Size Does (Not) Matter? Sparsification and Graph Neural Network Sampling for Large-scale Graphs

Jana Vatter
Technical University of Munich
Munich, Germany
jana.vatter@tum.de

Maurice L. Rochau
maurice.rochau@outlook.com

Ruben Mayer
University of Bayreuth
Bayreuth, Germany
ruben.mayer@uni-bayreuth.de

Hans-Arno Jacobsen
University of Toronto
Toronto, Canada
jacobsen@eecg.toronto.edu

ABSTRACT

With the ever-growing size of real-world graphs, optimizing Graph Neural Network (GNN) training has become essential. Two key methods for this are graph sparsification and GNN sampling, both aim at reducing graph size while preserving valuable information. The question arises what kind of information should be preserved and what a reasonable graph size is. We propose combining random graph sparsification with GNN sampling, showing that this approach can significantly reduce training time while maintaining accuracy. Our experiments demonstrate that sparsification to around 40% of the original graph and sampling with a fanout parameter of 4 yields the best results in terms of training time and accuracy. Beyond training time, also inference time can be decreased up to 75% which enables scalability for time-critical applications such as fraud detection. Finally, we identify open challenges and new research directions, including sampling-aware graph reduction methods, mining new graph datasets, and the prevention of bias.

VLDB Workshop Reference Format:
Jana Vatter, Maurice L. Rochau, Ruben Mayer, and Hans-Arno Jacobsen. Size Does (Not) Matter? Sparsification and Graph Neural Network Sampling for Large-scale Graphs. VLDB 2024 Workshop: 3rd International Workshop on Large-Scale Graph Data Analytics (LSGDA 2024).

1 INTRODUCTION

Real-world graphs are rapidly growing with billions of nodes and edges [17, 20]. They can be found all around us in various domains including social networks, citation graphs, and co-purchasing networks [21, 29]. Graph Neural Networks (GNNs) have gained greater attention throughout the past years and are used to learn and make predictions on those graphs. However, with the ever-growing size of real-world graphs, the need for optimization techniques has emerged to allow for efficient and scalable training. Numerous methods have been developed, for instance, graph partitioning [24, 26], model simplification [37, 46] and compression [11, 35]. Resource-efficient training can drastically reduce costs in the case of large graphs. If inference needs to be repeatedly calculated or is time-critical, such as in fraud detection [34], spam identification [45, 47] or recommender systems [14, 43, 53], it is important to allow for a fast inference step. Two optimization methods which train on a subgraph instead of the full graph are graph reduction [19] and sampling [32, 42]. Graph reduction first reduces the size of the graph and then performs graph algorithms such as GNN training. Sampling, on the other hand, draws new subgraphs at each training step. In this way, different nodes and edges are included throughout the whole training process, leading to a stable accuracy. The assumption of both methods is that in case of large-scale graphs, not all nodes and edges are needed to obtain satisfying performance. The question arises how small these graphs can be and how large they need to be when training and making predictions with GNNs. What kind of information should be preserved when reducing the graph? Can we combine the two optimization methods graph reduction and sampling-based GNN training? Do we get a better performance by combining both methods, or do we lose too much information?

We explore these questions against the background of the Law of Large Numbers (LLN). LLN states that if a sufficient number of independent observations have been collected, the mean of these observations will converge around (weak LLN) or on the true value (strong LLN). We theorize, alongside the lines of graph reduction and sampling, that for satisfactory GNN results, large-scale graphs are not necessary.

This work investigates graph reduction combined with graph sampling during GNN training. We systematically analyze existing methods for graph reduction and GNN sampling as well as the underlying theory. We propose to draw random subgraphs out of large-scale graphs and perform sampling-based GNN training. In this way, we demonstrate that relevant relationships can be preserved and no additional bias is induced. We infer the findings to application domains and future research directions.

Our contributions can be summarized as follows:

- We investigate how small a graph can be and how large a graph needs to be in order to learn and predict with a GNN.
- We explore graph reduction combined with sampling during GNN training.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

- We investigate specialized as well as random graph reduction methods and show that the latter is able to preserve important relations within the graph without the need for intensive pre-processing steps.
- We highlight how graph reduction and sampling can influence bias within a graph.
- We thoroughly analyze graph reduction in combination with sampling methods to identify practical implications, open questions, and future research directions.

The paper is organized as follows. First, we give an introduction to Graph Neural Networks, Graph Reduction and Sampling during GNN training in Section 2. In Section 3, we provide a theoretical and empirical analysis. The latter presents our experimental methodology as well as corresponding results. We discuss open challenges and future research directions in Section 4. Section 5 gives an overview of related work, and Section 6 summarizes our findings.

2 BACKGROUND

A graph G can be formally denoted as $G = (V, E)$ where V is the set of vertices, also called nodes, and E the set of edges. A vertex v represents an object while an edge e models the relation between two objects. An edge e is described by $e = (a, b)$ where a and b are nodes that are connected by e. Consequently, a graph is a collection of edges and nodes, showing how different nodes relate to each other through directed and undirected edges.

