing the accuracy of the 3-layer normal GNN and the 3-layer SHADOW-GNN, we can clearly see that a shallow neighborhood contains sufficient information to generate a good node embedding. Comparing the accuracy of the 5-layer normal GNN and the 5-layer SHADOW-GNN, we observe even higher accuracy gains – for normal GNNs, increasing from 3 layers to 5 layers sometimes results in accuracy degradation (even with DropEdge). For SHADOW-GNN, making the GNN deeper is beneficial most of the time. These observations validates our oversmoothing and expressivity analysis in Section 3.1. Finally, the higher accuracy achieved by the PPR sampler compared with the 2-hop sampler demonstrates the importance of a good sampling algorithm.

INFERENCE COST From Table 1, the inference cost of SHADOW-GNN is often orders of magnitude lower than that of the baselines. The high cost of the baselines reflects the "neighbor explosion" challenge for deep GNNs. In short, for normal GNN, since we do not enforce a fixed size subgraph, the inference computation complexity may grow exponentially with the depth of the GNN (Chen et al., 2017). For example, the number of ℓ-hop neighbors may be $10\times$ the number of $(\ell-1)$-hop neighbors. For SHADOW-GNN, it is clear that the inference cost only grows linearly with the GNN depth. From Table 1, we observe that even when the GNN depth is only 5, the high computation cost already makes the inference of normal GNNs practically infeasible. The results show the high efficiency of our design. Finally, note that for GraphSAINT, its sampling is only applied during training. The inference computation of GraphSAINT falls back to be identical as its backbone architecture (which samples the full L-hop neighborhood). On the other hand, for SHADOW-GNN, we apply the same sampling and batching strategy, whether it is for training or for inference.

Table 1: Comparison with state-of-the-art on test accuracy and inference cost (tuned with DropEdge)

Method	Layers	Flickr		Reddit		Yelp		ogbn-arxiv		ogbn-products	
		Accuracy	Cost	Accuracy	Cost	F1-micro	Cost	Accuracy	Cost	Accuracy	Cost
GCN	3	0.516±0.002	2E0	0.953±0.000	6E1	0.402±0.002	2E1	0.717±0.003	1E0	0.756±0.002	5E0
	5	0.522±0.002	2E2	0.949±0.001	1E3	OOM	1E3	0.719±0.002	2E0	OOM	9E2
GraphSAGE	3	0.514±0.001	5E0	0.965±0.000	9E1	0.617±0.003	3E1	0.719±0.003	1E0	0.785±0.001	8E0
	5	0.515±0.005	3E2	0.962±0.000	2E3	OOM	3E3	0.719±0.004	3E0	OOM	2E3
GAT	3	0.507±0.003	3E1	OOM	5E2	OOM	3E2	0.720±0.001	1E0	OOM	6E1
	5	0.516±0.003	4E2	OOM	3E3	OOM	4E3	OOM	5E0	OOM	4E3
GraphSAGE + GraphSAINT-RW	3	0.517±0.003	5E0	0.967±0.000	9E1	0.645±0.001	3E1	0.710±0.000	1E0	0.792±0.002	8E0
	5	0.520±0.003	3E2	0.967±0.001	2E3	0.639±0.000	3E3	0.701±0.002	3E0	0.796±0.002	2E3
GAT + GraphSAINT-RW	3	0.522±0.005	3E1	0.967±0.000	5E2	0.645±0.000	3E2	0.697±0.000	1E0	0.802±0.003	6E1
	5	0.515±0.003	4E2	0.965±0.002	3E3	0.647±0.001	4E3	0.695±0.001	5E0	0.799±0.007	4E3
SHADOW-GCN +PPR	3	0.526±0.002	(1)	**0.958**±0.000	(1)	0.526±0.001	(1)	0.719±0.002	(1)	0.777±0.003	(1)
	5	**0.527**±0.002	1E0	**0.958**±0.000	1E0	**0.527**±0.001	2E0	**0.721**±0.003	2E0	**0.784**±0.003	2E0
SHADOW-SAGE + 2-hop	3	0.529±0.001	2E0	0.966±0.000	2E0	0.649±0.000	2E0	0.716±0.001	2E0	0.799±0.001	2E0
	5	0.534±0.004	3E0	0.966±0.000	3E0	0.650±0.000	3E0	0.718±0.001	3E0	0.801±0.002	3E0
SHADOW-SAGE + PPR	3	0.534±0.002	2E0	**0.969**±0.000	2E0	**0.651**±0.000	2E0	0.723±0.003	3E0	0.794±0.002	2E0
	5	**0.542**±0.002	3E0	**0.969**±0.000	3E0	0.650±0.000	3E0	**0.725**±0.001	3E0	**0.804**±0.002	3E0
SHADOW-GAT + PPR	3	**0.538**±0.003	2E0	0.970±0.001	2E0	**0.655**±0.000	2E0	0.724±0.001	3E0	0.801±0.001	2E0
	5	0.534±0.002	3E0	**0.971**±0.001	3E0	0.654±0.000	3E0	**0.728**±0.003	5E0	**0.809**±0.003	3E0

