SUGAR: Efficient Subgraph-level Training via Resource-aware Graph Partitioning

Zihui Xue,Yuedong Yang,and Radu Marculescu, *Fellow, IEEE*

Abstract—Graph Neural Networks (GNNs) have demonstrated a great potential in a variety of graph-based applications, such as recommender systems, drug discovery, and object recognition. Nevertheless, resource-efficient GNN learning is a rarely explored topic despite its many benefits for edge computing and Internet of Things (IoT) applications. To improve this state of affairs, this work proposes efficient subgraph-level training via resource-aware graph partitioning (SUGAR). SUGAR first partitions the initial graph into a set of disjoint subgraphs and then performs local training at the subgraph-level. We provide a theoretical analysis and conduct extensive experiments on five graph benchmarks to verify its efficacy in practice. Our results across five different hardware platforms demonstrate great runtime speedup and memory reduction of SUGAR on large-scale graphs. We believe SUGAR opens a new research direction towards developing GNN methods that are resource-efficient, hence suitable for IoT deployment.

Index Terms—Graph Neural Networks, Resource-efficient Learning, Edge Computing

1 Introduction

Graphs are non-Euclidean data structures that can model complex relationships among a set of interacting objects, for instance, social networks, knowledge graphs, or biological networks. Given the huge success of deep neural networks for Euclidean data (*e.g.*, images, text and audio), there is an increasing interest in developing deep learning approaches for graphs too. Graph Neural Networks (GNNs) generalize the convolution operation to the non-Euclidean domain [1]; they demonstrate a great potential for various graph-based applications, such as node classification [2], link prediction [3] and recommender systems [4].

The rapid development of smart devices and IoT applications has spawned a great interest in many edge AI applications. Training models locally becomes a growing trend as this can help avoid data transmission to the cloud, reduce communication latency, and better preserve privacy [5]. For instance, in a graph-based recommender system, user data can be quite sensitive and hence it's better to store it locally [6]. This brings about the need for *resource-efficient graph learning*.

While there is much discussion about locally training Convolutional Neural Networks (CNNs) [7], efficient on-device training for GNNs is rarely explored. Different from CNNs, where popular models such as ResNet [8] are deep and have a large parameter space, mainstream GNN models are shallow and more lightweight. However, the major bottleneck of GNN training comes from the nodes dependencies in the input graph. Consequently, graph convolution suffers from a high computational cost, as the representation of a node in the current layer needs to be computed recursively by the representations of all neighbors in its previous layer. Moreover, storing the intermediate features for all nodes requires much memory space, especially when the graph size grows. For instance, for the *ogbn-products* graph in our experiments (Table 1), full-batch training requires a GPU with 33GB of memory [9]. Thus scaling GNN training to large-scale graphs remains a big challenge. The problem is more severe for a resource-constrained scenario like IoT, where GNN training is heavily constrained by the computation, memory, and communication costs.

Various approaches have been proposed to alleviate the computation and memory burden of GNNs. For instance, sampling-based approaches aim at reducing the neighborhood size via layer sampling [10], [11], [12], clustering based sampling [13] and graph sampling [14] techniques; these prior works approach this problem purely from an algorithmic angle. A few recent works [15], [16] investigate the topic of distributed multi-GPU training of GNNs and achieve good parallel efficiency and memory scalability while using large GPU clusters.

A common limitation of all these approaches is that they *do not* take the real hardware constraints into consideration. For mobile devices with limited memory budgets, the input graph can be too large to fit entirely in the main memory. In addition, the communication overhead among real IoT devices is significantly larger than when using GPU clusters, rendering distributed training approaches not readily applicable to such scenarios. This calls for a new approach for *resource-efficient GNN learning*, which is precisely the focus of our paper.

In this work, we propose a novel approach that trains GNNs efficiently with multiple devices in a resource-limited scenario. To this end, we (1) design a graph partitioning method that accounts for resource constraints and graph topology; (2) train a set of local GNNs at the *subgraph-level* for computation, memory and communication savings. Our contributions are as follows:

- We formulate the problem of training GNNs with multiple resource-constrained devices. Although our formulation targets various mobile and edge devices

- *Zihui Xue, Yuedong Yang and Radu Marculescu are with the Department of Electrical and Computer Engineering at The University of Texas at Austin.*
E-mail: {sherryxue, albertyoung, radum}@utexas.edu

(*e.g.*, mobile phones, Raspberry Pi), it is also applicable to powerful machines equipped with GPUs.

- We propose SUGAR, a GNN training framework that aims at improving training scalability. We provide complexity analysis, error bound and convergence analysis of the proposed estimator.
- We show that SUGAR achieves the best runtime and memory usage (with similar accuracy) when compared against state-of-the-art GNN approaches on five large-scale datasets and across multiple hardware platforms, ranging from edge devices (*i.e.*, Raspberry Pi, Jetson Nano) to a desktop equipped with powerful GPUs.
- We illustrate the flexibility of SUGAR by integrating it with both full-batch and mini-batch algorithms such as GraphSAGE [10] and GraphSAINT [14]. Experimental results demonstrate that SUGAR can achieve up to 33× runtime speedup on *ogbn-arxiv* and 3.8× memory reduction on *Reddit*. On the *ogbn-products* graph with over 2 million nodes and 61 million edges, SUGAR achieves 1.62× speedup over GraphSAGE and 1.83× memory reduction over GraphSAINT with a better test accuracy ($\sim$0.7%).

The remainder of the paper is organized as follows. In Section 2, we discuss prior work. In Section 3, we formulate the problem and describe our proposed training framework SUGAR. Experimental results are presented in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

The relevant prior work comes from three directions as discussed next.

2.1 Graph Neural Networks

Modern GNNs adopt a neighborhood aggregation scheme to learn representations for individual nodes or the entire graph. Graph Convolution Network (GCN) [2] is a pioneering work that generalizes the use of regular convolutions to graphs. GraphSAGE [10] provides an inductive graph representation learning framework. To improve the representation ability of GNNs, Graph Attention Networks (GAT) [17] introduce self-attention to the graph convolution operation. Apart from pursuing higher accuracy, a few GNN architecture improvements [18], [19] have been made towards higher training efficiency.

2.2 GNN training algorithms

Current GNN training algorithms can be categorized into full-batch training and mini-batch training.

Full-batch training was first proposed for GCNs [2]; the gradient is calculated based on the global graph and updated once per epoch. Despite being fast, full-batch gradient descent is generally infeasible for large-scale graphs due to excessively large memory requirements and slow convergence.

Mini-batch training was first proposed in GraphSAGE [10]; the gradient update is based on a proportion of nodes in the graph and updated a few times during each training epoch. Mini-batch training leads to memory efficiency at the cost of increased computation. Since the neighborhood aggregation scheme involves recursive calculation of a node's neighbors layer by layer, time complexity becomes exponential with respect to the number of GNN layers; this is known as the *neighborhood expansion problem*.

Following the idea of neighbor sampling, FastGCN [11] further proposes the importance node sampling to reduce variance. The work of [12] proposes a control variate based algorithm that allows a smaller neighbor sample size.

A few recent works propose alternative ways to construct mini-batches instead of layer-wise sampling. For instance, ClusterGCN [13] first partitions the training graph into clusters and then randomly groups clusters together as a batch. GraphSAINT [14] builds mini-batches by sampling the training graph and ensures a fixed number of nodes in all layers.

2.3 Graph Sparsification

Recent works have also investigated graph sparsification (*i.e.*, pruning edges of the training graph) for GNN learning. In many real-world applications, graphs exhibit complex topology patterns. Some edges may be erroneous or task-irrelevant, and thus aggregating this information weakens the generalizability of GNNs [20]. As shown by [21] and [22], edges of the input graph may be pruned without loss of accuracy.

Two recent works introduce computation efficiency into the problem. More precisely, SGCN [23] proposes a neural network that prunes edges of the input graph; they show that using sparsified graphs as the new input for GNNs brings computational benefits. UGS [24] presents a graph lottery ticket type of approach; they sparsify the input graph, as well as model weights during training to save inference computation.

3 Our Proposed Method

3.1 Problem Formulation

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set and $\mathcal{E}$ represents the set of edges. Let $N = |\mathcal{V}|$ denote the number of nodes and $A \in R^{N\times N}$ be the adjacency matrix of $\mathcal{G}$. Every node i is characterized by a F-dimensional feature vector $x_i \in R^F$. We use $X \in R^{N\times F}$ to represent the feature matrix of all nodes in $\mathcal{G}$.

Consider a node-level prediction problem with the following objective:

$$\begin{aligned} \min_W \mathcal{L} &= \frac{1}{N}\sum_{i=1}^{N} f(y_i, z_i) \\ z_i &= g(x_i; W) \end{aligned} \tag{1}$$

where f is the objective function (*e.g.*, cross entropy for node classification), y_i and z_i denotes the true label and prediction of node i, respectively. $g(\cdot)$ denotes a graph neural network parameterized by W that generates node-level predictions.

Suppose there are K devices available for training, and let $\mathcal{B}^k_{MEM}$ denote the memory budget of device k. Motivated by the notorious inefficiency that centralized graph learning suffers from, we aim at *distributing the training process* to improve the training scalability. The key is to assign N nodes of graph $\mathcal{G}$ to K devices, and then do local training on *each* device. We formulate it as two subproblems below.

First, we define a graph partitioning strategy $\mathcal{P} : \mathcal{V} \rightarrow (\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_K)$ that divides the node set $\mathcal{V}$ into K subsets such that:

$$\cup_k \mathcal{V}_k = \mathcal{V}, \;\; H(\mathcal{SG}_k) < \mathcal{B}^k_{MEM}, \;\; \forall k \in [K] \tag{2}$$

where $[K] = \{1, ..., K\}$, $\mathcal{SG}_k$ denotes the subgraph induced by node set $\mathcal{V}_i$, H is a static function that maps a given subgraph $\mathcal{SG}_i$ to the device memory requirements for training. For maximum generality, here we do not require $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$. In other words, a node i can be assigned to more than one hardware device, and let $\mathcal{P}_i$ denote the set of hardware devices where node i is assigned to.