2.1 Graph Neural Network Training

GNNs are a specialized machine learning technique designed to learn and predict on graph-structured data [16, 41]. Each node initially contains feature information and depending on the application, edges can also contain features. At each layer, two main steps are performed: *aggregation* and *update*. During the *aggregation* phase, each node exchanges messages with its neighbors, sharing their current representation, also called embedding. This is done using an aggregation function

$$a_v^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k-1)} | u \in N(v)\}) \tag{1}$$

where $h_u^{(k-1)} | u \in N(v)$ represents the representations from the previous layer of the neighboring vertices $N(v)$ [18]. Subsequently, each vertex combines the received representations and updates its own state. This *update* phase is formally defined as

$$h_v^{(k)} = UPDATE^{(k)}(a_v^{(k)}, h_v^{(k-1)}) \tag{2}$$

where the representation of node v of the previous layer $h_v^{(k-1)}$ is combined with the aggregated representations $a_v^{(k)}$.

When using a k-layer GNN, the k-hop neighborhood of a node is explored. According to a given loss function, the model weights are adapted, and a new epoch is started using stochastic gradient descent. As soon as a given number of epochs is performed, the final model weights can be used to compute node (and edge) representations for downstream tasks on node-, edge-, and graph-level.

2.2 Graph Reduction

There are different ways to reduce the size of a graph. The methods can be categorized into graph sparsification, graph coarsening, and graph condensation [19]. In the following, we give an overview of graph sparsification techniques. In Section 5, we give an overview of graph coarsening and graph condensation techniques since they are highly related to our approach.

Graph sparsification represents a given graph by a sparse subgraph which includes a subset of nodes and edges. While the graph is sparsified, the aim is to maintain the graph properties. Depending on the downstream task, this could be the preservation of clusters, graph connectivity, or important nodes. A simple and intuitive way to sparsify the graph is by randomly selecting a set of nodes and/or edges with an equal probability to be included [7]. More specialized techniques include setting the probabilities proportional to edge weights or pruning edges in relation to the corresponding node degree [39] to maintain high graph connectivity. Other sparsification techniques are Spanning Forest or t-Spanner [7]. Forest Fire [28] adds one edge after another starting from a given node. This is done until a threshold is reached. In their method, Feng et al. [13] first identify an extremely sparse subgraph which is then expanded iteratively. In recent years, learning-based methods have been developed. In GSGAN, a generative adversarial network (GAN) [15] is used to sparsify graphs for community detection. To preserve subgraph modularity, SparRL [52] uses deep reinforcement learning. Here, edges are sequentially pruned based on a plug-in reward function ensuring task flexibility. Another type of sparsification strategy deals with graph learning methods in combination with sparsification. Here, the objective is to maintain the performance of GNNs in relation to the prediction task. Salha et al. [40] find a subgraph according to k-core decomposition, train the graph learning model, and finally approximate nodes that are not included in the subgraph. Interpretable Graph Sparsification (IGS) [30] extracts edge importance values obtained by GNNExplainer [55] to prioritize nodes with a high importance value during the sparsification step. The graph, its features, and model parameters are dynamically pruned during GNN training by the Comprehensive Graph Gradual Pruning (CGP) method [31]. This ensures efficient training and fast inference.

Apart from random sparsification, the methods usually have certain objectives or application domains. Learning-based methods adapt to the use case, but might be very resource intensive. Usually, edges are pruned instead of nodes. The underlying assumption is that edges contain redundant information. Thus, some edges can be deleted without losing much information. Further, some methods calculate importance values based on characteristics like the node degree or the connectivity. Thus, these methods assume that some nodes or edges are more important than other ones.

2.3 Sampling during GNN training

In contrast to graph sparsification where the graph is reduced before performing graph algorithms such as GNN training, sampling constructs and uses new subgraphs at each GNN training step. When drawing new subgraphs at each training step, different nodes are included which leads to a stable accuracy. The main effects of sampling are reduced training time, less memory consumption, and faster convergence. Numerous sampling strategies have been developed and can be categorized into node-wise [6, 18], layer-wise

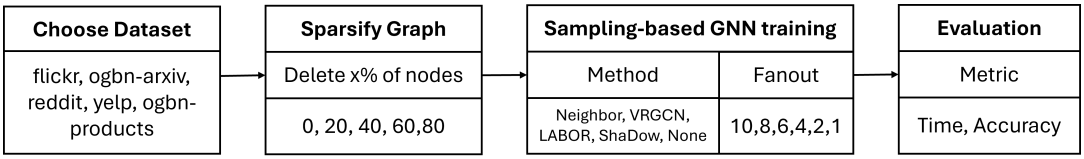


Figure 1: Overview of our experimental setup

[1, 5, 63], subgraph-based [8, 57, 58] and learning-based [56, 61] sampling.

Given a target node, k neighboring nodes are chosen at random or based on an importance score when performing node-wise sampling [6, 18]. The parameter k is called fanout and given by the user. This step is independently repeated for the subsequent layers from the previous sampled nodes to construct a subgraph. Layer-wise sampling [1, 5, 63] differs by choosing the nodes of the next layer dependent on the sampled nodes in the current layer. This leads to reduced variance compared to node-wise sampling. The third category, subgraph-based sampling [8, 57, 58], first forms subgraphs and then performs sampling within the fixed subgraphs resulting in a restricted size of the neighborhood. Recently, learning-based sampling strategies have emerged [56, 61]. They learn how to sample and adapt to the given input graph by training a separate model which then predicts which nodes and edges should be included in a sample or generates a sample. In terms of evaluation metrics like accuracy, the methods prove to work well. However, they are time- and memory-intensive due to the additional learning step.