Evaluation on Sampler. We further evaluate the time overhead of sampling on real machines and how the sampling sizes affect the accuracy-execution time tradeoff. In Figure 2, we parallelize the PPR sampler (implemented by `C++` and `OpenMP`) on a 40-core Xeon CPU and the GNN inference computation (implemented by `PyTorch`) on a NVIDIA P100 GPU. Clearly, the sampler is lightweight: the sampling time is lower than the GNN computation time in most cases. Also, the sampling time per node does not necessarily grow with the full graph size. As described in Section 3.2, the approximate PPR computation (Andersen et al., 2006) achieves efficiency and scalability by only traversing a local region around each target node. Note that the CPU executing the sam-

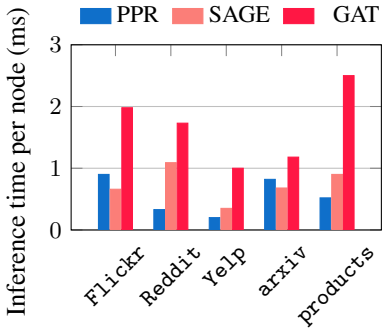

Figure 2: Inference time after parallelization on commodity hardware

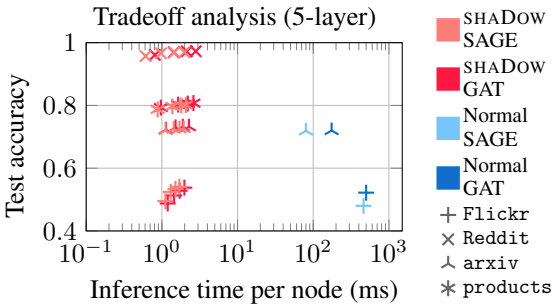

Figure 3: Inference performance tradeoff. We test pre-trained models by subgraphs of various sizes.

pler is much less powerful than the GPU. We can thus further reduce the sampling time by trivial parallelization on a more powerful machine. To understand how the sizes of the subgraphs affect the tradeoff between accuracy and inference time, we take the 5-layer models of Table 1 as the pre-trained models. Then for SHADOW-GNN, we vary the PPR sampling size p from 50 to 200 with stride 50. In Figure 3[1], the inference time of SHADOW-GNN has already included the PPR sampling time. Firstly, consistent with Table 1, inference of SHADOW-GNNs achieves higher accuracy than the normal GNNs, with orders of magnitude speedup in time. In addition, based on the application requirements (*e.g.*, latency constraint), SHADOW-GNNs have the flexibility of adjusting the sampling size without the need of retraining. For `Reddit` and `ogbn-arxiv`, reducing the subgraph size from 200 to 50 speeds up inference by $2\times$ to $4\times$ at the cost of less than 1% accuracy drop.

Evaluation on Other GNN Architectures. The idea of using shallow subgraphs on deep GNNs can potentially benefit a wide range of GNN architectures. In addition to the GCN, GraphSAGE and GAT models evaluated in Table 1, we further compare JK-Net and GIN with their SHADOW versions in Table 2. Similar to DropEdge, the skip connection (or "jumping knowledge") helps accuracy improvement of deeper models. Compared with the normal JK-Net, we observe that increasing the model depth benefits SHADOW-JK more. The GIN architecture theoretically does not incur oversmoothing. However, we observe that the training of GIN is very sensitive to hyperparameter settings. Since GIN aggregates neighbors by "sum" operation rather than "mean", we hypothesize that it is more sensitive to noisy neighbors[2]. The accuracy improvement of SHADOW-GIN compared with the normal GIN may thus due to the exclusion of noises by shallow sampling (see Section 3.1). The impact of noises can sometimes be critical, as reflected by the 5-layer GIN accuracy on `Reddit`.