Next, we adopt subgraph-level training, *i.e.*, for device k, we maintain a local GNN model, denoted by $W^{\langle k \rangle}$ that takes the subgraph $\mathcal{SG}_k$ as its input graph. Let $W = \frac{1}{K}\sum_{k=1}^{K} W^{\langle k \rangle}$, thus the objective can be reformulated as:

$$\begin{aligned} \min_W \mathcal{L} &= \frac{1}{N}\sum_{i=1}^{N} f(y_i, z_i) \\ z_i &= \frac{1}{|\mathcal{P}_i|}\sum_{k \in \mathcal{P}_i} g(x_i; W^{\langle k \rangle}) \end{aligned} \tag{3}$$

Based on the formulation above, we propose SUGAR, a distributed training framework that: (1) partitions the input graph subject to resource constraints; (2) adopts local subgraph-level training. Figure 1 provides a simple illustration of SUGAR for a two-device system. We describe our design choices in detail in the following sections.

3.2 Theoretical Basis

Recall that we define a graph partitioning strategy $\mathcal{P}$ that divides N nodes into K node sets $(\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_K)$. Taking K subgraphs induced by the node sets into consideration, a graph partitioning strategy $\mathcal{P}$ can be viewed as a way to produce a sparser adjacency matrix A_{SG}, from the original matrix A. A_{SG} is a block-diagonal matrix of A, *i.e.*,

$$A_{SG} = \begin{bmatrix} A_{\mathcal{V}_1} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & & & \vdots \\ 0 & & A_{\mathcal{V}_k} & & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & A_{\mathcal{V}_K} \end{bmatrix} \tag{4}$$

where $A_{\mathcal{V}_k}$ denotes the adjacency matrix of subgraph k.

We show below that adopting A_{SG} for training offers the benefits of high computational efficiency and low memory requirements. Moreover, we provide the error bound and convergence analysis of this approximation for a graph convolutional network (GCN) [2].

Complexity Analysis. The propagation rule for the l-th layer GCN is:

$$Z^{(l+1)} = A^{norm} H^{(l)} W^{(l)}, H^{(l+1)} = \sigma(Z^{(l+1)}) \tag{5}$$

where σ represents an activation function, A^{norm} denotes the normalized version of A, *i.e.*, $A^{norm} = \hat{D}^{-1/2}\hat{A}\hat{D}^{1/2}, \hat{A} = A + I_N, \hat{D}_{ii} = \sum_j \hat{A}_{ij}$ and I_N is an N-dimensional identity matrix. $H^{(l)}$ and $H^{(l+1)}$ denotes the input and output feature matrices in layer l, respectively. $Z^{(l)}$ is the node feature matrix before the activation function in layer l and $Z^{(L)}$ denotes final node predictions (*i.e.*, output of the GCN). $W^{(l)} \in R^{F_l \times F_{l+1}}$ represents the weight matrix of layer l, where F_l and F_{l+1} is the input and output feature dimension, respectively. Therefore, for the l-th layer GCN, the *training time complexity* is $\mathcal{O}(|\mathcal{E}|F_l + NF_lF_{l+1})$ and *memory complexity* is $\mathcal{O}(NF_{l+1} + F_lF_{l+1})$. We make two observations here: (a) Real-world graphs are usually sparse and $\frac{|\mathcal{E}|}{N}$ is generally smaller than feature number F_{l+1}. Thus, the second term dominates the time complexity; (b) For large-scale graphs, the number of nodes N is much greater than the number of features. Consequently, $\mathcal{O}(NF_{l+1})$ dominates the memory complexity. It is easy to verify that the number of nodes N imposes a computation hurdle on training. Partitioning the input graph into K subgraphs reduces the number of nodes N to $N_k = |\mathcal{V}_k|$ for every local model. Since N_k is about $1/K$ of N, the proposed approach is expected to achieve up to K times speedup, and as little as $1/K$ of the original memory requirements.

Error Bound Analysis. Let our proposed estimator be SG. The l-th layer propagation rule of a GCN with the SG estimator is:

$$Z^{(l+1)}_{SG} = A^{norm}_{SG} H^{(l)}_{SG} W^{(l)}, H^{(l+1)}_{SG} = \sigma(Z^{(l+1)}_{SG}) \tag{6}$$

where $Z^{(l+1)}_{SG}$ and $H^{(l+1)}_{SG}$ denote the node representations produced by the SG estimator in layer $l+1$ before and after activation, respectively.

Assume that we run graph partitioning for M times to obtain a sample average of A^{norm}_{SG} before training. Let $\epsilon = \|A^{norm}_{SG} - A^{norm}\|_\infty$ denote the error in approximating A^{norm} with A^{norm}_{SG}. For simplicity, we will omit the superscript $norm$ from now on.

The following lemma states that the error of node predictions given by the SG estimator is bounded.

Lemma 1. *For a multi-layer GCN with fixed weights, assume that: (1) $\sigma(\cdot)$ is ρ-Lipschitz and $\sigma(0) = 0$, (2) input matrices A, X and model weights $\{W^{(l)}\}_{l=1}^{L}$ are all bounded, then there exists C such that $\left\|Z^{(l)}_{SG} - Z^{(l)}\right\|_\infty \leq C\epsilon, \forall l \in [L]$ and $\left\|H^{(l)}_{SG} - H^{(l)}\right\|_\infty \leq C\epsilon, \forall l \in [L-1]$.*

The proof of Lemma 1 is provided in Section **??**. Lemma 1 motivates us to design a graph partitioning method that generates small ϵ so that the output of the SG estimator is close to the exact value. This will be discussed in detail in the next subsection.

Convergence Analysis. Let W_t denote the model parameters at training epoch t and W_* denote the optimal model weights. $\nabla\mathcal{L}(W) = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial f(y_i, z_i^{(L)})}{\partial W}$ and $\nabla\mathcal{L}_{SG}(W) = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial f(y_i, z_{SG,i}^{(L)})}{\partial W}$ represent the gradients of the exact GCN and SG estimator with respect to model weights W, respectively.

Theorem 1 states that with high probability gradient descent training with the approximated gradients of the SG estimator (*i.e.*, $\nabla\mathcal{L}_{SG}(W)$) converges to a local minimum.

Theorem 1. *Assume that: (1) the loss function $\mathcal{L}(W)$ is ρ-smooth, (2) the gradients of the loss $\nabla\mathcal{L}(W)$ and $\nabla\mathcal{L}_{SG}(W)$ are bounded*

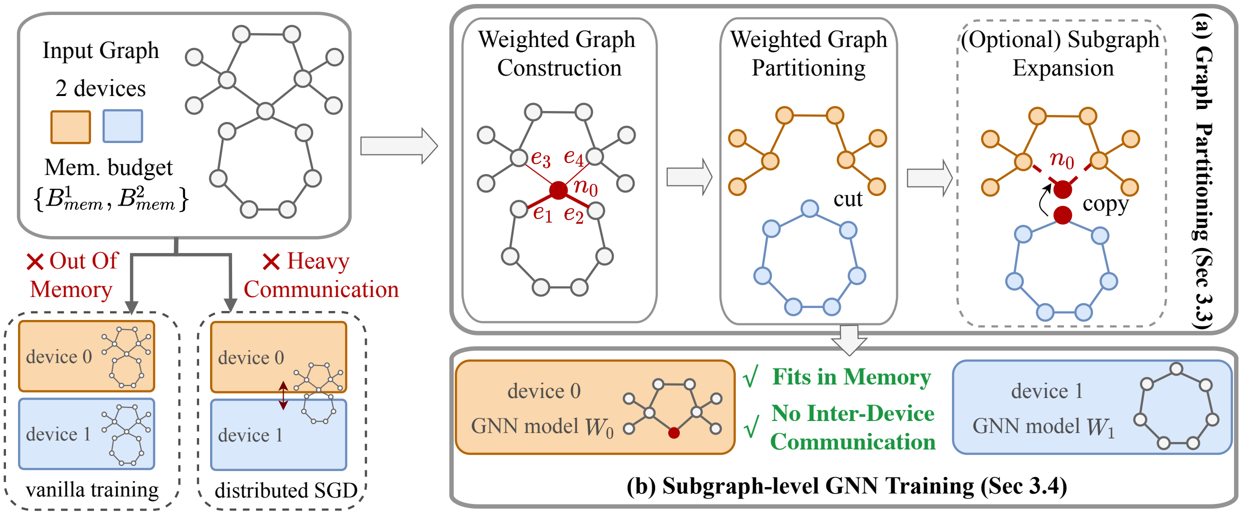

Input Graph
2 devices
Mem. budget
$\{B^1_{mem}, B^2_{mem}\}$
✕ Out Of Memory
✕ Heavy Communication
device 0
device 1
vanilla training
device 0
device 1
distributed SGD
Weighted Graph Construction
e_3 e_4 n_0 e_1 e_2
Weighted Graph Partitioning
cut
(Optional) Subgraph Expansion
n_0
copy
(a) Graph Partitioning (Sec 3.3)
device 0
GNN model W_0
√ Fits in Memory
√ No Inter-Device Communication
device 1
GNN model W_1
(b) Subgraph-level GNN Training (Sec 3.4)

Fig. 1. While vanilla training is likely to run out of memory when the graph size is large and distributed stochastic gradient descent (SGD) requires heavy intermediate communication among devices, SUGAR provides a solution that is memory efficient and requires no inter-device communication. The proposed SUGAR consists of two stages: (a) graph partitioning and (b) subgraph-level GNN training. Graph partitioning involves three steps: (1) transform the input graph $\mathcal{G}$ to a weighted graph $\mathcal{G}^w$; (2) apply METIS to the weighted graph $\mathcal{G}^w$, where edges with large weights are more likely to be preserved; (3) (optional) expand the node set of the obtained subgraph according to memory budgets.

for any choice of W, (3) the gradient of the objective function $\frac{\partial f(y,z)}{\partial z}$ is ρ-Lipschitz and bounded, (4) the activation function $\sigma(\cdot)$ is ρ-Lipschitz, $\sigma(0) = 0$ and its gradient is bounded,

then there exists $C > 0$, s.t., $\forall M, T$, for a sufficiently small δ, if we run graph partitioning for M times and run gradient descent for $R \leq T$ epochs (where R is chosen uniformly from $[T]$, the model update rule is $W_{t+1} = W_t - \gamma \nabla \mathcal{L}_{SG}(W_t)$, and step size $\gamma = \frac{1}{\rho\sqrt{T}}$), we have:

$$P(\mathbb{E}_R \|\nabla \mathcal{L}(W_R)\|_F^2 \leq \delta) \geq$$
$$1 - 2\exp\{-2M(\frac{\delta}{2C} - \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C - \delta}{2C(\sqrt{T} - 1)})^2\}$$

With M and T increasing, the right-hand-side of the inequality becomes larger. This implies that there is a higher probability for the loss to converge to a local minimum. The full proof is provided in the Appendix.

