GIST: Distributed Training for Large-Scale Graph Convolutional Networks

Cameron R. Wolfe[*1], **Jingkang Yang**[*2], **Arindam Chowdhury**[3], **Chen Dun**[1], **Artun Bayer**[3], **Santiago Segarra**[3], **and Anastasios Kyrillidis**[1]

[1]Department of Computer Science, Rice University, Houston, TX, USA.
[2]School of Computer Science and Engineering, Nanyang Technology University, Singapore.
[3]Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA.

Abstract

The graph convolutional network (GCN) is a go-to solution for machine learning on graphs, but its training is notoriously difficult to scale both in terms of graph size and the number of model parameters. Although some work has explored training on large-scale graphs (e.g., GraphSAGE, ClusterGCN, etc.), we pioneer efficient training of large-scale GCN models (i.e., ultra-wide, overparameterized models) with the proposal of a novel, distributed training framework. Our proposed training methodology, called `GIST`, disjointly partitions the parameters of a GCN model into several, smaller sub-GCNs that are trained independently and in parallel. In addition to being compatible with any GCN architecture, `GIST` improves model performance, scales to training on arbitrarily large graphs, significantly decreases wall-clock training time, and enables the training of markedly overparameterized GCN models. Remarkably, with `GIST`, we train an astonishgly-wide 32,768-dimensional GraphSAGE model, which exceeds the capacity of a single GPU by a factor of $8\times$, to SOTA performance on the Amazon2M dataset.

1 Introduction

Since not all data can be represented in Euclidean space [5], many applications rely on graph-structured data. For example, social networks can be modeled as graphs by regarding each user as a node and friendship relations as edges [28, 31]. Alternatively, in chemistry, molecules can be modeled as graphs, with nodes representing atoms and edges encoding chemical bonds [2, 4].

To better understand graph-structured data, several (deep) learning techniques have been extended to the graph domain [11, 14, 29]. Currently, the most popular one is the graph convolutional network (GCN) [23], a multi-layer architecture that implements a generalization of the convolution operation to graphs. Although the GCN handles node and graph-level classification, it is notoriously inefficient and unable to handle large-scale graphs [7, 8, 12, 19, 42, 45].

To deal with these issues, node partitioning methodologies have been developed. These schemes can be roughly categorized into neighborhood sampling [8, 16, 50] and graph partitioning [9, 45] approaches. The goal is to partition a large graph into multiple smaller graphs that can be used as mini-batches for training the GCN. In this way, GCNs can handle larger graphs during training, expanding their potential into the realm of big data.

Although some papers perform large-scale experiments [9, 45], the models (and data) used in GCN research remain small in the context of deep learning [23, 41], where the current trend is towards incredibly large models and datasets [6, 10]. Despite the widespread moral questioning of this trend

[*]Equal Contribution.

Preprint. Under review.

[17, 34, 36], the deep learning community continues to push the limits of scale, as overparameterized models are known to discover generalizable solutions [30]. Although deep GCN models suffer from oversmoothing [23, 25], overparameterized GCN models can still be explored through experiments with larger hidden layers. *As such, this work aims to provide a training framework that enables GCN experiments with wider models and larger datasets.*

This paper. We propose a novel, distributed training methodology that can be used for any GCN architecture and is compatible with existing node sampling techniques. This methodology randomly partitions the hidden feature space in each layer, decomposing the global GCN model into multiple, narrow sub-GCNs of equal depth. Sub-GCNs are trained independently for several iterations in parallel prior to having their updates synchronized. This process of randomly partitioning, independently training, and synchronizing sub-GCNs is repeated until convergence. We call this method graph independent subnetwork training (`GIST`). By performing graph partitioning on larger datasets, `GIST` can easily scale to arbitrarily large graphs. `GIST` significantly reduces the wall-clock time of large-scale GCN experiments, allowing larger models and datasets to be explored. We focus specifically on enabling the training of "ultra-wide" GCNs (i.e., GCN models with very large hidden layers), as deeper GCNs are prone to oversmoothing [25]. The contributions of this work are summarized below:

- We develop a novel, distributed training methodology for GCN architectures, based on decomposing the model into independently-trained sub-GCNs.
- We show that `GIST` can be used to train several GCN architectures to state-of-the-art performance with reduced wall-clock time in comparison to standard training methodologies.
- We use `GIST` to enable the training of markedly overparameterized GCN models. In particular, `GIST` is used to train a **two-layer GraphSAGE model with a hidden dimension of 32,768** on the Amazon2M dataset. *Such a model exceeds the capacity of a single GPU by* $8\times$.