Table 2: Test accuracy on other architectures

	`Flickr`		`Reddit`		`ogbn-arxiv`	
	Normal	SHADOW	Normal	SHADOW	Normal	SHADOW
JK (3)	0.495±0.007	**0.532**±0.003	0.965±0.001	**0.968**±0.000	0.714±0.003	**0.720**±0.002
JK (5)	0.494±0.008	**0.533**±0.003	0.964±0.001	**0.969**±0.000	0.716±0.005	**0.723**±0.002
GIN (3)	0.513±0.002	**0.523**±0.003	0.935±0.003	**0.958**±0.001	0.708±0.002	**0.717**±0.003
GIN (5)	0.501±0.007	**0.526**±0.002	0.755±0.003	**0.956**±0.001	0.694±0.005	**0.714**±0.002

5 CONCLUSION

We have presented a design principle, "deep GNNs, shallow samplers", for accurate and efficient GNNs. We have enhanced the learning of various state-of-the-art GNNs by constructing their SHADOW versions. We have extensively analyzed the advantage of SHADOW-GNN in terms of expressivity and proposed various sampling algorithms. We have also revealed the relationship between SHADOW-GNN and several existing models by simple architectural extensions.

[1]We do not include `Yelp` in the plot since its accuracy metric is F1-score. On `Reddit` and `ogbn-products`, the inference time for normal GraphSAGE and GAT is over 1000ms. Thus, their markers fall outside the plot.

[2]For example, a δ magnitude noise will affect the GIN aggregation by δ, but will only affect the GraphSAGE aggregation by δ/N (where N is the neighborhood size).

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. *arXiv preprint arXiv:1905.00067*, 2019.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2020.

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 475–486. IEEE, 2006.

Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 2464–2473, New York, NY, USA, 2020. Association for Computing Machinery. doi: 10.1145/3394486.3403296.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction, 2017.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

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

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *International Conference on Machine Learning*, pp. 2434–2444, 2019.

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

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling, 2020.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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, 2020.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*, pp. 4558–4567, 2018.

Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general Graph Convolutional Networks, 2020.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 538–543, 2002.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 66–74, New York, NY, USA, 2020. Association for Computing Machinery. doi: 10.1145/3394486.3403049.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019a.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning, 2019b.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All you need to train deeper gcns, 2020a.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding – design provably more powerful gnns for structural representation learning, 2020b.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into Graph Convolutional Networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Aug 2020. doi: 10.1145/3394486.3403076. URL http://dx.doi.org/10.1145/3394486.3403076.

Yu-Chen Lo, Stefano E Rensi, Wen Torng, and Russ B Altman. Machine learning in chemoinformatics and drug discovery. *Drug discovery today*, 23(8):1538–1546, 2018.

Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. In *Advances in Neural Information Processing Systems 32*, pp. 10945–10955. Curran Associates, Inc., 2019.

Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. Simple and deep graph convolutional networks. 2020.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 3697–3707, 2017.

Kenta Oono and Taiji Suzuki. Graph Neural Networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. PinnerSage: Multi-modal user embedding framework for recommendations at Pinterest. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Aug 2020. doi: 10.1145/3394486.3403280.

Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 596–606, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards deep Graph Convolutional Networks on node classification. In *International Conference on Learning Representations*, 2020.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

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

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.

Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. AM-GCN: Adaptive multi-channel graph convolutional networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 1243–1253, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403177.

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

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

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

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018b.

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

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.

Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks, 2020.

Lingxiao Zhao and Leman Akoglu. PairNorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

A PROOFS

Proof of Proposition 3.1. The GCN model performs symmetric normalization on the adjacency matrix. SHADOW-GCN follows the same way to normalize the subgraph adjacency matrix as:

$$\widetilde{\boldsymbol{A}}_{[v]} = \left(\boldsymbol{D}_{[v]} + \boldsymbol{I}\right)^{-\frac{1}{2}} \cdot \left(\boldsymbol{A}_{[v]} + \boldsymbol{I}\right) \cdot \left(\boldsymbol{D}_{[v]} + \boldsymbol{I}\right)^{-\frac{1}{2}} \tag{4}$$

where $\boldsymbol{A}_{[v]} \in \mathbb{R}^{N \times N}$ is the binary adjacency matrix for $\mathcal{G}_{[v]}$.

$\widetilde{\boldsymbol{A}}_{[v]}$ is a real symmetric matrix and has the largest eigenvalue of 1. Since SAMPLE ensures the subgraph $\mathcal{G}_{[v]}$ is connected, so the multiplicity of the largest eigenvalue is 1. By Theorem 1 of Huang et al. (2020), we can bound the eigenvalues λ_i by $1 = \lambda_1 > \lambda_2 > \ldots > \lambda_N > -1$.

Performing eigen-decomposition on $\widetilde{\boldsymbol{A}}_{[v]}$, we have

$$\widetilde{\boldsymbol{A}}_{[v]} = \boldsymbol{E}_{[v]} \boldsymbol{\Lambda} \boldsymbol{E}_{[v]}^{-1} = \boldsymbol{E}_{[v]} \boldsymbol{\Lambda} \boldsymbol{E}_{[v]}^{\mathsf{T}} \tag{5}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix $\Lambda_{i,i} = \lambda_i$ and matrix $\boldsymbol{E}_{[v]}$ consists of the N normalized eigenvectors.