3.3 Graph Partitioning

From Lemma 1, we conclude that a graph partitioning method that yields a smaller $|A_{SG} - A|$ leads to a smaller error in node predictions. Therefore, we aim at minimizing the difference between A_{SG} and A. In other words, the objective of graph partitioning should be to *minimize the number of edges of the incident nodes* that belong to different subsets. As such, this is identical to the goal of various existing graph partitioning methods, making such approaches good candidates to use with our framework. We choose METIS [25] due to its efficiency in handling large-scale graphs. However, the traditional graph partitioning algorithms are *not* intended for modern GNNs and the learning component of the problem is missing. Consequently, we present a modified version of METIS that is suited to our problem and relies on two new ideas discussed next.

a) Weighted Graph Construction. We build a weighted graph $\mathcal{G}^w$ from the input graph $\mathcal{G}$. The *weight* of an edge e_{uv} is defined based on the degree of its two incident nodes:

$$\begin{aligned} weight(e_{uv}) &= d_{max} + 1 - deg(u) - deg(v) \\ d_{max} &= max\{deg(u) + deg(v), \forall e_{uv} \in \mathcal{E}\} \end{aligned} \tag{7}$$

Let A^w denote the adjacency matrix of the weighted graph $\mathcal{G}^w$, where element a^w_{ij} is the edge weight $weight(e_{ij})$; a^w_{ij} is 0 if there is no edge connecting nodes i and j.

The key intuition behind our first idea lies in the neighborhood aggregation scheme of GNNs. Consider two nodes u and v, where u is a hub node connected to many other nodes, while v has only one neighbor. As GNNs propagate by aggregating the neighborhood information of nodes, removing the only edge of node v may possibly lead to wrong predictions. On the other hand, pruning an edge of u is more acceptable since there are many neighbors contributing to its prediction. Consider the graph in Figure 1 as an example. Cutting the edges $e_1 \cup e_2$ and $e_3 \cup e_4$ are both feasible solutions for METIS. However, considering the fact that nodes connected to e_1 and e_2 have less topology information, our proposed method will preserve them and cut edges $e_3 \cup e_4$ instead; this can lead to a better learning performance.

As can be concluded from this small example, edges connected to small-degree nodes are critical to our problem and should be preserved. Conversely, edges connected to high-degree nodes may be intentionally ignored. This explains our weights definition strategy. Consequently, we incorporate the above observation into our partitioning objective and apply METIS to the pre-processed graph $\mathcal{G}^w$.

b) Subgraph Expansion. After obtaining the partitions with our modified METIS, we propose the second idea, *i.e.*, expand the subgraph based on available hardware resources. Although METIS only provides partitioning results where the node sets do not overlap, our general formulation in Section 3.1 allows nodes to belong to multiple partitions. This brings great flexibility to our approach to adjust the node number for each device according to its memory budget.

Suppose the available memory of device k is larger than the actual requirement of training a GNN on subgraph k (*i.e.*, $H(\mathcal{SG}_k) < B^k_{MEM}$), then we may choose to expand the node set $\mathcal{V}_k$ by adding the one-hop neighbors of nodes that do not belong to $\mathcal{V}_k$. As illustrated in Figure 1 (a), we can expand the node set of the subgraph on device 0 (marked in light brown) to include node n_0 as well. While expanding the subgraph is likely to yield higher accuracy, training time and memory requirement will also increase. Therefore, this is an optional step, only if the hardware resources allow it.

3.4 Subgraph-level Local Training

From the original formulation in Equation 3, if $|\mathcal{P}_i| > 1$, *i.e.*, a node i is assigned to multiple devices, calculating its loss and backpropagation can involve heavy communication among devices. To address this problem, we provide the following result to decouple the training of K local GNN models from each other.

Proposition 1. *If $f(y, z)$ is convex with respect to z, then the upper bound of $\mathcal{L}$ in Equation 3 is given by:*

$$\begin{aligned} \mathcal{L} &\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in\mathcal{V}_k}\frac{1}{|\mathcal{P}_i|}f(y_i, z_i) \\ z_i &= g(x_i, W^{\langle k\rangle}) \end{aligned} \tag{8}$$

Proof. By convexity of f, using Jensen's inequality [26] gives us:

$$f(y_i, \frac{1}{|\mathcal{P}_i|}\sum_{k\in\mathcal{P}_i} g(x_i; W^{\langle k\rangle}) \leq \frac{1}{|\mathcal{P}_i|}\sum_{k\in\mathcal{P}_i} f(y_i, g(x_i; W^{\langle k\rangle})) \tag{9}$$

By changing the operation order and regrouping the indices, we further derive:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{|\mathcal{P}_i|}\sum_{k\in\mathcal{P}_i} f(y_i, z_i) = \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in\mathcal{V}_k}\frac{1}{|\mathcal{P}_i|}f(y_i, z_i) \tag{10}$$

Therefore,

$$\begin{aligned} \mathcal{L} &\leq \frac{1}{K}\sum_{k=1}^{K}\sum_{i\in\mathcal{V}_k}\frac{1}{|\mathcal{P}_i|}f(y_i, z_i) \\ z_i &= g(x_i, W^{\langle k\rangle}) \end{aligned} \tag{11}$$

Proposition 1 is proved.

Proposition 1 allows us to shift the perspective from 'node-level' to 'device-level'. We adopt the upper bound of $\mathcal{L}$ in Equation 8 as the new training objective. Now, the local model updates involving node i *do not* depend on other models (*i.e.*, $\{W^{\langle k\rangle}\}_{k\in\mathcal{P}_i}$) any more. Optimizing the new objective naturally reduces the upper bound of the original one and avoids significant communication costs, thus leading to high training efficiency.

Furthermore, motivated by deployment challenges in real IoT applications, where communication among devices is generally not guaranteed, we propose to reduce inter-device communication down to zero in our framework. In particular, we maintain K distinct (local) models instead of a single (global) model by keeping the local model updates within each device. The objective of our proposed subgraph-level local GNN training can be summarized as follows:

$$\begin{aligned} \min_{W^{\langle k\rangle}} \mathcal{L}_k &= \sum_{i\in\mathcal{V}_k}\frac{1}{|\mathcal{P}_i|}f(y_i, z_i),\ \forall k\in[K] \\ z_i &= g(x_i, W^{\langle k\rangle}) \end{aligned} \tag{12}$$

In training round t, every device performs local updates as:

$$W^{\langle k\rangle}_{t+1} \leftarrow W^{\langle k\rangle}_{t} - \gamma\nabla_{W^{\langle k\rangle}}\mathcal{L}_k,\ \forall k\in[K] \tag{13}$$

where $\mathcal{L}_k$ denotes the training objective of device k and γ is the learning rate (*i.e.*, step size). By decoupling training dependency among devices, we propose a feasible solution to train GNNs in resource-limited scenarios, where typical distributed GNN approaches are not applicable.

3.5 Putting it all together

Algorithm 1 SUGAR

```
Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; node feature matrix $X$; available device number $K$; device memory budget $\{B^k_{MEM}\}^K_{k=1}$; total training epochs $T$.
1: Construct $\mathcal{G}^w$ from $\mathcal{G}$ according to Equation 7
2: Partition $\mathcal{G}^w$ into $K$ subgraphs $\{\mathcal{SG}_i\}^K_1$
3: (Optional) Expand $\mathcal{SG}_i$ if $H(\mathcal{SG}_i) < B^i_{MEM}$
4: for each device $k = \{1, 2, \cdots, K\}$ in parallel do
5:     Initialize GNN model weight $W^{\langle k\rangle}_1$
6:     for epoch $t = 1, 2, \cdots, T$ do
7:         $W^{\langle k\rangle}_{t+1} \leftarrow W^{\langle k\rangle}_{t} - \gamma\nabla_{W^{\langle k\rangle}}\mathcal{L}_k$
8:     end for
9: end for
```

To sum up, the SUGAR algorithm consists of two stages: (a) graph partitioning (lines 1-3) and (b) subgraph-level GNN training (lines 4-9). Specifically, the graph partitioning involves three steps: (1) construct a weighted graph $\mathcal{G}^w$ from $\mathcal{G}$ to account for the influence of node degrees in learning (line 1). (2) Apply METIS to the weighted graph $\mathcal{G}^w$ to obtain partitioning results (line 2). (3) According to the memory budget, expand the subgraph to cover the one-hop neighbors for better performance (line 3). Then, we train K local models in parallel without requiring training-time communication among devices (lines 4-9). The proposed subgraph-level training with multiple devices achieves high training efficiency, low memory requirements and zero communication costs.

4 Experiments

4.1 Experimental Setup

We evaluate SUGAR on five node classification datasets [9], [27], selected from very diverse applications: (1) categorizing types of images based on the descriptions and common properties of online images (*Flickr*); (2) predicting communities of online posts based on user comments (*Reddit*); (3) predicting the subject areas of arxiv papers based on its title and abstract (*ogbn-arxiv*); (4) predicting the presence of protein functions based on biological associations between proteins (*ogbn-proteins*); (5) predicting the category of a

TABLE 1
Dataset Statistics. K and M denote 1,000 and 1,000,000, respectively. ‘AvgDeg.’ represents the average node degree. ‘ACC’ denotes accuracy.

Dataset	*Flickr*	*Reddit*	*ogbn-arxiv*	*ogbn-proteins*	*ogbn-products*
#Nodes	89.3K	233K	169K	133K	2,449K
#Edges	0.90M	11.6M	1.17M	39.6M	61.9M
AvgDeg.	10	50	13.77	597	50.5
#Tasks	1	1	1	112	1
#Classes	7	41	40	2	47
Metric	ACC	ACC	ACC	ROC-AUC	ACC

product in an Amazon product co-purchasing network (*ogbn-products*). Note that the task of *ogbn-proteins* is multi-label classification, while other tasks are multi-class classification. Dataset statistics are summarized in Table 1.