2 What is the `GIST` of this work?

Algorithm 1 `GIST` Algorithm

```
Parameters: $T$ synchronization iterations, $m$ sub-GCNs, $\zeta$ local iterations, $c$ clusters, $\mathcal{G}$ training graph.
$\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}) \leftarrow$ randomly initialize GCN
$\{\mathcal{G}_{(j)}\}_{j=1}^{c} \leftarrow$ Cluster$(\mathcal{G}, c)$
for $t = 0, \dots, T-1$ do
    $\{\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}^{(i)})\}_{i=1}^{m} \leftarrow$ subGCNs$(\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}), m)$
    Distribute each $\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}^{(i)})$ to a different worker
    for $i = 1, \dots, m$ do
        for $z = 1, \dots, \zeta$ do
            $\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}^{(i)}) \leftarrow$ subTrain$(\mathbf{\Theta}^{(i)}, \{\mathcal{G}_{(j)}\}_{j=1}^{c})$
        end for
    end for
    $\Psi_{\mathcal{G}}(\,\cdot\,;\mathbf{\Theta}) \leftarrow$ subAgg$(\{\mathbf{\Theta}^{(i)}\}_{i=1}^{m})$
end for
```

GCN Architecture. The GCN [23] is arguably the most widely-used neural network architecture on graphs. Consider a graph $\mathcal{G}$ comprised of n nodes with d-dimensional features $\mathbf{X} \in \mathbb{R}^{n \times d}$. The output $\mathbf{Y} \in \mathbb{R}^{n \times d'}$ of a GCN can be expressed as $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}; \mathbf{\Theta})$, where $\Psi_{\mathcal{G}}$ is an L-layered architecture with trainable parameters $\mathbf{\Theta}$. If we define $\mathbf{H}_0 = \mathbf{X}$, we then have that $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}; \mathbf{\Theta}) = \mathbf{H}_L$, where an intermediate ℓ-th layer of the GCN is given by

$$\mathbf{H}_{\ell+1} = \sigma(\bar{\mathbf{A}}\,\mathbf{H}_\ell\,\mathbf{\Theta}_\ell). \tag{1}$$

In (1), σ is an elementwise activation function (e.g., ReLU), $\bar{\mathbf{A}}$ is the degree-normalized adjacency matrix of $\mathcal{G}$ with added self-loops, and the trainable parameters $\mathbf{\Theta} = \{\mathbf{\Theta}_\ell\}_{\ell=0}^{L-1}$ have dimensions $\mathbf{\Theta}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ with $d_0 = d$ and $d_L = d'$. In Figure 1 (top), we illustrate nested GCN layers for $L = 3$, but our methodology extends to arbitrary L. The activation function of the last layer is typically the identity or softmax transformation – we omit this in Figure 1 for simplicity.

`GIST` **overview.** We overview `GIST` in Algorithm 1 and present a schematic depiction in Figure 2. We partition our (randomly initialized) global GCN into m smaller, disjoint sub-GCNs with the `subGCNs` function ($m = 2$ in Figures 1 and 2) by sampling the feature space at each layer of the GCN; see Section 2.1. Each sub-GCN is assigned to a different worker (i.e., a different GPU) for ζ rounds of distributed, independent training through `subTrain`. Then, newly-learned sub-GCN parameters are aggregated (`subAgg`) into the global GCN model. This process repeats for T iterations. Our graph