Due to different objectives and assumptions, not all methods work equally well for all graphs. In case pre-computations before the actual sampling step are needed, like the forming of partitions or calculation of importance scores, the corresponding method is less resource efficient than random sampling. The same holds for learning-based methods. An assumption many strategies hold is that some nodes or edges contain more information and are more beneficial for training than others.

3 ANALYSIS

3.1 Theoretical Analysis

The LLN states that if a sufficient number of independent observations has been collected, the mean of these observations will converge around (weak LLN) or on the true value (strong LLN) of a population, given that such a value exists [9]. An example of that is rolling a dice: Rolling a dice many times, the mean will align on 3.5 which is the true mean of all dice sides. Stopping beforehand, the mean of the throws will, with an increasing number of dice rolls, align on 3.5.

We posit that the concept of the LLN can also be transferred to graphs and GNNs. We base this on the following assumptions.

Assumption 1: We assume that a graph is a collection of observations of relationships. In a graph, nodes and edges are collected for one or multiple domain areas. Just as any scientific experiment, this collection is a sample of a larger population. We observe a fraction of all the relationships that really exist in the target domain.

Assumption 2: We assume that relationships have a true mean. As we observe relationships, we assume that sometimes observed relationships can be wrong, rare, or misleading. As an example from the domain of academic citation graphs: If a paper is about Large Language Models, it is highly likely that recent applications of the concept as well as the original paper are cited. However, any other, related or unrelated, paper in the same or any other domain could be cited as well. We therefore theorize that edges can be thought of as being relevant and irrelevant, but most of the time it is impossible to know beforehand. As we are primarily concerned with observing relevant edges to find the true mean, we assume that a graph can be reduced randomly as long as there are still enough relevant edges to train on them.

Assumption 3: We assume that large-scale graphs have so many observations that it is permissible to leave out observations. According to the LLN, more observations will lead to closer alignment with the true mean. In a large-scale graph, we theorize that the many observations already lead to a near perfect alignment with the true mean. By removing observations, the alignment to the true mean is less strong, however, we assume that the alignment is still strong enough to approximate the true mean closely enough.

3.2 Empirical Analysis

In the following, we investigate the effect of random graph sparsification in combination with sampling. We explore how accuracy, training, and inference time change when using smaller graphs. Further, we analyze the tradeoff between graph size and accuracy along with the question of what the boundaries of minimal graph sizes or maximal graph reduction are in the case of GNN training. We give an overview of our experimental setup in Figure 1.

3.2.1 Methodology. For our experiments, we use the `ogbn-arxiv` and `ogbn-products` graph provided by the Open Graph Benchmark[1] (OGB) [21] as well as `flickr`, `reddit`, and `yelp` included in the Deep Graph Library[2] (DGL) [49]. Details about the datasets are summarized in Table 1. We use the full graph and delete x percent of the nodes ranging from 20% to 80% percent of all nodes in steps of 20%. Deleting edges mainly changes the graph structure and graph characteristics. The result is a sparse graph containing an unchanged number of nodes. On the contrary, deleting nodes makes the graph truly smaller in terms of the total number of nodes and edges which is our objective. By deleting nodes, we want to verify the hypothesis that in large-scale graphs, single nodes are not as important as graph characteristics such as the existence of clusters and relations between nodes.

In addition to sparsifying the graph by randomly deleting nodes, we perform mini-batch training with sampling. The sampling strategies include Neighbor Sampling [18], VR-GCN [6], LABOR [1] and

[1] https://ogb.stanford.edu/
[2] https://www.dgl.ai/

Table 1: Characteristics of the graph datasets (ND: node degree, CC: cluster coefficient)

Name	#Nodes	#Edges	Avg. ND	Avg. CC	#Classes	Feature Size	Prediction Task
flickr	89,250	899,756	10.1	0.033	7	500	category of image
ogbn-arxiv	169,343	1,166,243	13.7	0.226	40	128	paper subject area
reddit	232,965	114,615,892	492	0.579	50	602	community of post
yelp	716,847	13,954,819	20.5	0.092	100	300	category of business
ogbn-products	2,449,029	61,859,140	50.5	0.411	47	100	category of product

Table 2: Sparsification and inference time in relation to the end-to-end-time (in s)

Graph	Perc. of nodes	Spars. Time	Inf. Time	End2End Time
flickr	80	2.5	2.6	319.6
	20	0.8	1.0	35.7
ogbn-arxiv	80	0.9	3.9	280.8
	20	0.9	2.4	42.4
reddit	80	23.5	46.7	51,587.5
	20	6.4	11.2	3,175.2
yelp	80	18.1	38.7	8847.3
	20	18.9	38.6	8781.2
ogbn-products	80	38.4	145.3	8,122.5
	20	35.0	109.4	1,302.1

ShaDow [57]. We chose these four strategies because they all employ the same fanout parameter leading to a better and more fair comparison. Additionally, we include one or more sampling strategies each for the categories node-wise (Neighbor, VR-GCN), layer-wise (LABOR) and subgraph-based sampling (ShaDow). The fanout parameter is chosen from $\{10, 8, 6, 4, 2, 1\}$.