We include the following GNN architectures and SOTA GNN training algorithms for comparison:

- GCN [2]: Full-batch Graph Convolutional Networks.
- GraphSAGE [10]: An inductive representation learning framework that efficiently generates node embeddings for previously unseen data. Mini-batch GraphSAGE are denoted by GraphSAGE-mb.
- GAT [17]: Graph Attention Networks, a GNN architecture that leverages masked self-attention layers.
- SIGN [18]: Scalable Inception Graph Neural Networks, an architecture using graph convolution filters of different size for efficient computation.
- ClusterGCN [13]: A mini-batch training technique that partitions the graphs into a fixed number of subgraphs and draws mini-batches from them.
- GraphSAINT [14]: A mini-batch training technique that constructs mini-batches by graph sampling. The random node, random edge, and random walk based samplers are denoted by GraphSAINT-N, GraphSAINT-E, GraphSAINT-RW, respectively.

SUGAR is implemented with PyTorch [28] and DGL [29]. For all the baseline methods, we use the parameters reported in their github pages or the original paper. We report accuracy results averaged over 5 runs for *ogbn-proteins* and 10 runs for the other datasets.

For completeness, we run our experiments across multiple hardware platforms. We select five different devices with various computing and memory capabilities, namely, (1) Raspberry Pi 3B, (2) NVIDIA Jetson Nano, (3) Android phone with Snapdragon 845 processor, (4) laptop with Intel i5-8279U CPU, and (5) desktop with AMD Threadripper 3970X CPU and two NVIDIA RTX 3090 GPUs.

TABLE 2
Runtime, memory & accuracy results on *ogbn-arxiv*. ‘Avg. Time’ is the training time per epoch averaged over 100 epochs and ‘Max Mem’ denotes peak allocated memory on GPU.

	Avg. Time [ms]	SUGAR Speedup	Max Mem [GB]	Test Acc. [%]
GCN	26.9	1.68×	1.60	72.37 ± 0.10
GAT	207.8	12.99×	5.41	72.95 ± 0.14
GraphSAGE	534.7	33.42×	0.95	71.98 ± 0.17
SIGN	291.6	18.23×	0.94	71.79 ± 0.08
SUGAR	16.0		0.92	72.22 ± 0.14

TABLE 3
Runtime, memory & accuracy results on *Reddit*.

	Avg. Time [ms]	SUGAR Speedup	Max Mem [GB]	Test Acc. [%]
GraphSAGE	110.6	1.87×	5.70	96.39 ± 0.03
GraphSAGE-mb	316.5	5.36×	2.33	95.08 ± 0.05
ClusterGCN	414.4	7.01×	1.83	96.34 ± 0.01
GraphSAINT-N	341.8	5.78×	1.29	96.17 ± 0.06
GraphSAINT-E	299.8	5.07×	1.22	96.15 ± 0.06
GraphSAINT-RW	467.5	7.91×	1.23	96.23 ± 0.06
SIGN	352.8	5.97×	2.17	96.12 ± 0.05
SUGAR	59.1		1.51	96.01 ± 0.03

4.2 Results

4.2.1 Evaluations on GPUs

First, we provide evaluation of SUGAR on a two-GPU system. Table 2 and Table 3 report the average training time per epoch, maximum GPU memory usage and accuracy on *ogbn-arxiv* and *Reddit*. We base SUGAR on full-batch GCN and GraphSAGE for these two datasets, respectively. As shown in these tables, when compared with full-batch methods (*i.e.*, GCN and GAT for *ogbn-arxiv*; GraphSAGE for *Reddit*), SUGAR is much more memory efficient, as it reduces the peak memory by 1.7× for *ogbn-arxiv* and 3.8× for *Reddit* data. When compared against mini-batch methods (*i.e.*, mini-batch GraphSAGE, ClusterGCN, GraphSAINT and SIGN), the runtime of SUGAR is significantly smaller. This demonstrates the great benefits of our proposed subgraph-level training. Indeed, by restricting the neighborhood search size, SUGAR effectively alleviates the neighborhood expansion problem. In addition, it achieves very competitive test accuracies.

We combine SUGAR with popular mini-batch training methods and evaluate them on *Flickr* and *ogbn-products* dataset. Table 4 presents results of SUGAR incorporated with GraphSAINT for three sampler modes (*i.e.*, node, edge, and random walk based samplers) on *Flickr* data. Note that the accuracy we obtain (about 50%) is consistent with results in [14]. SUGAR achieves more than 2× runtime speedup and requires less memory than GraphSAINT. Test accuracy loss is within 1% in all cases.

For the largest *ogbn-products* dataset, we implement SUGAR together with three competitive GNN baselines, namely GraphSAGE, ClusterGCN and GraphSAINT. The

TABLE 4
Runtime, memory & accuracy results on *Flickr*.

	Avg. Time [ms]	Max Mem [GB]	Test Acc. [%]
GraphSAINT-N	97.0	0.41	50.64 ± 0.28
SUGAR	49.9	0.31	50.11 ± 0.12
Improvement	1.94×	1.32×	
GraphSAINT-E	71.1	0.53	50.91 ± 0.12
SUGAR	32.6	0.41	49.96 ± 0.12
Improvement	2.18×	1.29×	
GraphSAINT-RW	108.9	0.65	51.03 ± 0.20
SUGAR	37.3	0.49	50.15 ± 0.24
Improvement	2.92×	1.33×	

TABLE 5
Runtime, memory & accuracy results on *ogbn-products*.

	Avg. Time [ms]	Max Mem [GB]	Test Acc. [%]
GraphSAGE-mb	2.42	7.29	79.25 ± 0.22
SUGAR	1.49	4.43	79.97 ± 0.23
Improvement	1.62×	1.65×	
ClusterGCN	2.90	6.59	78.51 ± 0.33
SUGAR	1.97	3.36	79.34 ± 0.41
Improvement	1.47×	1.96×	
GraphSAINT-E	0.30	7.16	79.54 ± 0.27
SUGAR	0.28	3.92	80.20 ± 0.23
Improvement	1.07×	1.83×	

TABLE 6
Runtime, memory & accuracy results on *ogbn-proteins*.

	Avg. Time [sec]	Max Mem [GB]	Valid Acc. [%]	Test Acc. [%]
GAT	6.20	10.77	92.08 ± 0.08	87.20 ± 0.17
SUGAR	4.09	6.22	92.51 ± 0.08	86.41 ± 0.18
Improvement	1.52×	1.73×		

results are summarized in Table 5. SUGAR provides a better solution that leads to runtime speedup, memory reduction and even a slightly increased test accuracy for all three methods. We hypothesize that the graph partitioning eliminates some task-irrelevant edges in the original graph, and thus leads to better generalization of GNNs.

Table 6 provides results on the dense *ogbn-proteins* graph. When it comes to training GNNs on dense graphs, memory poses a significant challenge due to the neighborhood expansion problem. The results show that GAT suffers from considerable memory usage. In contrast, SUGAR effectively alleviates the issue with $1.52\times$ runtime speedup and $1.73\times$ memory reduction.

4.2.2 Evaluations on mobile and edge devices

Following the GPU setting, we proceed to evaluate SUGAR on mobile and edge devices with CPUs.

Training Time. Table 7 presents the average training time per epoch of SUGAR compared with baselines on the *Flickr* and *ogbn-arxiv* datasets. Due to the relative small size of these two datasets, we are able to train GNNs on all five hardware devices, ranging from a Raspberry Pi 3B, to a desktop equipped with high-performance CPUs. We also list the runtime on GPUs in the last column for easy comparison.

From Table 7, we can see that SUGAR demonstrates consistent speedup across all platforms, achieving over $2\times$ and $1.5\times$ speedup on the *Flickr* and *ogbn-arxiv* datasets, respectively. In addition, training a GCN on the Raspberry Pi 3B fails due to running out of memory, while SUGAR demonstrates good memory scalability and hence it can be used with such a device with a limited memory budget (*i.e.*, 1GB in this case). This also holds true for the *Reddit* dataset: SUGAR provides a feasible solution for local training on the Jetson Nano (time per epoch is 50.27s), while other baselines can not work due to large memory requirements.

Thus, for the other three datasets, we compare the runtime on Desktop-CPU and report our results in Table 8. We also observe consistent speedup across all datasets: SUGAR nearly halves the training time in all three cases.

Memory Usage. We compare the memory usage of SUGAR against GNN baselines on a CPU setting. Figure 2 illustrates the resident set size (RSS) memory usage during training on the four datasets: *ogbn-arxiv*, *Reddit*, *ogbn-proteins* and *ogbn-products*. Note that we train a full-batch version of GCN and the batch size of GAT is larger compared with GraphSAGE and GraphSAINT. This accounts for higher fluctuation in the corresponding figure. It is evident that our proposed SUGAR achieves substantial memory reductions compared with baseline GNNs. We emphasize that memory plays a critical role in GNN training. In the context of devices with limited resources, the situation is more severe since the graph dataset is already big and loading the full dataset may not be possible. By adopting subgraph-level training, SUGAR effectively alleviates the problem.

Finally, we present a case study of SUGAR on NVIDIA Jetson Nano in Table 9 to demonstrate the applicability of SUGAR to edge devices. Jetson Nano is a popular, cheap and readily available platform (we adopt the model with quad Cortex-A57 CPU and 4GB LPDDR memory) and thus considered as a good fit for our problem scenario. Apart from training time, we measure the peak RSS memory usage for the training process and calculate energy consumption. As shown in Table 9, SUGAR achieves low latency, consumes less memory and is more energy efficient when compared with baseline GNN algorithms. Therefore, it provides an ideal choice to train GNNs on devices with limited memory and battery capacity.

4.3 Scalability Analysis

4.3.1 Number of partitions

So far we have demonstrated the great performance of SUGAR with two available devices. A natural follow-up question is, *how does SUGAR perform on more devices, i.e., device number* $K > 2$. Below we provide a scalability analysis of SUGAR based on the number of partitions (*i.e.*, device number K).

We vary the number of available devices K from 2 to 8 and evaluate SUGAR on the *ogbn-arxiv*, *Reddit* and *ogbn-products* datasets. The evaluation is conducted on Desktop-GPU. Runtime speedup, peak GPU memory reduction,

TABLE 7

Average training time per epoch [sec] of SUGAR compared with GraphSAINT and GCN on *Flickr* and *ogbn-arxiv* data. We record the training time on five platforms with CPU models listed. OOM denotes Out Of Memory. We note that training a GCN on Raspberry Pi 3B is infeasible since it exceeds memory, while SUGAR still works.