We have:

$$\widetilde{\boldsymbol{A}}_{[v]}^{L} = \boldsymbol{E}_{[v]} \boldsymbol{\Lambda}^{L} \boldsymbol{E}_{[v]}^{\mathsf{T}} \tag{6}$$

Since $|\lambda_i| < 1$ when $i \neq 1$, we have $\lim_{L \to \infty} \widetilde{\boldsymbol{A}}_{[v]}^{L} = \boldsymbol{e}_{[v]} \boldsymbol{e}_{[v]}^{\mathsf{T}}$, where $\boldsymbol{e}_v$ is the eigenvector corresponding to λ_1. It is easy to see that $\left[\boldsymbol{e}_{[v]}\right]_u \propto \sqrt{\delta_{[v]}(u)}$ (Huang et al., 2020). After normalization, $\left[\boldsymbol{e}_{[v]}\right]_u = \sqrt{\frac{\delta_{[v]}(u)}{\sum_{w \in \mathcal{V}_{[v]}} \delta_{[v]}(w)}}$.

It directly follows that $\boldsymbol{m}_{[v]} = \left[e_{[v]}\right]_v \cdot \left(\boldsymbol{e}_{[v]}^{\mathsf{T}} \boldsymbol{X}_{[v]}\right)$, with value of $\boldsymbol{e}_{[v]}$ defined above. □

Proof of Theorem 3.2. We first prove the case of $\overline{\boldsymbol{m}}_{[v]} = \boldsymbol{m}_{[v]}$. *i.e.*, $\phi_{\mathcal{G}}(v) = 1$.

According to Proposition 3.1, the aggregation for each target node equals $\boldsymbol{m}_{[v]} = \left[e_{[v]}\right]_v \boldsymbol{e}_{[v]}^{\mathsf{T}} \boldsymbol{X}_{[v]}$. Let $N = |\mathcal{V}|$. Let $\bar{\bar{\boldsymbol{e}}}_{[v]} \in \mathbb{R}^{N \times 1}$ be the vector expanded from $\boldsymbol{e}_{[v]}$, such that $\left[\bar{\bar{e}}_{[v]}\right]_u = 0$ if $u \notin \mathcal{V}_{[v]}$ and the rest of the non-zero elements of $\bar{\bar{\boldsymbol{e}}}_{[v]}$ are copied from $\boldsymbol{e}_{[v]}$ accordingly.

We define:

$$\boldsymbol{\epsilon} = \left[e_{[v]}\right]_v \cdot \bar{\bar{\boldsymbol{e}}}_{[v]}^{\mathsf{T}} - \left[e_{[u]}\right]_u \cdot \bar{\bar{\boldsymbol{e}}}_{[u]}^{\mathsf{T}} \tag{7}$$

Suppose we only compute $\boldsymbol{\epsilon}$ when two nodes u and v have non-identical sampled neighborhood. i.e., $\mathcal{V}_{[u]} \neq \mathcal{V}_{[v]}$. Then given $\mathcal{G}$, SAMPLE and some distribution to generate $\boldsymbol{X}$, there are finite number of possible $\boldsymbol{\epsilon}$ values. The reason is that $\mathcal{G}$ is finite and $\boldsymbol{\epsilon}$ does not depend on $\boldsymbol{X}$ (where $\boldsymbol{X}$ itself can take infinitely many values). Each of the possible $\boldsymbol{\epsilon}$ values defines a hyperplane in $\mathbb{R}^{N-1}$ by $\boldsymbol{\epsilon} \cdot \boldsymbol{x} = \boldsymbol{0}$ (where $\boldsymbol{x} \in \mathbb{R}^N$). Let all such hyperplanes be $\mathcal{H}$.

Now suppose we build SHADOW-GCN by performing SAMPLE on $\mathcal{G}$ and $\boldsymbol{X}$. For any two v and u with $\mathcal{V}_{[v]} \neq \mathcal{V}_{[u]}$, suppose their aggregations are the same: $\boldsymbol{m}_{[v]} = \boldsymbol{m}_{[u]}$. Then we must have $\boldsymbol{\epsilon} \cdot \boldsymbol{X} = \boldsymbol{0}$. In other words, $\forall i,\ \boldsymbol{X}_{:,i}$ must fall on one of the hyperplanes in $\mathcal{H}$.

However, since $\boldsymbol{X}$ is generated from a continuous distribution in $\mathbb{R}^{N \times f}$, $\boldsymbol{X}_{:,i}$ would not fall on all of the hyperplanes in $\mathcal{H}$, almost surely. Therefore, for any v and u such that $\mathcal{V}_{[v]} \neq \mathcal{V}_{[u]}$, $\boldsymbol{m}_{[v]} \neq \boldsymbol{m}_{[u]}$ almost surely.