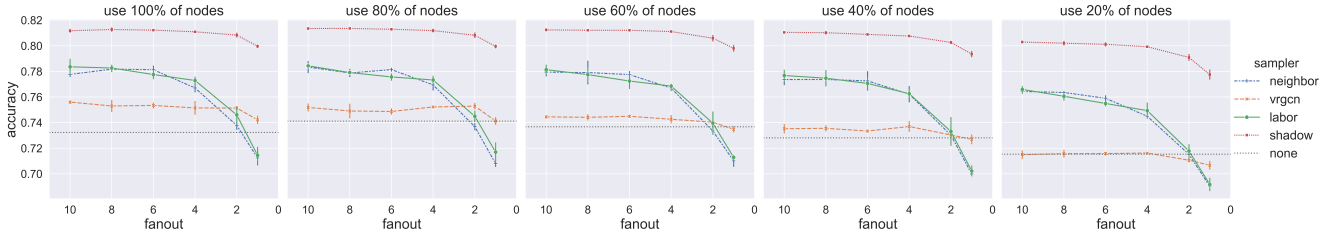
A two-layer GCN as GNN architecture with a learning rate of $1e^{-3}$ and the Adam Optimizer [25] are used. Our loss function is the Cross Entropy Loss for the node classification task. Evaluation is done based on time and accuracy. We train each model for 20 epochs and repeat each experiment 3 times. The average result along with the standard deviation are reported. This ensures robustness and reliability. Since the standard deviation is relatively small, 3 repetitions are sufficient. All experiments are implemented with DGL and PyTorch [38] and run on an Ubuntu Focal Fossa (20.04) virtual machine with 32 vCPUs and 128 GB RAM.

3.2.2 Results. We give an overview of the time it takes to construct the sparse graphs in relation to the overall training time with full-neighbor sampling in Table 2. We can see that for large graphs, such as ogbn-products, it only takes 0.005% to 0.027% of the end-to-end time to construct the graph when using 80% and 20% of nodes, respectively.

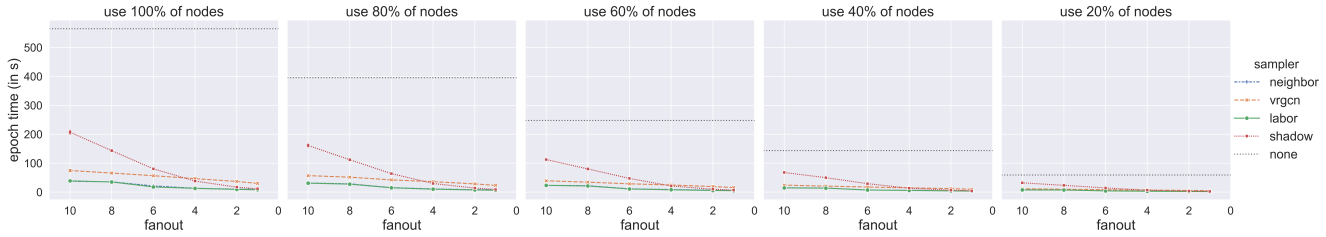
Sparsification: Figure 2 and Figure 3 give an overview of the experimental results for two representative datasets, namely `ogbn-products` and `reddit`. Overall, the accuracy remains relatively stable with smaller subgraphs and across all datasets. Starting at 40% of nodes, the accuracy decreases around 0.02 when going from 40% to 20% of nodes (`ogbn-products`). Similar observations can be made across all investigated sampling methods. Not only training can take large amounts of time and needs to be optimized, but also inference. In large-scale graphs and time-critical applications, it is crucial to be able to make fast predictions. When looking at the `ogbn-products` graph, inference time decreases by up to 75% when regarding inference time for the full graph compared to reduced by 60% of nodes. This also holds for the remaining graphs we investigate (Table 2).

Sampling: Regarding sampling, the random Neighbor sampler overall results in a good epoch time and accuracy. ShaDow sampling can come with better accuracy in some scenarios. However, the training times are longer across all graphs. With fewer nodes, VR-GCN results in a similar performance than no sampling on the `ogbn-products` graph. On the other hand, Neighbor, LABOR, and ShaDow lead to a higher accuracy compared to no sampling. At the same time, sampling-based training can be around 2.7 times faster at a fanout of 10 which illustrates the effectiveness of our method. Another observation is how the performance changes with smaller fanouts for sampling-based training. Overall, the accuracy does not show a significant decrease when reducing the fanout to 6 across all methods and datasets. Some samplers, such as VR-GCN and ShaDow, show satisfying performance even with a fanout of 2 on sparse graphs. Thus, a fanout between 6 and 2 can be used while maintaining accuracy and decreasing training time.

Sparsification and Sampling combined: An important finding is that both methods, graph sparsification and sampling, combined are able to maintain the accuracy while reducing training time significantly. Combining both outperforms using only one method standalone in regards to epoch time. However, there is a trade-off between the sparsity of the graph, the choice of sampling method, and the value of the fanout parameter. We identify a rule holding true for our experiments determining the degree of sparsification in combination with the value of the fanout parameter. Best results can be achieved with around 40% of the original graph size. This size is sufficient to only experience a marginal drop in accuracy (if any). One can also use a subgraph down to 20% of the original size if a drop in terms of accuracy around 0.01 is not critical for the application. Regardless of the chosen sampling strategy, a fanout of 4 is a good trade-off between a stable accuracy and a decrease in epoch time. We refer to the rule of using 40% of the graph size and a fanout of 4 as the *40/4*-rule.