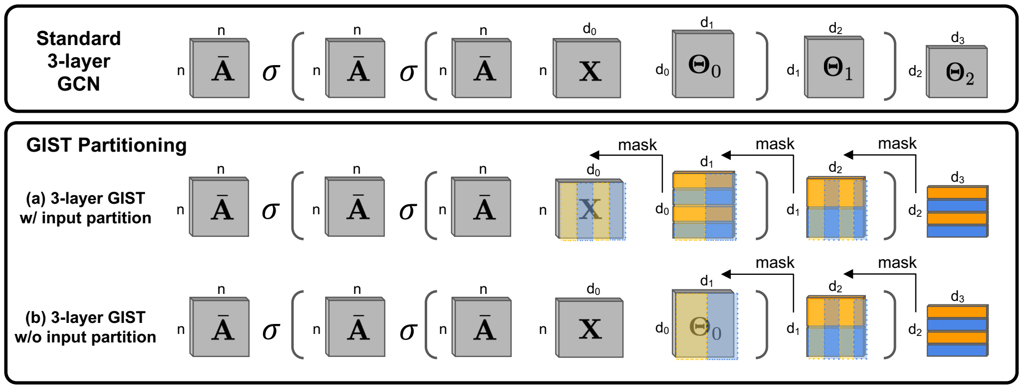
Standard 3-layer GCN
GIST Partitioning
(a) 3-layer GIST w/ input partition
(b) 3-layer GIST w/o input partition
mask
n
d0
d1
d2
d3
Ā
X
σ
Θ0
Θ1
Θ2

Figure 1: GCN partition into $m = 2$ sub-GCNs. Orange and blue colors depict different feature partitions. Both hidden dimensions (d_1 and d_2) are partitioned. The output dimension (d_3) is not partitioned. Partitioning the input dimension (d_0) is optional. **We do not partition $\mathbf{d}_0$ in** `GIST`.

domain is partitioned into c sub-graphs through the `Cluster` function ($c = 2$ in Figure 2). This operation is only relevant for large graphs ($n >$ 50,000), and we omit it ($c = 1$) for smaller graphs that don't require partitioning.[2]

2.1 `subGCNs`: Constructing Sub-GCNs

`GIST` partitions a global GCN model into several narrower sub-GCNs of equal depth. Formally, consider an arbitrary layer ℓ and a random, disjoint partition of the feature set $[d_\ell] = \{1, 2, \ldots, d_\ell\}$ into m equally-sized blocks $\{\mathcal{D}_\ell^{(i)}\}_{i=1}^m$.[3] Accordingly, we denote by $\mathbf{\Theta}_\ell^{(i)} = [\mathbf{\Theta}_\ell]_{\mathcal{D}_\ell^{(i)} \times \mathcal{D}_{\ell+1}^{(i)}}$ the matrix obtained by selecting from $\mathbf{\Theta}_\ell$ the rows and columns given by the ith blocks in the partitions of $[d_\ell]$ and $[d_{\ell+1}]$, respectively. With this notation in place, we can define m different sub-GCNs $\mathbf{Y}^{(i)} = \Psi_{\mathcal{G}}(\mathbf{X}^{(i)}; \mathbf{\Theta}^{(i)}) = \mathbf{H}_L^{(i)}$ where $\mathbf{H}_0^{(i)} = \mathbf{X}_{[n] \times \mathcal{D}_0^{(i)}}$ and each layer is given by:

$$\mathbf{H}_{\ell+1}^{(i)} = \sigma(\bar{\mathbf{A}}\, \mathbf{H}_\ell^{(i)}\, \mathbf{\Theta}_\ell^{(i)}). \tag{2}$$

Sub-GCN partitioning is illustrated in Figure 1-(a), where $m = 2$. Partitioning the input features is optional (i.e., (a) vs. (b) in Figure 1). *We do not partition the input features within `GIST` so that* sub-GCNs have identical input information (i.e., $\mathbf{X}^{(i)} = \mathbf{X}$ for all i); see Section 4.1. Similarly, we do not partition the output feature space to ensure that the sub-GCN output dimension coincides with that of the global model, thus avoiding any need to modify the loss function. This decomposition procedure (`subGCNs` in Algorithm 1) extends to arbitrary L.

2.2 `subTrain`: Independently Training Sub-GCNs

Assume $c = 1$ so that the `Cluster` operation in Algorithm 1 is moot and $\{\mathcal{G}_{(j)}\}_{j=1}^c = \mathcal{G}$. Because $\mathbf{Y}^{(i)}$ and $\mathbf{Y}$ share the same dimension, sub-GCNs can be trained to minimize the same global loss function. One application of `subTrain` in Algorithm 1 corresponds to a single step of stochastic gradient descent (SGD). Inspired by local SGD [27], multiple, independent applications of `subTrain` are performed in parallel (i.e., on separate GPUs) for each sub-GCN prior to aggregating weight updates.[4] The number of independent training iterations between synchronization rounds, referred to as local iterations, is denoted by ζ, and the total amount of training is split across sub-GCNs.[5] Ideally, the number sub-GCNs and local iterations should be increased as much as possible to minimize

[2]Though any clustering method can be used, we advocate the use of METIS [20, 21] due to its proven efficiency in large-scale graphs.