For a more general $\phi_{\mathcal{G}}(v)$, since $\phi_{\mathcal{G}}$ does not depend on $\boldsymbol{X}$, the proof follows exactly the same steps as above.

□

Proof of Corollary 3.2.1. Note that $\forall u \in \mathcal{V},\ u \in \mathcal{V}_{[u]}$. For any node v, there are at most $n-1$ other nodes in $\mathcal{V}$ with the same neighborhood as $\mathcal{V}_{[v]}$. Such $n-1$ possible nodes are exactly those in $\mathcal{V}_{[v]}$.

By Theorem 3.2, $\forall v \in \mathcal{V}$, there are at most $n-1$ other nodes in $\mathcal{V}$ having the same aggregation as $\boldsymbol{m}_{[v]}$. Equivalently, total number of possible aggregations is at least $\lceil |\mathcal{V}| / n \rceil$. □

Proof of Corollary 3.2.2. By definition of $\texttt{SAMPLE}_2$, any pair of nodes have non-identical neighborhood. By Theorem 3.2, any pair of nodes have non-identical aggregation. Equivalently, all nodes have different aggregation and $|\mathcal{M}| = |\mathcal{V}|$. □

Proof of Theorem 3.3. Define $\mathcal{G}_{[v]}^{L}$ as the subgraph induced from all the ℓ-hop neighbors of v, where $1 \leq \ell \leq L$.

We first prove that an L'-layer GNN is at least as expressive as an L-layer GNN on any L-hop subgraph. We note that for the target node v, the only difference between these two architectures is that an L-layer GNN exactly performs L message passing iterations to propagate node information from at most L hops away, while an L'-layer GNN has $L'-L$ more message passing iterations before performing the L message passings. Thanks to the universal approximation theorem (Hornik et al., 1989), we can let $f_1^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \sum_{u \sim v} f_2^{(\ell)}\left(\boldsymbol{h}_v^{(\ell-1)}, \boldsymbol{h}_u^{(\ell-1)}\right)\right) = \boldsymbol{h}_v^{(\ell-1)}, \forall 1 \leq \ell \leq L'-L$ since MLPs can model and learn such functions. Then, the L'-layer GNN will have the same output as the L-layer GNN.

Then, we show that an L'-layer GNN can learn something that an L-layer GNN cannot learn on some $\mathcal{G}_{[v]}^{L}$. This can be proved exactly by the example given in Figure 1, where the red edge is included in the subgraph. In this $\mathcal{G}_{[v']}^{2}$ around v', a 2-layer GNN fails to capture the red edge, and will output the representation of v' the same as when there is no red edge. In contrast, a GNN with more than 2 layers can capture the red edge immediately after the first message passing round, where the two end nodes of the red edge receive messages from each other. And after 3 message passing rounds, this captured additional information will be propagated to v', resulting in a representation different from that learned by the 2-layer GNN.

Finally, we prove that with injective f_1 and f_2, the GNN model described above (using $L' > L$ layers on the L-hop subgraph) is strictly more discriminative than the 1-dimensional Weisfeiler-Lehman (1-WL) test. We first prove that no matter how many iterations L 1-WL runs, we can always run this GNN model on the L-hop subgraph with L' layers to get at least the same discriminative power as 1-WL. According to Xu et al. (2018a), an L-layer GNN on $\mathcal{G}_{[v]}^{L}$ is theoretically as discriminative as 1-WL by being able to discriminate all nodes v that 1-WL with L iterations of color update can discriminate. Suppose there are two node v, v' that 1-WL gives different colors to after L iterations. Then applying the L-layer GNN on $\mathcal{G}_{[v]}^{L}$ and $\mathcal{G}_{[v']}^{L}$ will also return $\boldsymbol{h}_v^{(L)} \neq \boldsymbol{h}_{v'}^{(L)}$. Since f_1 and f_2 are injective, we know $\boldsymbol{h}_v^{(L+i)} \neq \boldsymbol{h}_{v'}^{(L+i)}$ for any $i > 0$, which means an L'-layer GNN ($L' > L$) can also discriminate the two nodes.

Then it suffices to give an example where 1-WL always fails to distinguish two nodes no matter how many iterations it runs, while the GNN model above can discriminate them with some L and L'. We construct a very simple example, as shown in Figure 4. Consider an undirected graph with two connected components (CC). The first CC is a hexagon. The second CC is a triangle. Both CC are 2-regular. Suppose all nodes have identical attributes and edges are unweighted. Our goal is to discriminate a node u in the first CC from a node v in the second CC. For 1-WL, no matter how many iterations L it runs, 1-WL always assigns the same label to u and v. For the GNN above, when we set $L = 1$ and $L' = 2$, we can already discriminate between the two nodes.