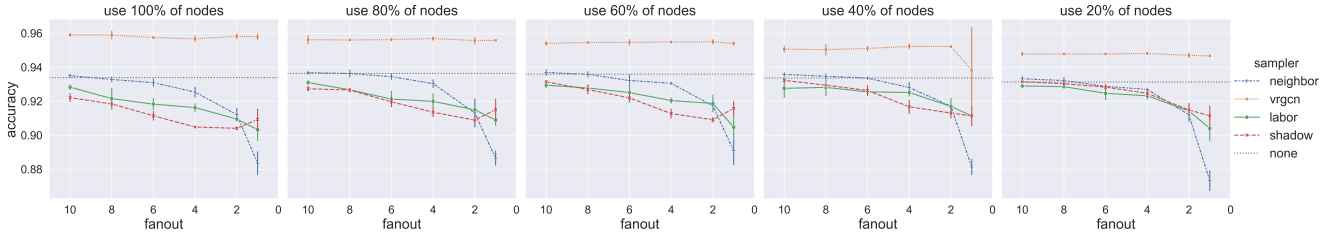


(a) Accuracy for the ogbn-products graph

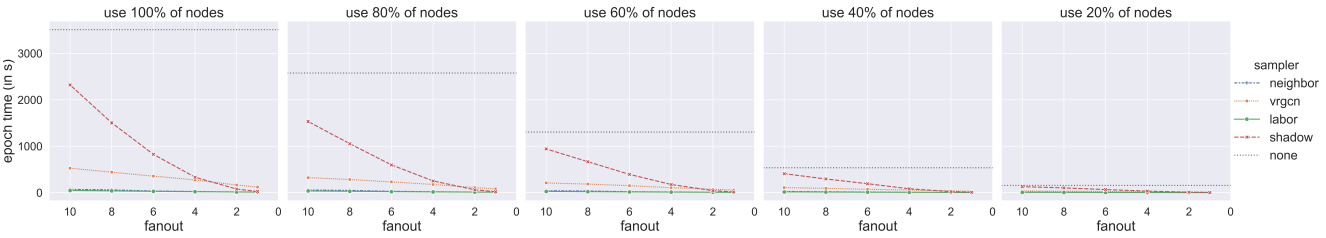


(b) Epoch time for the ogbn-products graph

Figure 2: Results for the ogbn-products graph, different samplers, fanouts, and sparsification levels



(a) Accuracy for the reddit graph



(b) Epoch time for the reddit graph

Figure 3: Results for the reddit graph, different samplers, fanouts, and sparsification levels

4 OPEN CHALLENGES

4.1 Graph sparsification and sampling

The aim of all investigated methods is to reduce the size of the graph for faster execution of a given graph algorithm or GNN training. Due to different assumptions and objectives, not every method is beneficial for all scenarios. While some methods aim to preserve certain graph characteristics, others concentrate on the node or edge features. Besides that, an overhead might be introduced due to pre-processing steps with only little improvement in terms of performance. With more and more methods available in the past years, the choice of method becomes inherently harder. The question arises in which cases these specialized methods are needed to obtain satisfying results on large-scale graphs. In which cases might a random selection be sufficient? If using sparsification and sampling methods, how small should the graph be made and which sampling parameter should be chosen? In our experiments, we show that large-scale graphs can be reduced by up to 40% of the original size without a significant decrease in accuracy. Combined

with sampling, it is even more effective leading to a better accuracy than when using the complete graph.

4.2 Applications

By reducing training time in both epoch time as well as in the absolute training time, the process of training GNNs and evaluating their performance in relation to their application is simplified and streamlined. Graphs and GNNs have multiple application domains including spam detection [45, 47], fraud detection [34] or recommender systems [14, 43, 53].

In recommender systems, one challenge that persists, as reported in multiple surveys [14, 43, 53], is scalability. With many graphs being extremely large in size, most current approaches are not scalable beyond certain points. By using sampling, the scalability of GNNs running on large-scale graphs can be improved. However, some graphs remain too large to be used efficiently. The approach we present in this study, to combine sampling with graph reduction, shows that accuracy can be kept at acceptable levels while reducing training time significantly. For recommender systems in large-scale settings, this approach can help to regularly re-train the GNNs and to deliver results more quickly. Further, selecting nodes and edges at random does not introduce (pre-)computation overhead while maintaining accuracy. This also holds for other time-critical applications, such as fraud detection [34] or spam detection [45, 47], where fast predictions are crucial and the GNN needs to be frequently re-trained.

4.3 Bias

Bias is the phenomenon that different steps in a (data) task might have inherent errors. These errors lead to so-called biased results: the results vary around the actual values one wants to observe and measure. Bias has many different forms that can be strongly or weakly expressed depending on a given experimental design. Hence, it "should be considered primarily a function of study design and execution" [44].

For the method we propose, we have purposefully not investigated bias as we are primarily concerned with designing an easily usable and scalable method to make gains in the fields of training, inference, and applications. We assume that, by applying our random method, common bias on the data and sample level are not introduced. If a dataset is reduced at random, existing bias within that dataset is also reduced at random in alignment with how often a specific bias exists in the dataset. Hence, for bias at data level, we assume that neither additional bias is introduced nor that existing bias is reduced. For instance, the number of nodes corresponding to a specific attribute is reduced to the same degree as the number of nodes of other attributes when reducing a graph at random. This also translates to the classes which are predicted where a class might be underrepresented or overrepresented.