Dataset		RPi 3B Cortex-A53	Jetson Cortex-A57	Phone SDM-845	Laptop i5-8279U	Desktop-CPU Zen2 3970X	Desktop-GPU RTX3090
Flickr	GraphSAINT-N	104.1	16.86	7.67	2.86	1.48	0.097
	SUGAR	48.2	7.61	3.54	1.21	0.67	0.050
	Speedup	2.16×	2.22×	2.17×	2.36×	2.24×	1.94×
ogbn-arxiv	GCN	OOM	28.10	21.96	13.80	5.16	0.027
	SUGAR	501.59	18.39	13.33	6.51	2.71	0.016
	Speedup	-	1.53×	1.65×	2.12×	1.91×	1.69×

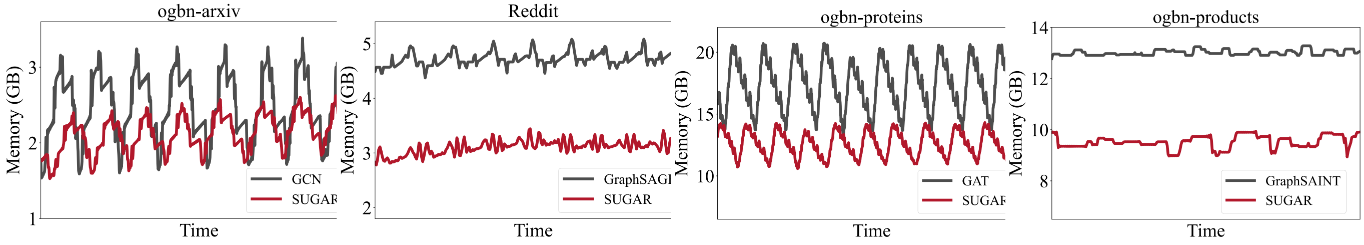


Fig. 2. Memory variation during training GNNs on Desktop-CPU for *ogbn-arxiv*, *Reddit*, *ogbn-proteins* and *ogbn-products*. For SUGAR, we plot the memory variation of the device that consumes most memory.

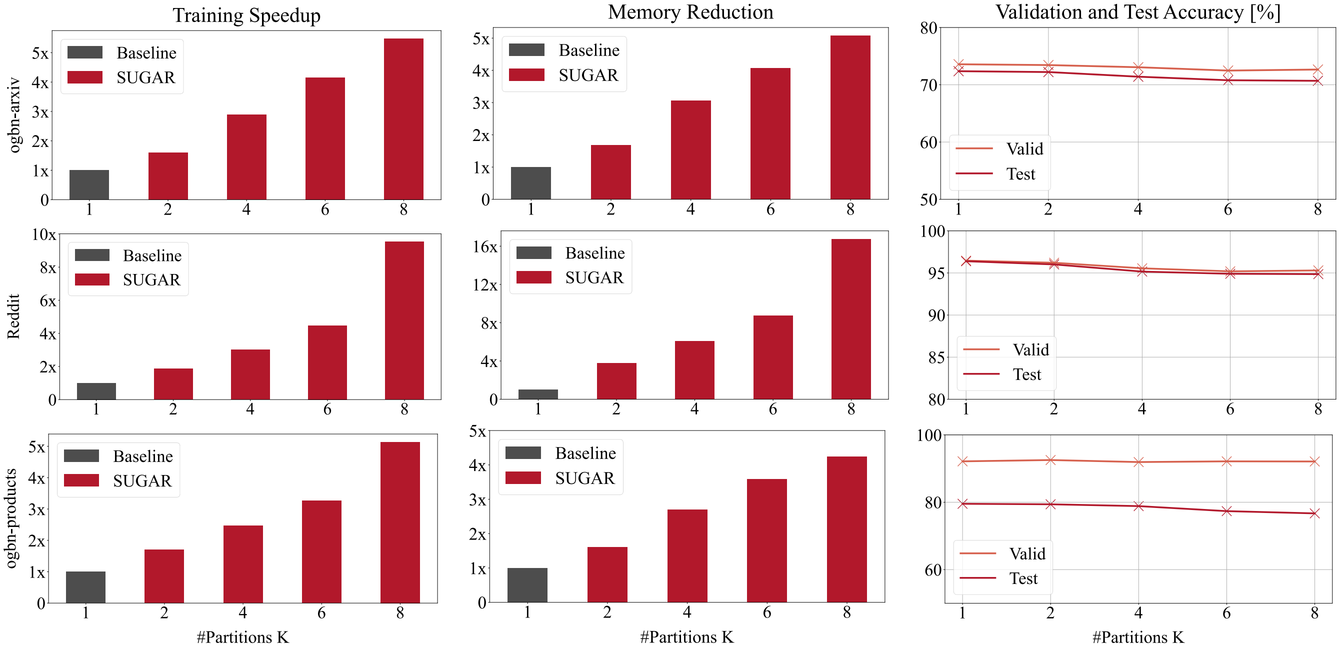
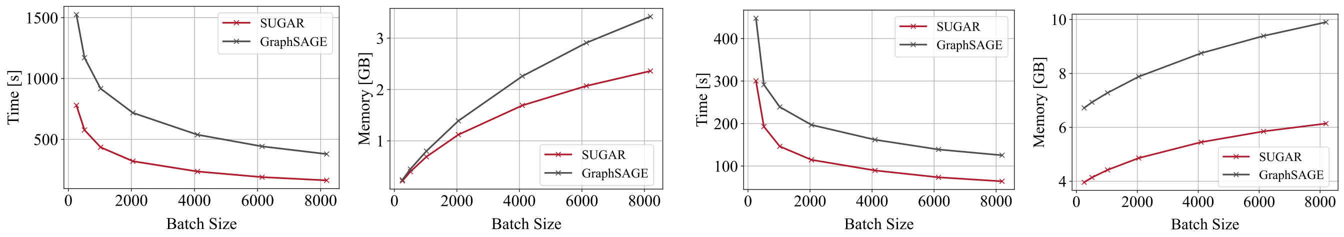


Fig. 3. Scalability analysis on the number of partitions (*i.e.*, the number of available devices K) for SUGAR. $K = 1$ refers to the baseline GNN (*i.e.*, GCN for *ogbn-arxiv*; GraphSAGE for *Reddit*; GraphSAINT for *ogbn-products*). We report the smallest training time speedup and peak GPU memory reduction among K devices (*i.e.*, the worst-case scenario) of SUGAR over the baseline.

validation and test accuracy are presented in Figure 3. With increasing K, we observe a decreased training time and peak memory usage for each local device.

As we can see, while distributing the GNN model to more devices yields computation efficiency, test accuracy drops a bit. For instance, in the case of 8 devices, the biggest decrease happens in the *ogbn-products* dataset: test accuracy is 76.69% while the baseline accuracy is 79.54%. In the meantime, SUGAR leads to $5.13\times$ speedup, as well as $4.24\times$ memory reduction compared with the baseline. Generally speaking, there exists a tradeoff between training scalability and performance. The underlying reason is that the increase of partition number K leads to more inter-device edges, which corresponds to a larger error in estimating with A_{SG} with A.

We further evaluated SUGAR in a 128-device setting.



(a) graph data on CPU

(b) graph data on GPU

Fig. 4. The training time and peak GPU memory with varying batch sizes of GraphSAGE and SUGAR for the *ogbn-products* data. We investigate two settings: (a) graph data loaded on GPU for faster execution (b) graph data loaded on CPU for memory savings.

TABLE 8
Runtime comparison against baseline methods on three large datasets. Average training time per epoch [sec] is reported. Baseline refers to GraphSAGE for *Reddit* and *ogbn-products*. GAT is the baseline for *ogbn-proteins*.

	Reddit	ogbn-products	ogbn-proteins
Baseline	2.02	170.75	269.70
SUGAR	0.88	77.05	142.7
Speedup	2.30×	2.22×	1.89×

TABLE 9
Evaluations of SUGAR on NVIDIA Jetson Nano for *Flickr* and *ogbn-arxiv*. ‘Avg. Time’ and ‘Max Mem’ denotes training time per epoch and peak resident set size (RSS) memory. We measure the time, memory and energy for training 10 epochs. SUGAR improves average training time, memory usage and energy consumption per device over baseline GNNs (*i.e.*, GraphSAINT and GCN).

Dataset		Avg. Time [sec]	Max Mem [GB]	Energy [kJ]
Flickr	GraphSAINT-N	22.62	1.05	1.13
	SUGAR	10.50	0.89	0.52
	Improvement	2.15×	1.18×	2.17×
ogbn-arxiv	GCN	28.10	2.24	1.27
	SUGAR	18.39	1.46	0.81
	Improvement	1.53×	1.53×	1.57×

The results show that the test accuracy drop compared with baseline GNNs is small, *i.e.*, within 5% when scaling up to 128 devices (*e.g.*, accuracy decreases from 72.37% to 67.80% for *ogbn-arxiv*, from 96.39% to 92.32% for *Reddit*, from 50.64% to 46.31% for *Flickr*). At the same time, we note that the memory savings are great (*e.g.*, peak memory usage per device is reduced from 1.60GB to 0.02GB for *ogbn-arxiv*). This shows that SUGAR can work with very small computation and memory requirements at the cost of slightly downgraded performance. Thus, SUGAR provides a feasible solution in extremely resource-limited scenarios while general GNN training methods are not applicable.

4.3.2 Batch Size

Finally, we study the influence of batch sizes on computational efficiency and memory scalability on SUGAR when compared with mini-batch training algorithms.

For mini-batch training algorithms, when the limited memory of device renders GNN training infeasible, a natural idea is reduce the batch size for memory savings. Here, we analyze the influence of SUGAR and the act of reducing batch sizes on computational efficiency, as well as memory scalability. We conduct experiments on the largest *ogbn-products* graph with GraphSAGE as the baseline. Two settings are considered: (a) graph data loaded on CPU, longer training time and smaller memory consumption is expected; (b) graph data loaded on GPU, the model runs faster, yet requires more GPU memory. Figure 4 provides runtime and memory results with varying batch sizes.

We list two observations below: First, SUGAR mainly improves runtime in setting (a) and achieves greater memory reduction in setting (b). This is related to the mechanism of SUGAR: each local model adopts one subgraph for training instead of the original graph. Thus, data loading time is reduced in setting (a) and putting a subgraph on GPU is more memory efficient in setting (b).

Secondly, SUGAR demonstrates to be a better technique in reducing memory usage than tuning the batch size. While it is generally known that there exists a tradeoff between computation and memory requirements as reducing batch size increases training time, SUGAR is able to improve on both accounts.