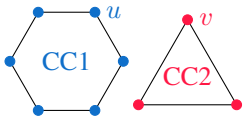

Figure 4: Example graph on which SHADOW-GNN is more expressive than 1-WL

□

B INFERENCE COMPLEXITY CALCULATION

Here we describe the equations to compute the "inference cost" of Table 1. Recall that inference cost is a measure of computation complexity to generate node embeddings for a given GNN architecture.

The numbers in Table 1 shows on average, how many arithmetic operations is required to generate the embedding for each node. For a GNN layer ℓ, denote the number of input nodes as $n^{(\ell-1)}$ and the number of output nodes as $n^{(\ell)}$. Denote the number of edges connecting the input and output nodes as $m^{(\ell)}$. Recall that we use $d^{(\ell)}$ to denote the number of channels, or, the hidden dimension.

In the following, we ignore the computation cost of non-linear activation, batch-normalization and applying bias, since their complexity is negligible compared with the other operations.

For the GCN architecture, each layer mainly performs two operations: aggregation of the neighbor features and linear transformation by the layer weights. So the number of multiplication-addition (MAC) operations of a layer equals:

$$C_{\text{GCN}}^{(\ell)} = m^{(\ell)} d^{(\ell-1)} + n^{(\ell)} d^{(\ell-1)} d^{(\ell)} \tag{8}$$

Similarly, for GraphSAGE architecture, the number of MAC operations equals:

$$C_{\text{SAGE}}^{(\ell)} = m^{(\ell)} d^{(\ell-1)} + 2 \cdot n^{(\ell)} d^{(\ell-1)} d^{(\ell)} \tag{9}$$

where the 2 factor is due to the weight transformation on the self-features.

For GAT, suppose the number of attention heads is t. Then the layer contains t weight matrices $\boldsymbol{W}^i$, each of shape $d^{(\ell-1)} \times \frac{d^{(\ell)}}{t}$. We first transform each of the $n^{(\ell-1)}$ nodes by $\boldsymbol{W}^i$. Then for each edge (u, v) connecting the layer input u to the layer output v, we obtain its edge weight (*i.e.*, a scalar) by computing the dot product between u's, v's transformed feature vectors and the model attention weight vector. After obtaining the edge weight, the remaining computation is to aggregate the $n^{(\ell-1)}$ features into the $n^{(\ell)}$ nodes. The final output is obtained by concatenating the features of different heads. The number of MAC operations equals:

$$C_{\text{GAT}}^{(\ell)} = t \cdot n^{(\ell-1)} d^{(\ell-1)} \frac{d^{(\ell)}}{t} + 2t \cdot m^{(\ell)} \frac{d^{(\ell)}}{t} + m^{(\ell)} d^{(\ell)} \tag{10}$$

$$= 3m^{(\ell)} d^{(\ell)} + n^{(\ell-1)} d^{(\ell-1)} d^{(\ell)} \tag{11}$$

On the same graph, GCN is less expensive than GraphSAGE. GraphSAGE is in general less expensive than GAT, due to $n^{(\ell-1)} > n^{(\ell)}$. In addition, we note that for all architectures, $C_*^{(\ell)}$ grows proportionally with $n^{(\ell)}$ and $m^{(\ell)}$. For the normal GNN architecture, since we are using the full ℓ-hop neighborhood for each node, the value of $n^{(\ell)}$ and $m^{(\ell)}$ may grow exponentially with ℓ. This is the "neighbor explosion" phenonemon and is the root cause of the high inference cost of the Table 1 baselines.

For SHADOW-GNN, suppose the subgraph contains n nodes and m edges. Then $n^{(\ell)} = n$ and $m^{(\ell)} = m$. The inference cost of any SHADOW-GNN is ensured to only grow linearly with the depth of the GNN.

C Detailed Experimental Setup

C.1 Additional Dataset Details

The statistics for the five benchmark graphs is listed in Table 3. Note that for `Yelp`, each node may have multiple labels, and thus we follow the original paper (Zeng et al., 2020) to report its F1-micro score. For all the other graphs, a node is only associated with a single label, and so we report accuracy. Note that for `Reddit` and `Flickr`, other papers (Hamilton et al., 2017a; Zeng et al., 2020) also report F1-micro score as the metric. However, since each node only has a single label, "F1-micro score" is exactly the same as "accuracy".