[3]For example, if $d_\ell = 4$ and $m = 2$, one valid partition would be given by $\mathcal{D}_\ell^{(1)} = \{1, 4\}$ and $\mathcal{D}_\ell^{(2)} = \{2, 3\}$.

[4]The number of sub-GCNs must be less than or equal to the number of GPUs available for parallel training.

[5]For example, if a global model is trained on a single GPU for 10 epochs, a comparable experiment for `GIST` with two sub-GCNs would train each sub-GCN for only 5 epochs.

communication and training costs. In practice, however, such benefits may come at the cost of statistical inefficiency; see Section 4.1.

If $c > 1$, subTrain first selects one of the c subgraphs in $\{\mathcal{G}_{(j)}\}_{j=1}^{c}$ to use as a mini-batch for SGD. Alternatively, the union of several sub-graphs in $\{\mathcal{G}_{(j)}\}_{j=1}^{c}$ can be used as a mini-batch for training. Aside from using mini-batches for each SGD update instead of the full graph, the use of graph partitioning does not modify the training approach outlined above. This pipeline, which allows our methodology to scale to arbitrarily large graphs, is illustrated in Figure 2 for $c = 2$ clusters and $m = 2$ sub-GCNs. Pre-processing the training graph into smaller sub-graphs during training loosely resembles the ClusterGCN [9] methodology. However, *the focus of this paper is upon developing a distributed training methodology, not developing novel graph partitioning strategies.* We simply adopt existing graph partitioning methods to ease scalability to larger graphs.

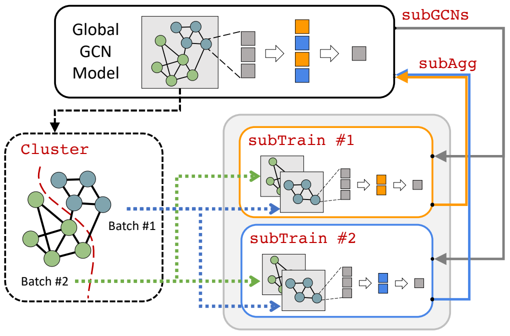

Figure 2: Pipeline for GIST. subGCNs divides the global GCN into several sub-GCNs. Every sub-GCN is trained by subTrain using mini-batches (smaller sub-graphs) generated by Cluster. Sub-GCN parameters are intermittently aggregated through subAgg.

2.3 subAgg: Synchronizing Sub-GCNs

After each sub-GCN completes ζ training iterations, their updates are aggregated into the global model (i.e., subAgg function in Algorithm 1). Within subAgg, each worker replaces global parameter entries $\mathbf{\Theta}$ with its own parameters $\mathbf{\Theta}^{(i)}$, where no collisions occur due to the disjointness of sub-GCN partitions. Interestingly, not every parameter in the global GCN model is updated by subAgg. For example, focusing on $\mathbf{\Theta}_1$ in Figure 1-(a), one worker will be assigned $\mathbf{\Theta}_1^{(1)}$ (i.e., overlapping orange blocks), while the other worker will be assigned $\mathbf{\Theta}_1^{(2)}$ (i.e., overlapping blue blocks). The rest of $\mathbf{\Theta}_1$ is not considered within subAgg. Nonetheless, since sub-GCN partitions are randomly drawn in each cycle t, one expects all of $\mathbf{\Theta}$ to be updated multiple times if T is sufficiently large.

2.4 What is the value of GIST?

Architecture-Agnostic Distributed Training. GIST is a generic, distributed training methodology that can be used for any GCN architecture. We implement GIST for vanilla GCN, GraphSAGE, and GAT architectures, but GIST is not limited to these models; see Section 4.

Compatibility with Sampling Methods. GIST's feature partitioning strategy is compatible with node partitioning methodologies, which can yield further benefits in training efficiency. For example, GIST is combined with graph partitioning strategies in Section 2.2. Similarly, layer sampling methodologies [7, 8, 50] are compatible with GIST; see experiments with GraphSAGE in Section 4.2.