5 Conclusion

We have proposed SUGAR, an efficient GNN training method that improves training scalability with multiple devices. SUGAR can reduce computation, memory and communication costs during training through two key contributions: (1) a novel graph partitioning strategy with memory budgets and graph topology taken into consideration; (2) subgraph-level local GNN training. We provided a thorough theoretical analysis of SUGAR and conducted extensive experiments to evaluate SUGAR. Experiments results across multiple hardware platforms demonstrate high training efficiency and memory scalability of SUGAR.

More importantly, SUGAR demonstrates the potential of deploying modern GNN algorithm on resource-limited devices, which opens up discussion in developing resource-efficient GNN approaches that are suitable for IoT deployment.

REFERENCES

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[3] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 31, pp. 5165–5175, 2018.

[4] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, 2019, pp. 417–426.

[5] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[6] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "Fedgnn: Federated graph neural network for privacy-preserving recommendation," 2021.

[7] H. Bagherinezhad, M. Rastegari, and A. Farhadi, "Lcnn: Lookup-based convolutional neural network," in *IEEE Conference on Computer Vision and Pattern Recognition*, July 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[9] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2021.

[10] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 1025–1035.

[11] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," 2018.

[12] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," 2018.

[13] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[14] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2020.

[15] C. R. Wolfe, J. Yang, A. Chowdhury, C. Dun, A. Bayer, S. Segarra, and A. Kyrillidis, "Gist: Distributed training for large-scale graph convolutional networks," 2021.

[16] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: distributed graph neural network training for billion-scale graphs," in *Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 2020, pp. 36–44.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[18] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," 2020.

[19] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[20] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, "Learning to drop: Robust graph neural network via topological denoising," in *International Conference on Web Search and Data Mining*, 2021, pp. 779–787.

[21] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *International Conference on Learning Representations*, 2020.

[22] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in gnns," 2020.

[23] J. Li, T. Zhang, H. Tian, S. Jin, M. Fardad, and R. Zafarani, "Sgcn: A graph sparsifier based on graph convolutional networks," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2020, pp. 275–287.

[24] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1695–1706.

[25] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[26] M. Kuczma, *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality*. Springer Science & Business Media, 2009.

[27] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

[29] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[30] P. W. Glynn and D. Ormoneit, "Hoeffding's inequality for uniformly ergodic markov chains," *Statistics & probability letters*, vol. 56, no. 2, pp. 143–146, 2002.

Zihui Xue Zihui Xue received the B.S. degree in School of Information Science and Technology, Fudan University, China, in 2020. She is currently pursuing the Ph.D. degree in Electrical and Computer Engineering department at The University of Texas at Austin. Her current research interests include multimodal perception, efficient machine learning and graph neural networks.

Yuedong Yang Yuedong Yang received the B.E. degree in School of Automation Science and Engineering from Xi'an Jiaotong University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree in Electrical and Computer Engineering department at The University of Texas at Austin. His current research interests include efficient machine learning algorithms, machine learning systems, and embedded systems.

Radu Marculescu Radu Marculescu is the Laura Jennings Turner Chair in Engineering and Professor in the Electrical and Computer Engineering department at The University of Texas at Austin. He received his Ph.D. in Electrical Engineering from the University of Southern California in 1998. Radu's current research focuses on machine learning methods and tools for modeling and optimization of embedded systems, cyber-physical systems, and social networks.

Appendix
Theoretical Analysis of SUGAR

In this appendix, we provide details of the following theoretical results used in Section 3:

(1) **Lemma 1**: For a multi-layer GCN with fixed weights, the error of the activations of the SG estimator are bounded.

(2) **Lemma 2**: For a multi-layer GCN with fixed weights, the error of the gradients of the SG estimator are bounded.

(3) **Theorem 1**: With high probability gradient descent training with the approximated gradients by the SG estimator can converge to a local minimum.

The proof builds on [12], but with different assumptions. More precisely, while [12] assume that model weights change slowly during training, our theoretical analysis is based on the difference in the adjacency matrices produced by graph partitioning.

.1 Notations

Let $[L] = \{1, ..., L\}$. The infinity norm of a matrix is defined as $\|A\|_\infty = \max_{i,j} |A_{i,j}|$. By Proposition B in [12], we know that:

1) $\|AB\|_\infty \leq col(A)\|A\|_\infty\|B\|_\infty$
2) $\|A \circ B\|_\infty \leq \|A\|_\infty\|B\|_\infty$
3) $\|A + B\|_\infty \leq \|A\|_\infty + \|B\|_\infty$

where $col(A)$ represents the number of columns of matrix A and $\circ$ is the element-wise product. We define η to be the maximum number of columns we can possibly encounter in the proof. We review some notations defined in the main text. Our proposed estimator is denoted by SG. The propagation rule of a l-th layer GCN with the exact estimator is given by:

$$Z^{(l+1)} = A^{norm}H^{(l)}W^{(l)}, H^{(l+1)} = \sigma(Z^{(l+1)}) \tag{14}$$

Similarly, the propagation rule of a l-th layer GCN with the SG estimator is given by:

$$Z_{SG}^{(l+1)} = A_{SG}^{norm}H_{SG}^{(l)}W^{(l)}, H_{SG}^{(l+1)} = \sigma(Z_{SG}^{(l+1)}) \tag{15}$$

where σ represents an activation function, A^{norm} denotes the normalized version of A, *i.e.*, $A^{norm} = \hat{D}^{-1/2}\hat{A}\hat{D}^{1/2}, \hat{A} = A + I_N, \hat{D}_{ii} = \sum_j \hat{A}_{ij}$ and I_N is an N-dimensional identity matrix.$H^{(l)}$ and $H_{SG}^{(l)}$ denote node representations in the l-th layer produced by the exact GCN and SG estimator, respectively. $W^{(l)}$ represents the weight matrix in layer l. Note that while we write Equation 15 in a compact matrix form, in real implementation, the training process is distributed across K devices.

Recall that A_{SG} is a block-diagonal matrix produced by the graph partitioning module that serves as an approximation of A. Before training, we run graph partitioning for M times to obtain a sample average, *i.e.*, $A_{SG}^{norm} = \frac{1}{M}\sum_{m=1}^{M} A_{SG,m}^{norm}$. Let $\epsilon = \|A_{SG}^{norm} - A^{norm}\|_\infty$ denote the error in approximating A^{norm} with A_{SG}^{norm}. For simplicity, we will omit the superscript $norm$ from now on.

The model parameters at training epoch t are denoted by W_t. For W at a given time point (*i.e.*, fixed model weights), we omit the subscript in the proof. Let W_* denote the optimal model weights. $\nabla\mathcal{L}(W) = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial f(y_i, z_i^{(L)})}{\partial W}$ and $\nabla\mathcal{L}_{SG}(W) = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial f(y_i, z_{SG,i}^{(L)})}{\partial W}$ represent the gradients of the exact GCN and SG estimator with respect to model weights W, respectively. $f(\cdot, \cdot)$ is the objective function (*e.g.*, cross entropy for node classification tasks).

.2 Activations of Multi-layer GCN

.2.1 Single-layer GCN

Proposition 2 states that for a single-layer GCN, (1) the outputs are bounded if the inputs are bounded, (2) if the difference between the input of the SG estimator and the exact GCN is small, then the output of the SG estimator is close to the output of the exact GCN.

Proposition 2. *For a one-layer GCN, if the activation function $\sigma(\cdot)$ is ρ-Lipschitz and $\sigma(0) = 0$, for any input matrices A, A_{SG}, X, X_{SG} and any weight matrix W that satisfy:*

1) *All the matrices are bounded by β: $\|A\|_\infty \leq \beta$, $\|A_{SG}\|_\infty \leq \beta$, $\|X\|_\infty \leq \beta$, $\|X_{SG}\|_\infty \leq \beta$ and $\|W\|_\infty \leq \beta$,*
2) *The differences between inputs are bounded: $\|X_{SG} - X\|_\infty \leq \alpha\epsilon$, where $\epsilon = \|A_{SG} - A\|_\infty$.*

Then, there exist B and C that depend on ρ, η and β, s.t.,

1) *The outputs are bounded: $\|H\|_\infty \leq B$ and $\|H_{SG}\|_\infty \leq B$,*
2) *The differences between outputs of the SG estimator and the exact estimator are bounded: $\|Z_{SG} - Z\|_\infty \leq C(1 + \alpha)\epsilon$ and $\|H_{SG} - H\|_\infty \leq C(1 + \alpha)\epsilon$.*

Proof. We know that $\|Z\|_\infty = \|AXW\|_\infty \leq \eta^2\|A\|_\infty\|X\|_\infty\|W\|_\infty \leq \eta^2\beta^3$. By Lipschitz continuity of $\sigma(\cdot)$, $\|\sigma(Z) - \sigma(0)\|_\infty \leq \rho\eta^2\beta^3$ and we have $\|\sigma(Z)\|_\infty \leq \rho\eta^2\beta^3$. Thus $\|H\|_\infty \leq D$, where $B = \max\{\eta^2\beta^3, \rho\eta^2\beta^3\}$. Similarly, $\|H_{SG}\|_\infty \leq B$.

We proceed to show that the differences between outputs are bounded below:

$$\begin{aligned}
&\|Z_{SG} - Z\|_\infty \\
&= \|A_{SG}X_{SG}W - AXW\|_\infty \\
&\leq \eta\|W\|_\infty\|A_{SG}X_{SG} - AX\|_\infty \\
&\leq \eta\beta(\|A_{SG}(X_{SG} - X)\|_\infty + \|X(A_{SG} - A)\|_\infty) \\
&\leq \eta\beta(\eta\beta\alpha\epsilon + \eta\beta\epsilon) \\
&= (1 + \alpha)\eta^2\beta^2\epsilon
\end{aligned} \tag{16}$$

By Lipschitz continuity of $\sigma(\cdot)$, we have $\|H_{SG} - H_t\|_\infty \leq \rho(1+\alpha)\eta^2\beta^2\epsilon$. Choose $C = \max\{(1+\alpha)\eta^2\beta^2, \rho(1+\alpha)\eta^2\beta^2\}$, and the proof is complete.

.2.2 Multi-layer GCN

The following lemma relates the approximation error in activations (*i.e.*, $\left\|H_{SG}^{(l)} - H^{(l)}\right\|_\infty$) with the approximation error in input adjacency matrices (*i.e.*, $\epsilon = \|A_{SG} - A\|_\infty$).