Table 3: Dataset statistics

Dataset	Setting	Nodes	Edges	Degree	Feature	Classes	Train / Val / Test
`Flickr`	Inductive	89,250	899,756	10	500	7	0.50 / 0.25 / 0.25
`Reddit`	Inductive	232,965	11,606,919	50	602	41	0.66 / 0.10 / 0.24
`Yelp`	Inductive	716,847	6,977,410	10	300	100	0.75 / 0.10 / 0.15
`ogbn-arxiv`	Transductive	169,343	1,166,243	7	128	40	0.54 / 0.18 / 0.29
`ogbn-products`	Transductive	2,449,029	61,859,140	25	100	47	0.10 / 0.02 / 0.88

C.2 Hardware Specification and Environment

We run our experiments on a DGX1 machine with Dual 20-core Intel Xeon CPUs (E5-2698 v4 @ 2.2Ghz) and eight NVIDIA Tesla P100 GPUs (15GB of HBM2 memory). The main memory is 512GB DDR4 memory. The code is written in `Python 3.6.8` (where the sampling part is written with `C++` parallelized by `OpenMP`, and the interface between `C++` and `Python` is via `PyBind11`). We use `PyTorch 1.4.0` on `CUDA 10.2` with `CUDNN 7.2.1` to train the model on GPU.

When measuring the execution time of Figure 2 and 3, we use the CPU (40-cores in total) to sample subgraphs in parallel and the GPU to execute the GNN layer operations.

C.3 Hyperparameter Searching Procedure

For all the experiments, as described in the "setup" paragraph of Section 4, we set the hidden dimension to be always $d^{(\ell)} = 256$. In addition, for all the GAT and SHADOW-GAT experiments, we set the number of attention heads to be $t = 4$. For all the GIN and SHADOW-GIN experiments, we use a 2-layer MLP (with hidden dimension 256) to perform the injective mapping in each GIN layer. For all the JK-Net and SHADOW-JK experiments, we use the concatenation operation to aggregate the hidden features of each layer in the JK layer.

For all the baseline and SHADOW-GNN experiments, we use Adam optimizer (Kingma & Ba, 2014). We perform grid search on the hyperparameter space defined by:

- Activation function: $\{\texttt{ReLU}, \texttt{ELU}\}$
- Dropout: $\{0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$
- Dropout: $\{0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$
- Learning rate: $\{0.01, 0.002, 0.001, 0.0002, 0.0001, 0.00002\}$

The sampling hyperparameters are tuned as follows.

For the PPR sampler, we consider two versions: one based on fixed sampling budget p and the other based on PPR score thresholding θ.

- If with fixed budget, we then we disable the θ thresholding. We tune the budget by $\{100, 125, 150, 175, 200\}$.
- If with thresholding, we set $\theta \in \{0.01, 0.05\}$. We still have an upper bound p on the subgraph size. So if there are q nodes in the neighborhood with PPR score larger than θ, the final subgraph size would be $\max\{p, q\}$. Such an upper bound eliminates the corner cases which may cause hardware inefficiency due to very large q. In this case, we set p to be either 200 or 500 depending on the graph.

For the k-hop sampler, we define the hyperparameter space as:

- Depth $k = 2$
- Budget $b \in \{5, 10, 15, 20\}$

The hyperparameters to reproduce the Table 1 SHADOW-GNN results are listed in Tables 4 and 5. The code will be shared with the reviewers at the end of the rebuttal period and released to public after paper acceptance.

Table 4: Training configuration of SHADOW-GNN for Table 1 (PPR sampler)

Arch.	Dataset	Layers	Learning Rate	Dropout	DropEdge	Budget p	Threshold θ
SHADOW-GCN	Flickr	3	0.001	0.40	0.10	200	–
		5	0.001	0.40	0.10	200	–
	Reddit	3	0.00002	0.20	0.05	150	–
		5	0.00002	0.20	0.05	150	–
	Yelp	3	0.001	0.10	0.00	100	–
		5	0.001	0.10	0.00	100	–
	ogbn-arxiv	3	0.00002	0.25	0.10	200	–
		5	0.00002	0.25	0.10	200	–
	ogbn-products	3	0.002	0.40	0.05	150	–
		5	0.002	0.40	0.05	150	–
SHADOW-SAGE	Flickr	3	0.001	0.40	0.00	200	–
		5	0.001	0.40	0.00	200	–
	Reddit	3	0.0002	0.20	0.10	150	–
		5	0.0002	0.20	0.10	150	–
	Yelp	3	0.001	0.10	0.00	200	–
		5	0.001	0.10	0.00	200	–
	ogbn-arxiv	3	0.0001	0.30	0.10	500	0.01
		5	0.00002	0.25	0.15	200	0.01
	ogbn-products	3	0.002	0.40	0.15	150	–
		5	0.002	0.40	0.15	150	–
SHADOW-GAT	Flickr	3	0.001	0.40	0.00	200	–
		5	0.001	0.40	0.00	200	–
	Reddit	3	0.0001	0.20	0.00	150	–
		5	0.0001	0.20	0.00	150	–
	Yelp	3	0.001	0.10	0.00	100	–
		5	0.001	0.10	0.00	100	–
	ogbn-arxiv	3	0.0001	0.20	0.00	175	–
		5	0.0001	0.20	0.00	175	–
	ogbn-products	3	0.001	0.35	0.00	150	–
		5	0.001	0.35	0.00	150	–