Enabling Ultra-Wide GCN Training. GIST indirectly updates the global GCN through the training of smaller sub-GCNs, enabling models with hidden dimensions that exceed the capacity of a single GPU by a factor of $8\times$ to be trained. In this way, GIST allows markedly overparametrized ("ultra-wide") GCN models to be trained on existing hardware. In Section 4.2, *we leverage this capability to train a two-layer GCN model with a hidden dimension of 32,768 on Amazon2M.* Although GIST does not explicitly enable the training of deeper GCNs, *we argue that overparameterization through width is more valuable because deeper GCNs suffer from oversmoothing* [25].

Improved Model Complexity. Consider a single GCN layer, trained over M machines with input and output dimension of d_{i-1} and d_i, respectively. For one synchronization round, the communication complexity of GIST and standard distributed training is $O(\frac{1}{M}d_i d_{i-1})$ and $O(M d_i d_{i-1})$, respectively. GIST reduces communication by only communicating sub-GCN parameters. Existing node partitioning techniques cannot similarly reduce communication complexity because model parameters are never partitioned. Furthermore, the computational complexity of the forward pass for a GCN model trained with GIST and using standard methodology is $O(\frac{1}{M}N^2 d_i + \frac{1}{M^2}N d_i d_{i-1})$ and

$O(N^2 d_i + N d_i d_{i-1})$, respectively, where N is the number of nodes in the partition being processed.[6] Node partitioning can reduce N by a constant factor but is compatible with GIST.

2.5 Implementation Details

We provide an implementation of GIST in PyTorch [32] using the NCCL distributed communication package for training GCN [23], GraphSAGE [16] and GAT [41] architectures. Our implementation is centralized, meaning that a single process serves as a central parameter server. From this central process, the weights of the global model are maintained and partitioned to different worker processes (including itself) for independent training. Experiments are conducted with 8 NVIDIA Tesla V100-PCIE-32G GPUs, a 56-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, and 256 GB of RAM.

3 Related Work

GCN training. In spite of their widespread success in several graph related tasks, GCNs often suffer from training inefficiencies [12, 19]. Consequently, the research community has focused on developing efficient and scalable algorithms for training GCNs [7, 8, 9, 16, 45, 50]. The resulting approaches can be divided roughly into two areas: *neighborhood sampling* and *graph partitioning*. However, it is important to note that these two broad classes of solutions are not mutually exclusive, and reasonable combinations of the two approaches may be beneficial.

Neighborhood sampling methodologies aim to sub-select neighboring nodes at each layer of the GCN, thus limiting the number of node representations in the forward pass and mitigating the exponential expansion of the GCNs receptive field. VRGCN [7] implements a variance reduction technique to reduce the sample size in each layer, which achieves good performance with smaller graphs. However, it requires to store all the intermediate node embeddings during training, leading to a memory complexity close to full-batch training. GraphSAGE [16] learns a set of aggregator functions to gather information from a node's local neighborhood. It then concatenates the outputs of these aggregation functions with each node's own representation at each step of the forward pass. FastGCN [8] adopts a Monte Carlo approach to evaluate the GCN's forward pass in practice, which computes each node's hidden representation using a fixed-size, randomly-sampled set of nodes. LADIES [50] introduces a layer-conditional approach for node sampling, which encourages node connectivity between layers in contrast to FastGCN [8].

Graph partitioning schemes aim to select densely-connected sub-graphs within the training graph, which can be used to form mini-batches during GCN training. Such sub-graph sampling reduces the memory footprint of GCN training, thus allowing larger models to be trained over graphs with many nodes. ClusterGCN [9] produces a very large number of clusters from the global graph, then randomly samples a subset of these clusters and computes their union to form each sub-graph or mini-batch. Similarly, GraphSAINT [45] randomly samples a sub-graph during each GCN forward pass. However, GraphSAINT also considers the bias created by unequal node sampling probabilities during sub-graph construction, and proposes normalization techniques to eliminate this bias.

As explained in Section 2, GIST also relies on graph partitioning techniques (Cluster) to handle large graphs. However, the feature sampling scheme at each layer (subGCNs) that leads to parallel and narrower sub-GCNs is a hitherto unexplored framework for efficient GCN training.