Lemma 1. *For a multi-layer GCN with fixed model weights, given a (fixed) graph dataset, assume that:*

1) *$\sigma(\cdot)$ is ρ-Lipschitz and $\sigma(0) = 0$,*
2) *The inputs are bounded by β: $\|A\|_\infty \leq \beta$, $\|A_{SG}\|_\infty \leq \beta$, $\|X\|_\infty \leq \beta$,*
3) *The model weights in each layer are bounded by β: $\left\|W^{(l)}\right\|_\infty \leq \beta, \forall l \in [L]$.*

Then, there exist B and C that depend on ρ, η and β, s.t.,

1) *$\left\|H^{(l)}\right\|_\infty \leq B, \left\|H_{SG}^{(l)}\right\|_\infty \leq B$, $\forall l \in [L-1]$,*
2) *$\left\|Z_{SG}^{(l)} - Z^{(l)}\right\|_\infty \leq C\epsilon$, $\forall l \in [L]$ and $\left\|H_{SG}^{(l)} - H^{(l)}\right\|_\infty \leq C\epsilon$, $\forall l \in [L-1]$.*

Proof. Applying Proposition 2 to each layer of the GCN proves that $H^{(l)}$ and $H_{SG}^{(l)}$ are bounded for each layer l.

For the first layer of GCN, by Proposition 2 and input conditions, we know that there exists $C^{(1)}$ that satisfies:

$$\left\| Z_{SG}^{(1)} - Z^{(1)} \right\|_\infty \leq C^{(1)}\epsilon, \quad \left\| H_{SG}^{(1)} - H^{(1)} \right\|_\infty \leq C^{(1)}\epsilon$$

Note that for the first layer, the node feature matrix of the SG estimator and exact GCN are identical, *i.e.*, $X_{SG} = X$; this yields $\alpha = 0$ in Equation 16.

Let $\hat{C}^{(1)} = C^{(1)}$. Next, we apply Proposition 2 to the second layer of GCN: there exists $C^{(2)}$ that satisfies: $\left\| Z_{SG}^{(2)} - Z^{(2)} \right\|_\infty \leq C^{(2)}(1 + \hat{C}^{(1)})\epsilon, \left\| H_{SG}^{(2)} - H^{(2)} \right\|_\infty \leq C^{(2)}(1 + \hat{C}^{(1)})\epsilon$.

Let $\hat{C}^{(2)} = C^{(2)}(1 + \hat{C}^{(1)})$. By applying Proposition 2 to the subsequent layer of GCN repetitively, we have $\hat{C}^{(l+1)} = C^{(l+1)}(1 + \hat{C}^{(l)}), \forall l \in [L-1]$. We choose $C = \max_l \hat{C}^{(l)}$ and complete the proof.

.3 Gradients of Multi-layer GCN

Lemma 2 below provides a bound for the difference between gradients of the loss by the SG estimator and the exact GCN (*i.e.*, $\|\nabla \mathcal{L}_{SG}(W) - \nabla \mathcal{L}(W)\|_\infty$). Intuitively, the gradient difference is small if the approximation error in input adjacency matrices (*i.e.*, ϵ) is small.

Lemma 2. *For a multi-layer GCN with fixed model weights, given a (fixed) graph dataset, assume that:*

1) $\frac{\partial f(y,z)}{\partial z}$ *is* ρ*-Lipschitz and* $\left\| \frac{\partial f(y,z)}{\partial z} \right\|_\infty \leq \beta$,
2) $\sigma(\cdot)$ *is* ρ*-Lipschitz,* $\sigma(0) = 0$ *and* $\|\sigma'(\cdot)\|_\infty \leq \beta$,
3) $\|A\|_\infty \leq \beta$, $\|A_{SG}\|_\infty \leq \beta$, $\|X\|_\infty \leq \beta$, $\left\| W^{(l)} \right\|_\infty \leq \beta, \forall l \in [L]$.

Then, there exists C *that depends on* ρ*,* η *and* β*, s.t.,* $\|\nabla \mathcal{L}_{SG}(W) - \nabla \mathcal{L}(W)\|_\infty \leq C\epsilon$.

Proof. We begin by proving the following statements:

If the above assumptions hold, then there exist C *and* D *that depends on* ρ*,* η *and* β*, s.t.,*

1) *The gradients with respect to the activations of each layer of the SG estimator are close to be unbiased:*

$$\left\| \frac{\partial f}{\partial Z_{SG}^{(l)}} - \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty \leq C\epsilon, \quad \forall l \in [L] \tag{17}$$

2) *The gradients above are bounded:*

$$\left\| \frac{\partial f}{\partial Z_{SG}^{(l)}} \right\|_\infty \leq D\beta, \quad \left\| \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty \leq D\beta, \quad \forall l \in [L] \tag{18}$$

We prove these statements by induction. First we show that Equations 17 and 18 hold true for the final layer of GCN (*i.e.*, $l = L$). By Assumption 1 and Lemma 1, we know that there exists $\hat{C}$ that satisfies:

$$\left\| \frac{\partial f}{\partial Z_{SG}^{(L)}} - \frac{\partial f}{\partial Z^{(L)}} \right\|_\infty \leq \rho \left\| Z_{SG}^{(L)} - Z^{(L)} \right\|_\infty \leq \rho\hat{C}\epsilon \tag{19}$$

Let $C^{(L)} = \rho\hat{C}$ and $D^{(L)} = 1$. Next, suppose the statements hold for layer $l+1$, *i.e.*, there exist $C^{(l+1)}$ and $D^{(l+1)}$ that satisfy:

$$\begin{aligned} \left\| \frac{\partial f}{\partial Z_{SG}^{(l+1)}} - \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty &\leq C^{(l+1)}\epsilon, \\ \left\| \frac{\partial f}{\partial Z_{SG}^{(l+1)}} \right\|_\infty &\leq D^{(l+1)}\beta, \\ \left\| \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty &\leq D^{(l+1)}\beta \end{aligned} \tag{20}$$

We derive the gradients of the objective function with respect to activations in layer l by chain rule:

$$\begin{aligned} \left\| \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty &= \left\| \sigma'(Z^{(l)}) \circ \frac{\partial f}{\partial H^{(l)}} \right\|_\infty \\ &= \left\| \sigma'(Z^{(l)}) \circ A^T \ \frac{\partial f}{\partial Z^{(l+1)}} \ {W^{(l)}}^T \right\|_\infty \\ &\leq \eta^2 \left\| \sigma'(Z^{(l)}) \right\|_\infty \|A\|_\infty \left\| \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty \left\| W^{(l)} \right\|_\infty \\ &\leq \eta^2 \beta^4 D^{(l+1)} \end{aligned} \tag{21}$$

Thus, we know that $\left\| \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty \leq D^{(l)}\beta$. Similarly, $\left\| \frac{\partial f}{\partial Z_{SG}^{(l)}} \right\|_\infty \leq D^{(l)}\beta$, where $D^{(l)} = \eta^2\beta^3 D^{(l+1)}$.

We proceed to derive the error of the gradients by the SG estimator in layer l:

$$\begin{aligned} & \left\| \frac{\partial f}{\partial Z_{SG}^{(l)}} - \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty \\ & \leq \eta \left\| W^{(l)} \right\|_\infty \| \sigma'(Z_{SG}^{(l)}) \circ A_{SG}^T \ \frac{\partial f}{\partial Z_{SG}^{(l+1)}} \\ & \quad - \sigma'(Z^{(l)}) \circ A^T \ \frac{\partial f}{\partial Z^{(l+1)}} \|_\infty \\ & \leq \underbrace{\eta\beta \left\| (\sigma'(Z_{SG}^{(l)}) - \sigma'(Z^{(l)})) \circ A_{SG}^T \frac{\partial f}{\partial Z_{SG}^{(l+1)}} \right\|_\infty}_{(*)} \\ & \quad \underbrace{+ \eta\beta \left\| \sigma'(Z^{(l)}) \circ A_{SG}^T (\frac{\partial f}{\partial Z_{SG}^{(l+1)}} - \frac{\partial f}{\partial Z^{(l+1)}}) \right\|_\infty}_{(**)} \\ & \quad \underbrace{+ \eta\beta \left\| \sigma'(Z^{(l)}) \circ (A_{SG}^T - A^T) \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty}_{(***)} \end{aligned} \tag{22}$$

By Assumption 2 and Lemma 1, we know that there exists $\hat{C}$ such that $\left\| \sigma'(Z_{SG}^{(l)}) - \sigma'(Z^{(l)}) \right\|_\infty \leq \rho\hat{C}\epsilon$. From Equation

20, we have:

$$\begin{aligned}
&(*) \text{ in Eq. (22)} \\
&\leq \eta^2\beta \left\| \sigma'(Z_{SG}^{(l)}) - \sigma'(Z^{(l)}) \right\|_\infty \|A_{SG}\|_\infty \left\| \frac{\partial f}{\partial Z_{SG}^{(l+1)}} \right\|_\infty \\
&\leq \eta^2\beta \cdot \rho\hat{C}\epsilon \cdot \beta \cdot D^{(l+1)}\beta \\
&= (\eta^2\beta^3\rho\hat{C}D^{(l+1)})\epsilon \\
&(**) \text{ in Eq. (22)} \\
&\leq \eta^2\beta \left\| \sigma'(Z^{(l)}) \right\|_\infty \|A_{SG}\|_\infty \left\| \frac{\partial f}{\partial Z_{SG}^{(l+1)}} - \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty \\
&\leq \eta^2\beta \cdot \beta \cdot \beta \cdot C^{(l+1)}\epsilon \\
&= (\eta^2\beta^3 C^{(l+1)})\epsilon \\
&(***) \text{ in Eq. (22)} \\
&\leq \eta^2\beta \left\| \sigma'(Z^{(l)}) \right\|_\infty \left\| A_{SG}^T - A^T \right\|_\infty \left\| \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty \\
&\leq \eta^2\beta \cdot \beta \cdot \epsilon \cdot D^{(l+1)}\beta \\
&= (\eta^2\beta^3 D^{(l+1)})\epsilon
\end{aligned} \tag{23}$$

Therefore, $\left\| \frac{\partial f}{\partial Z_{SG}^{(l)}} - \frac{\partial f}{\partial Z^{(l)}} \right\|_\infty \leq C^{(l)}\epsilon$, where $C^{(l)} = \eta^2\beta^3[(\rho\hat{C}+1)D^{(l+1)} + C^{(l+1)}]$. By induction, Equations 17 and 18 hold true.