Future research directions could look into the areas of data selection biases and model biases that might arise by having a significantly smaller dataset. In case one wants to mitigate bias introduced by random deletion of nodes, one could train a model on several graphs and combine them when calculating inference. For instance, one could form n different random subgraphs with different seeds containing x percent of the original nodes. On each subgraph, a model is trained individually. After training, the models are combined to predict on a graph. This would still be more efficient in terms of time and memory, and one could easily distribute the training process on multiple machines which could lead to a fairer and more accurate model since different nodes and edges are included in each subgraph.

4.4 Mining New Graph Datasets

With the analysis of the LLN and our empirical evaluation, the question arises what kind of data is needed for specific prediction tasks. This can be transferred to the mining of new graph datasets. Does one need to collect billions of datapoints, or is a certain amount of data sufficient for the subsequent graph learning and prediction tasks? Since mining new datapoints is resource-intensive, a rough approximation could help to reduce costs. Our experiments could help to give an estimation of how many datapoints are needed to still be able to train and predict with a GNN. In other words, sampling could already be performed while mining the dataset. To ensure a certain number of random datapoints are picked, one could use a method called Reservoir Sampling [48]. This method is used to randomly choose n samples from an unknown number of data samples or an extremely large number of data samples N. The data is streamed, and a reservoir is filled with decreasing probability for the datapoints to be included. Elements in the reservoir have a probability of $1/n$ to stay in the reservoir and not be replaced with a new one. This ensures n random samples are picked out of N candidate data points. An advantage of this streaming-based method is that it is very memory efficient.

4.5 Future Research Directions

Several future research directions arise, for instance, a broader evaluation of the *40/4*-rule, an evaluation in regards to other graph reduction methods combined with sampling, the exploration of what a reasonable size for a graph with a given application task is, as well as graph reduction methods tailored to sampling or vice versa.

The question arises what a reasonable size of a graph for a given task is. Is the graph too large, and could some nodes and edges be removed? Or is the graph too small and more information is needed to make meaningful predictions? In our experiments, around 40% of nodes and sampling with a fanout of 4 usually leads to the best results (*40/4*-rule). However, more investigation needs to be done in the direction of required graph size in relation to the underlying prediction task. In this manner, researchers could better assess how many data points need to be mined to solve a given task.

We have shown that the random graph reduction method and also random sampling usually produce a sufficient accuracy while minimizing training and inference time. Further, we assume that no additional bias is introduced. There might be scenarios where more specialized methods are needed and some kind of bias is desired or should actively be decreased. We identify a novel research direction, namely the design of a graph reduction method in combination with a sampling method. This could further improve the results for certain application tasks.

5 RELATED WORK

Leskovec and Faloutsos [27] investigate several graph reduction methods for graph processing algorithms, such as shortest paths or centrality. The authors explore which method should be chosen and what an appropriate subgraph size is. In addition, they answer the question based on which metrics these methods should be evaluated on. We also explore graph reduction and what a sufficient subgraph size is. Our work differs by focusing on GNN training instead of graph processing. Further, we combine two methods reducing the graph size, namely graph sparsification and sampling for GNN training.

Another related work [32] centers on GNN sampling. Various strategies are analyzed and categorized into node-wise, layer-wise, subgraph-based, and heterogeneous strategies. The authors compare within categories as well as across categories to emphasize the strengths and weaknesses of the methods. Serafini et al. [42] survey sampling strategies and GNN systems using sampling. Comparison is done in regards to whole-graph training and based on a fixed fanout parameter. In contrast to the two papers above, we not only use sampling, but reduce the graph and use only a subset of nodes and edges as input for GNN training and sampling. Another difference are the varying fanouts from 10 to 1 we use to explore how small the parameter can be set while maintaining a satisfactory accuracy.

Graph reduction before GNN training is explored by Wei et al. [51]. Four reduction methods and multiple GNN architectures are evaluated. We differ by not only regarding graph reduction, but also sampling-based mini-batch training. Liu et al. [33] give an overview of GNN acceleration techniques including sampling and sparsification. They highlight different methods belonging to these categories and compare them. Our work analyzes the effectiveness of combining both graph sparsification and sampling while assessing which parameters should be chosen for both.

In our work, we explore graph sparsification. More specifically, random sparsification combined with sampling during GNN training. However, there are other methods to reduce the size of a graph, such as graph coarsening and graph condensation.

When performing *graph coarsening*, certain nodes are grouped into clusters and aggregated based on given criteria. A criterion could be proximity or similarity. The objective of graph coarsening is to reduce the graph size while preserving the original structure and graph characteristics [2]. Local Variation (LV) [36] can be used to calculate pairwise scores to decide which nodes to group. SCAL uses the coarsened graph by LV to train and predict with a GNN [22]. Another method grouping nodes based on a score is CONVMATCH [10]. Here, nodes with similar characteristics in relation to the GCN convolution are merged. With the help of GNNs, Cai et al. [2] learn how to reduce the number of nodes by merging certain clusters. Due to the learning step, this method is able to adapt to various graph types and reduction ratios. The Macro Graph Neural Network (MacGNN) architecture has been developed to be used in combination with MAcro Recommendation Graphs (MAG) [4] for recommender systems. Nodes with similar behavior patterns are grouped into macro nodes to reduce the graph from billions to hundreds of nodes. This method also ensures sampling bias is minimized by sampling large enough proportions of each group. Again, all methods employ various objectives and work well in certain scenarios. However, due to underlying assumptions, computation steps, and application domains, they might not be optimal for all use cases which makes it hard to choose which one to pick.