Distributed training. Distributed training is a heavily studied topic [37, 47]. Our work focuses on synchronous and distributed training techniques [26, 43, 46]. Some examples of synchronous, distributed training approaches include data parallel training, parallel SGD [1, 49], and local SGD [27, 39]. Our methodology holds similarities to model parallel training techniques, which have been heavily explored [3, 13, 15, 24, 33, 40, 48]. More closely, our approach is inspired by independent subnetwork training [44], explored for multi-layer perceptrons.

[6]We omit the complexity of applying the element-wise activation function for simplicity.

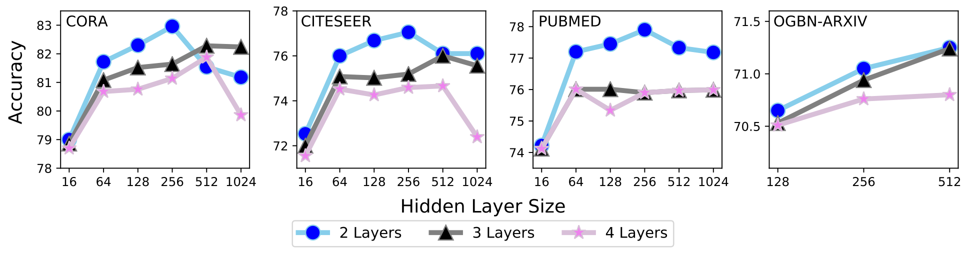

Figure 3: Test accuracy for different sizes (i.e., varying depth and width) of GCN models trained with standard, single-GPU methodology on small-scale datasets.

4 Experiments

We use `GIST` to train several GCN architectures on six public multi-node classification datasets; see Table 1. We adopt standard training, validation, and testing splits. In all experiments, we compare the performance of models trained with `GIST` to that of models trained with standard (single-GPU) methods. Comparisons of `GIST` to other distributed training methodologies (e.g., local SGD [27] and ensembles of GCNs) are provided in the supplementary material. Cora, Citeseer, Pubmed, and OGBN-Arxiv are considered "small-scale" datasets and are used to run low-cost ablation experiments; see Section 4.1. Reddit and Amazon2M are considered "large-scale" datasets. F1 score and training time are used to measure the performance of models trained with `GIST` on both of these datasets; see Section 4.2. For large-scale datasets, the goal of `GIST` is to i) train GCN models to state-of-the-art performance, ii) minimize wall-clock training time, and iii) enable training of very large GCN models.

Table 1: Details of relevant datasets.

Dataset	n	# Edges	# Labels	d
Cora [35]	2,708	5,429	7	1,433
CiteSeer [35]	3,312	4,723	6	3,703
Pubmed [35]	19,717	44,338	3	500
OGBN-Arxiv [18]	169,343	1.2M	40	128
Reddit [16]	232,965	11.6 M	41	602
Amazon2M [9]	2.5 M	61.8 M	47	100

4.1 Small-Scale Experiments

In this section, we present several numerical experiments conducted over the Cora, Citeseer, Pubmed, and OGBN-Arxiv datasets [35, 18]. Because these experiments run quickly, our small-scale analysis focuses on the impact of different design and hyperparameter choices on the performance of GCN models trained with `GIST`, rather than attempting to improve runtime (i.e., speeding up such short experiments is futile). Experiments are run for 400 epochs with a step learning rate schedule (i.e., $10\times$ decay at 50% and 75% of total epochs). A vanilla GCN model, as described in [23], is used. The model is trained in a full-batch manner using the Adam optimizer [22]. All reported results are averaged across five trials with different random seeds. For all models, d_0 and d_L are respectively given by the number of features and output classes in the dataset. The size of all hidden layers is the same, but may vary across experiments.