Next, we show below that there exists C that depends on ρ, η and β, *s.t.*,

$$\left\| \frac{\partial f}{\partial W_{SG}^{(l)}} - \frac{\partial f}{\partial W^{(l)}} \right\|_\infty \leq C\epsilon, \quad \forall l \in [L] \tag{24}$$

By backpropagation rule we derive that $\frac{\partial f}{\partial W^{(l)}} = (AH^{(l)})^T \frac{\partial f}{\partial Z^{(l)}}$. By Lemma 1, we know that $H_{SG}^{(l)}$ is bounded by some $\hat{B}$ and $\left\| H_{SG}^{(l)} - H^{(l)} \right\|_\infty \leq \tilde{C}\epsilon$ hold for some $\tilde{C}$. From the previous proof, we know that there exists $\hat{C}$ and $\hat{D}$, *s.t.*, Equations 17 and 18 hold; thus, we have:

$$\begin{aligned}
&\left\| \frac{\partial f}{\partial W_{SG}^{(l)}} - \frac{\partial f}{\partial W^{(l)}} \right\|_\infty \\
&\leq \left\| (A_{SG}H_{SG}^{(l)})^T \frac{\partial f}{\partial Z_{SG}^{(l+1)}} - (AH^{(l)})^T \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty \\
&\leq \left\| (A_{SG}H_{SG}^{(l)})^T (\frac{\partial f}{\partial Z_{SG}^{(l+1)}} - \frac{\partial f}{\partial Z^{(l+1)}}) \right\|_\infty \\
&\quad + \left\| ((A_{SG}H_{SG}^{(l)})^T - (AH^{(l)})^T) \frac{\partial f}{\partial Z^{(l+1)}} \right\|_\infty \\
&\leq \eta^2\beta \cdot \hat{B} \cdot \hat{C}\epsilon + \eta \left\| A_{SG}H_{SG}^{(l)} - AH^{(l)} \right\|_\infty \hat{D}\beta \\
&\leq \eta^2\beta\hat{B}\hat{C}\epsilon + \eta\beta\hat{D} \left\| A_{SG}(H_{SG}^{(l)} - H^{(l)}) \right\|_\infty \\
&\quad + \eta\beta\hat{D} \left\| (A_{SG} - A)H^{(l)} \right\|_\infty \\
&\leq \eta^2\beta\hat{B}\hat{C}\epsilon + \eta\beta\hat{D} \cdot \eta\beta \cdot \tilde{C}\epsilon + \eta\beta\hat{D} \cdot \eta\epsilon \cdot \hat{B} \\
&= \eta^2\beta(\hat{B}\hat{C} + \beta\tilde{C}\hat{D} + \hat{B}\hat{D})\epsilon
\end{aligned} \tag{25}$$

Therefore, Equation 24 holds, where $C = \eta^2\beta(\hat{B}\hat{C} + \beta\tilde{C}\hat{D} + \hat{B}\hat{D})$.

Finally, we have: $\|\nabla\mathcal{L}_{SG}(W) - \nabla\mathcal{L}(W)\|_\infty \leq C\epsilon$, and the proof is complete.

.4 Convergence Analysis

Theorem 1. *Assume that:*

1) *The loss function $\mathcal{L}(W)$ is ρ-smooth, i.e., $|\mathcal{L}(W_2) - \mathcal{L}(W_1) - \langle\mathcal{L}(W_1), W_2 - W_1\rangle| \leq \frac{\rho}{2}\|W_2 - W_1\|_F^2, \forall W_1, W_2$, where $\langle A, B\rangle = tr(A^TB)$ denotes the inner product of matrix A and B,*
2) *The gradients of the loss $\nabla\mathcal{L}(W)$ and $\nabla\mathcal{L}_{SG}(W)$ are bounded by G for any choice of W,*
3) *The gradient of the objective function $\frac{\partial f(y,z)}{\partial z}$ is ρ-Lipschitz and bounded,*
4) *The activation function $\sigma(\cdot)$ is ρ-Lipschitz, $\sigma(0) = 0$ and $\sigma'(\cdot)$ is bounded.*

Then there exists $C > 0$, s.t., $\forall M, T$, for a sufficiently small δ, if we run graph partitioning for M times and run gradient descent for $R \leq T$ epochs (where R is chosen uniformly from $[T]$, the model update rule is $W_{t+1} = W_t - \gamma\nabla\mathcal{L}_{SG}(W_t)$, step size $\gamma = \frac{1}{\rho\sqrt{T}}$), we have:

$$\begin{aligned}
&P(\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \leq \delta) \\
&\geq 1 - 2\exp\{-2M(\frac{\delta}{2C} - \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C - \delta}{2C(\sqrt{T}-1)})^2\}
\end{aligned}$$

Proof. Let $\delta_t = \nabla\mathcal{L}_{SG}(W_t) - \nabla\mathcal{L}(W_t)$ denote the differences between gradients at epoch t. By ρ-smoothness of $\mathcal{L}(W)$ we know that:

$$\begin{aligned}
&\mathcal{L}(W_{t+1}) \\
&\leq \mathcal{L}(W_t) + \langle\nabla\mathcal{L}(W_t), W_{t+1} - W_t\rangle + \frac{\rho}{2}\gamma^2\|\nabla\mathcal{L}_{SG}(W_t)\|_F^2 \\
&= \mathcal{L}(W_t) - \gamma\langle\nabla\mathcal{L}(W_t), \nabla\mathcal{L}_{SG}(W_t)\rangle + \frac{\rho}{2}\gamma^2\|\nabla\mathcal{L}_{SG}(W_t)\|_F^2 \\
&= \mathcal{L}(W_t) - \gamma\langle\nabla\mathcal{L}(W_t), \delta_t\rangle - \gamma\|\nabla\mathcal{L}(W_t)\|_F^2 \\
&\quad + \frac{\rho}{2}\gamma^2[\|\delta_t\|_F^2 + \|\nabla\mathcal{L}(W_t)\|_F^2 + 2\langle\delta_t, \nabla\mathcal{L}(W_t)\rangle] \\
&= \mathcal{L}(W_t) - (\gamma - \rho\gamma^2)\langle\nabla\mathcal{L}(W_t), \delta_t\rangle \\
&\quad - (\gamma - \frac{\rho}{2}\gamma^2)\|\nabla\mathcal{L}(W_t)\|_F^2 + \frac{\rho}{2}\gamma^2\|\delta_t\|_F^2
\end{aligned} \tag{26}$$

By Lemma 2, we know that at a given time point t, there exists $\hat{C}$ *s.t.*, δ_t is bounded by $\hat{C}\epsilon$. Therefore,

$$\begin{aligned}
&|\langle\nabla\mathcal{L}(W_t), \delta_t\rangle| \leq \eta\|\nabla\mathcal{L}(W_t)\|_\infty\|\delta_t\|_\infty \leq \eta G\hat{C}\epsilon \\
&\|\delta_t\|_F^2 \leq \|\nabla\mathcal{L}_{SG}(W_t)\|_\infty^2 + \|\nabla\mathcal{L}(W_t)\|_\infty^2 \leq 2G^2
\end{aligned} \tag{27}$$

Let $C = \max\{\eta G\hat{C}, 2G^2\}$. Equation 26 can be further derived as:

$$\begin{aligned}
\mathcal{L}(W_{t+1}) \leq \mathcal{L}(W_t) &+ (\gamma - \rho\gamma^2)C\epsilon \\
&- (\gamma - \frac{\rho}{2}\gamma^2)\|\nabla\mathcal{L}(W_t)\|_F^2 + \frac{\rho}{2}C\gamma^2
\end{aligned} \tag{28}$$

By summing up the above inequalities from $t = 1$ to T and rearranging the terms, we have:

$$\begin{aligned}
(\gamma - \frac{\rho}{2}\gamma^2)\sum_t \|\nabla\mathcal{L}(W_t)\|_F^2 &\leq \mathcal{L}(W_1) - \mathcal{L}(W_*) \\
&+ CT(\gamma - \rho\gamma^2)\epsilon + \frac{\rho}{2}CT\gamma^2
\end{aligned} \tag{29}$$

Dividing both sides of Equation 29 by $T(\gamma - \frac{\rho}{2}\gamma^2)$ and choosing $\gamma = \frac{1}{\rho\sqrt{T}}$ gives us:

$$\begin{aligned}
&\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \\
&\leq 2\frac{\mathcal{L}(W_1) - \mathcal{L}(W_*) + CT(\gamma - \rho\gamma^2)\epsilon + \frac{\rho}{2}CT\gamma^2}{T\gamma(2-\rho\gamma)} \\
&\leq \frac{2[\mathcal{L}(W_1) - \mathcal{L}(W_*)]}{T\gamma} + 2C(1-\rho\gamma)\epsilon + \rho C\gamma \\
&\leq \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)]}{\sqrt{T}} + 2C(1 - \frac{1}{\sqrt{T}})\epsilon + \frac{C}{\sqrt{T}} \\
&\leq \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C}{\sqrt{T}} + 2C(1 - \frac{1}{\sqrt{T}})\epsilon
\end{aligned} \tag{30}$$

Recall that ϵ denotes the infinity norm of the error in approximating A through M runs, *i.e.*, $\epsilon = \|A_{SG} - A\|_\infty$. Applying Hoeffding's inequality [30] to the largest element of the matrix $|A_{SG} - A|$ (which are bounded by the intervals $[0,1]$), we have:

$$P(\epsilon \geq \delta) \leq 2\exp(-2M\delta^2), \quad \forall\delta \geq 0 \tag{31}$$

Combining the two inequalities above, we have:

$$\begin{aligned}
&P(\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \geq \delta) \\
&\leq P(\frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C}{\sqrt{T}} + 2C(1 - \frac{1}{\sqrt{T}})\epsilon \geq \delta) \\
&\leq 2\exp\{-2M(\frac{\delta}{2C} - \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C - \delta}{2C(\sqrt{T}-1)})^2\}
\end{aligned} \tag{32}$$

Therefore, for a sufficiently small δ, we have the following inequality for $P(\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \leq \delta)$:

$$\begin{aligned}
&P(\mathbb{E}_R\|\nabla\mathcal{L}(W_R)\|_F^2 \leq \delta) \\
&\quad \geq 1 - 2\exp\{-2M(\frac{\delta}{2C} - \frac{2\rho[\mathcal{L}(W_1) - \mathcal{L}(W_*)] + C - \delta}{2C(\sqrt{T}-1)})^2\}
\end{aligned} \tag{33}$$

Theorem 1 is proved.