In contrast to sparsification and coarsening, *graph condensation* first distills knowledge from a graph dataset and then utilizes this to construct a smaller, synthetic graph. The idea was introduced by Jin et al. [23] and is inspired by dataset distillation [50] and dataset condensation [60]. The gradients of the original and the synthetic graph are matched during each training step, and an objective function is minimized. To enhance the optimization process, CTRL [59] adds the magnitude distance in the optimization process.

As these methods have some inefficiencies due to the optimization step, EXGCN [12] selects and focuses on important nodes during the training process instead of all nodes. The optimization step is very resource intensive. Therefore, KIDD [54] employs Kernel Ridge Regression (KRR) for a simplified optimization objective with a single-level problem. A related approach is SFGC [62] which also makes use of KRR. While the above methods output a condensed graph, SFGC outputs graph-free data like the node features but not the graph structure. The Graph-Skeleton model [3] classifies nodes of web graphs into target nodes and background nodes. Background nodes are fetched based on connectivity and feature correlation to support the target nodes. They are condensed, and a skeleton graph is created. The methods have in common that either a hand-crafted optimization objective is minimized or a handcrafted importance score is calculated. The underlying assumption is that some nodes are more important than other nodes or that some nodes can be grouped into super-nodes without losing too much information.

6 CONCLUSIONS

This work investigates the boundaries of minimal graph sizes or maximal graph reduction in the case of GNN training. We propose combining random graph sparsification with GNN sampling, demonstrating that both methods combined are able to maintain the accuracy while reducing training time significantly. The combination of both methods outperforms the use of either method alone in terms of epoch time and accuracy. We identify a trade-off between the degree of graph sparsification, the choice of sampling method, and the fanout parameter. Across all graphs, reducing to around 40% of the original size and a fanout parameter of 4 yields the best results in terms of training time reduction and maintaining a sufficient accuracy. Further, we assume that randomly reducing the graph size does not introduce additional bias on the input graph. Inference time decreases by up to 75% when reducing the graph by 60%, enhancing scalability for applications, such as recommender systems and fraud detection, where GNNs frequently need to be re-trained and predictions can be time-critical.

REFERENCES

[1] Muhammed Fatih Balin and Ümit Çatalyürek. 2024. Layer-Neighbor Sampling—Defusing Neighborhood Explosion in GNNs. *Advances in Neural Information Processing Systems* 36 (2024).

[2] Chen Cai, Dingkang Wang, and Yusu Wang. 2021. Graph coarsening with neural networks. In *9th International conference on Learning Representations*.

[3] Linfeng Cao, Haoran Deng, Yang Yang, Chunping Wang, and Lei Chen. 2024. Graph-Skeleton:˜ 1% Nodes are Sufficient to Represent Billion-Scale Graph. In *Proceedings of the ACM on Web Conference 2024*. 570–581.

[4] Hao Chen, Yuanchen Bei, Qijie Shen, Yue Xu, Sheng Zhou, Wenbing Huang, Feiran Huang, Senzhang Wang, and Xiao Huang. 2024. Online Billion-Scale Recommender Systems with Macro Graph Neural Networks. In *The Web Conference 2024*.

[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *Int. Conf. on Learning Representations*. https://openreview.net/forum?id=rytstxWAW

[6] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *International Conference on Machine Learning*. PMLR, 942–950.

[7] Yuhan Chen, Haojie Ye, Sanketh Vedula, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. 2023. Demystifying graph sparsification algorithms in graph properties preservation. *Proceedings of the VLDB Endowment* 17, 3 (2023), 427–440.

[8] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD int. conf. on knowledge discovery & data mining*. 257–266.

[9] Frederik Michel Dekking. 2005. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media. 181–190 pages.

[10] Charles Dickens, Edward Huang, Aishwarya Reganti, Jiong Zhu, Karthik Subbian, and Danai Koutra. 2024. Graph coarsening via convolution matching for scalable graph neural network training. In *Companion Proceedings of the ACM on Web Conference 2024*. 1502–1510.

[11] Sebastian Eliassen and Raghavendra Selvan. 2024. Activation Compression of Graph Neural Networks Using Block-Wise Quantization with Improved Variance Minimization. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 7430–7434.

[12] Junfeng Fang, Xinglin Li, Yongduo Sui, Yuan Gao, Guibin Zhang, Kun Wang, Xiang Wang, and Xiangnan He. 2024. Exgc: Bridging efficiency and explainability in graph condensation. In *Proceedings of the ACM on Web Conference 2024*. 721–732.

[13] Zhuo Feng. 2016. Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In *Proceedings of the 53rd Annual Design Automation Conference*. 1–6.

[14] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. 2023. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems* 1, 1 (2023), 1–51.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[16] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE Int. Joint Conf. on Neural Networks, 2005.*, Vol. 2. IEEE, 729–734.

[17] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proc. of the 22nd int. conf. on World Wide Web*. 505–514.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[19] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B Aditya Prakash, and Wei Jin. 2024. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. *arXiv preprint arXiv:2402.03358* (2024).

[20] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. In *Proc. of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1. Curran. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/db8e1af0cb3aca1ae2d0018624204529-Paper-round2.pdf

[21] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[22] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 675–684.

[23] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2021. Graph Condensation for Graph Neural Networks. In *International Conference on Learning Representations*.

[24] George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997).