Table 5: Training configuration of SHADOW-GNN for Table 1 (k-hop sampler)

Arch.	Dataset	Layers	Learning Rate	Dropout	DropEdge	Budget b	Depth k
SHADOW-SAGE	Flickr	3	0.001	0.40	0.00	20	2
		5	0.001	0.40	0.00	20	2
	Reddit	3	0.0001	0.20	0.00	15	2
		5	0.0001	0.20	0.00	15	2
	Yelp	3	0.01	0.10	0.00	5	2
		5	0.01	0.10	0.00	5	2
	ogbn-arxiv	3	0.0001	0.20	0.00	10	2
		5	0.0001	0.20	0.00	10	2
	ogbn-products	3	0.002	0.35	0.00	10	2
		5	0.002	0.35	0.00	10	2

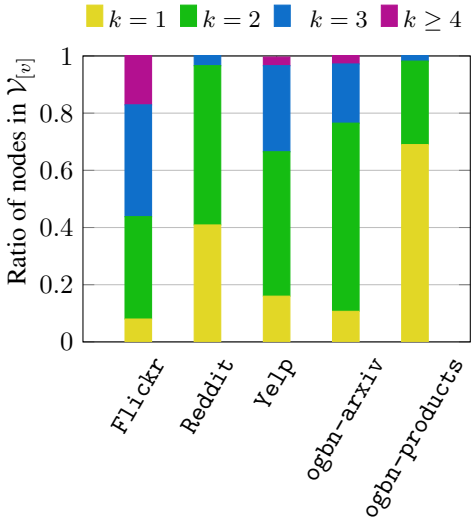

Figure 5: PPR sampler for 5-layer SHADOW-GNN: Composition of nodes u in $\mathcal{V}_{[v]}$ under various $\texttt{dist}\,(u, v) = k$

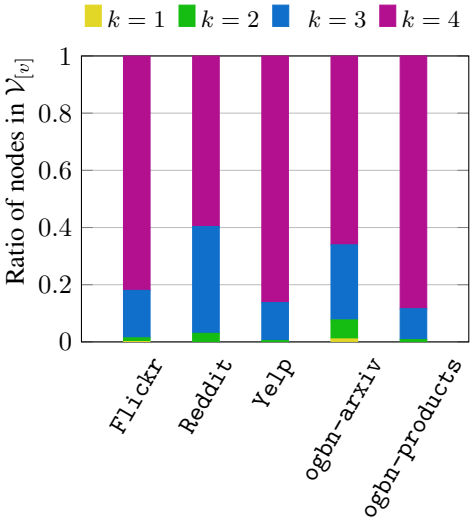

Figure 6: 4-hop sampler for normal 4-layer GNN: Composition of nodes u in $\mathcal{V}_{[v]}$ under various $\texttt{dist}\,(u, v) = k$

C.4 Additional Experimental Results

Is the PPR sampler really "shallow"? The PPR sampler defines the neighborhood $\mathcal{V}_{[v]}$ based on the magnitude of the PPR scores $\widetilde{\pi}_v$. Although it is possible that a node far away from the target v may have a high PPR score and be selected by SAMPLE, such a case rarely happens in practice. Most of the nodes in $\mathcal{V}_{[v]}$ are concentrated within the 2-hop neighborhood of v. Let $\texttt{dist}\,(v, u)$ denote the shortest path distance between nodes u and v. Figure 5 shows the distribution of $\texttt{dist}\,(v, u)$ for $u \in \mathcal{V}_{[v]}$.

On the other hand, Figure 6 shows the composition of the neighborhood under the normal GNN. It is clear that most of the nodes propagated by the normal GNN are far away from the target node.

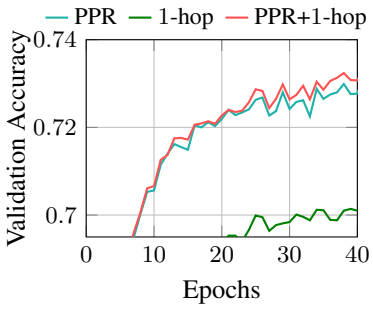

Figure 7: Subgraph ensemble: SHADOW-SAGE on ogbn-arxiv