[25] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)* (2015).

[26] Dominique LaSalle, Md Mostofa Ali Patwary, Nadathur Satish, Narayanan Sundaram, Pradeep Dubey, and George Karypis. 2015. Improving graph partitioning for modern graphs and architectures. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. 1–4.

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

[28] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.

[29] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[30] Gaotang Li, Marlena Duda, Xiang Zhang, Danai Koutra, and Yujun Yan. 2023. Interpretable sparsification of brain graphs: Better practices and effective designs for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1223–1234.

[31] Chuang Liu, Xueqi Ma, Yibing Zhan, Liang Ding, Dapeng Tao, Bo Du, Wenbin Hu, and Danilo P Mandic. 2023. Comprehensive graph gradual pruning for sparse training in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[32] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA J. of Automatica Sinica* 9, 2 (2021), 205–234.

[33] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, Dongrui Fan, Shirui Pan, and Yuan Xie. 2022. Survey on Graph Neural Network Acceleration: An Algorithmic Perspective. In *Proc. of the Thirty-First Int. Joint Conf. on AI, IJCAI-22*, Lud De Raedt (Ed.). Int. Joint Conferences on AI Organization, 5521–5529. https://doi.org/10.24963/ijcai.2022/772 Survey Track.

[34] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and Choose: A GNN-based Imbalanced Learning Approach for Fraud Detection. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3168–3177. https://doi.org/10.1145/3442381.3449989

[35] Zirui Liu, Kaixiong Zhou, Fan Yang, Li Li, Rui Chen, and Xia Hu. 2021. Exact: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*.

[36] Andreas Loukas. 2019. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research* 20, 116 (2019), 1–42.

[37] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. 2021. UltraGCN: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 1253–1262.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[39] Veeru Sadhanala, Yu-Xiang Wang, and Ryan Tibshirani. 2016. Graph sparsification approaches for laplacian smoothing. In *Artificial Intelligence and Statistics*. PMLR, 1250–1259.

[40] Guillaume Salha, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. A degeneracy framework for scalable graph autoencoders. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 3353–3359.

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

[42] Marco Serafini and Hui Guan. 2021. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review* 55, 1 (2021), 68–76.

[43] Kartik Sharma, Yeon-Chang Lee, Sivagami Nambi, Aditya Salian, Shlok Shah, Sang-Wook Kim, and Srijan Kumar. 2024. A Survey of Graph Neural Networks for Social Recommender Systems. *ACM Comput. Surv.* (apr 2024). https://doi.org/10.1145/3661821 Just Accepted.

[44] Gregory T Sica. 2006. Bias in research studies. *Radiology* 238, 3 (2006), 780–789.

[45] Zheng Song, Fengshan Bai, Jianfeng Zhao, and Jie Zhang. 2021. Spammer Detection Using Graph-level Classification Model of Graph Neural Network. In *2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*. 531–538. https://doi.org/10.1109/ICBAIE52039.2021.9390066

[46] Shyam A Tailor, René De Jong, Tiago Azevedo, Matthew Mattina, and Partha Maji. 2021. Towards efficient point cloud graph neural networks through architectural simplification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2095–2104.

[47] Jiangnan Tang, Youquan Wang, Jie Cao, Haicheng Tao, and Guixiang Zhu. 2022. Inter-and Intra-Graph Attention Aggregation Learning for Multi-relational GNN Spam Detection. *Procedia Computer Science* 214 (2022), 1522–1530.

[48] Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Trans. Math. Software* 11, 1 (March 1985), 37–57. https://doi.org/10.1145/3147.3165

[49] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[50] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset distillation. *arXiv preprint arXiv:1811.10959* (2018).

[51] Qiang Wei and Guangmin Hu. 2022. Evaluating graph neural networks under graph sampling scenarios. *PeerJ Computer Science* 8 (2022), e901.

[52] Ryan Wickman, Xiaofei Zhang, and Weizi Li. 2022. A Generic Graph Sparsification Framework using Deep Reinforcement Learning. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1221–1226.

[53] Hang-Yang Wu and Yi-Ling Chen. 2020. Graph sparsification with generative adversarial network. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1328–1333.

[54] Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, Hao Yang, and Hanghang Tong. 2023. Kernel Ridge Regression-Based Graph Dataset Distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2850–2861.

[55] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32 (2019).

[56] Taraneh Younesian, Thiviyan Thanapalasingam, Emile van Krieken, Daniel Daza, and Peter Bloem. 2023. GRAPES: Learning to Sample Graphs for Scalable Graph Neural Networks. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*.

[57] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 19665–19679.

[58] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *Int. Conf. on Learning Representations*. https://openreview.net/forum?id=BJe8pkHFwS

[59] Tianle Zhang, Yuchen Zhang, Beining Yang, Kai Wang, Tianyi Li, Kaipeng Zhang, Wenqi Shao, Ping Liu, Joey Tianyi Zhou, and Yang You. 2023. CTRL: Graph condensation via crafting rational trajectory matching. (2023).

[60] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2021. Dataset Condensation with Gradient Matching. In *Ninth International Conference on Learning Representations 2021*.

[61] Weichen Zhao, Tiande Guo, Xiaoxi Yu, and Congying Han. 2023. A learnable sampling method for scalable graph neural networks. *Neural Networks* 162 (2023), 412–424.

[62] Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. 2024. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. *Advances in Neural Information Processing Systems* 36 (2024).

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