Single-GPU Models. We first train baseline models of different depths and hidden dimensions using standard, single-GPU methodology. The results are shown in Figure 3. Deeper models do not yield performance improvements for small-scale datasets, but test accuracy improves as the model becomes wider. Based upon the results in Figure 3, we adopt as our *baseline* a three-layer GCN model with a hidden dimension of $d_1 = d_2 = 256$ for all small-scale experiments.[7]

Which layers should be partitioned? We investigate whether models trained with `GIST` are sensitive to the partitioning of certain layers. In particular, although the output dimension d_3 is never partitioned, we selectively partition dimensions d_0, d_1, and d_2 to observe the impact on model performance; see Table 2. Partitioning input features (d_0) significantly degrades test accuracy, which becomes more noticeable with larger m. For example, partitioning d_0 decreases model test accuracy from 79.58% to 48.32% on Cora when $m = 8$. Intuitively, this performance decrease occurs because each sub-GCN observes only a portion of node input features (i.e., each sub-GCN has d_0/m-dimensional input). Aside from the input layer, all GCN *hidden* layers (i.e., dimensions d_1 and d_2 in this case)

[7]Though two-layer models seem to perform best, we use a three-layer model as our baseline to enable more flexibility in examining the partitioning strategy of `GIST`.

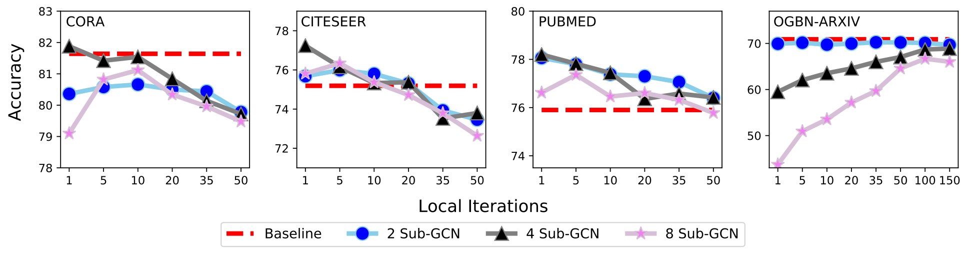

Figure 4: Test accuracy of GCN models trained with GIST using different numbers of local iterations and sub-GCNs on small-scale datasets. The performance of the same model trained with standard, single-GPU methodology is provided for reference. *Models trained with GIST are surprisingly robust to the number of local iterations used during training, no matter the number of sub-GCNs.*

Table 2: Test accuracy of a 3-layer GCN of width 256 trained with GIST. We selectively partition each feature dimension within the GCN model, indicated by a check mark. *Results show that partitioning on all hidden layers except the input layer can lead to the optimal performance.*

# Sub-GCNs	d_0	d_1	d_2	Cora	Citeseer	Pubmed	OGBN-Arxiv
Baseline				81.52 ± 0.005	75.02 ± 0.018	75.90 ± 0.003	70.85 ± 0.089
2	✓	✓	✓	80.00 ± 0.010	**75.95** ± 0.007	76.68 ± 0.011	65.65 ± 0.700
	✓	✓		78.30 ± 0.011	69.34 ± 0.018	75.78 ± 0.015	65.33 ± 0.347
		✓	✓	**80.82** ± 0.010	75.82 ± 0.008	**78.02** ± 0.007	**70.10** ± 0.224
4	✓	✓	✓	76.78 ± 0.017	70.66 ± 0.011	65.67 ± 0.044	54.21 ± 1.360
	✓	✓		66.56 ± 0.061	68.38 ± 0.018	68.44 ± 0.014	52.64 ± 1.988
		✓	✓	**81.18** ± 0.007	**76.21** ± 0.017	**76.99** ± 0.006	**68.69** ± 0.579
8	✓	✓	✓	48.32 ± 0.087	45.42 ± 0.092	54.29 ± 0.029	40.26 ± 1.960
	✓	✓		53.60 ± 0.020	54.68 ± 0.030	51.44 ± 0.002	26.84 ± 7.226
		✓	✓	**79.58** ± 0.006	**75.39** ± 0.016	**76.99** ± 0.006	**65.81** ± 0.378

can be partitioned without degrading model performance. Therefore, GIST adopts the strategy of partitioning all dimensions other than d_0 and d_L (i.e., input and output dimensions); see Figure 1-(b).

How many Sub-GCNs to use? Using more sub-GCNs during GIST training typically improves runtime because sub-GCNs i) become smaller, ii) are trained for fewer epochs each, and iii) are trained in parallel. We find that all models trained with GIST perform similarly for practical settings of m; see Table 2. In fact, model performance often improves as the number of sub-GCNs is increased. Therefore, when training a model with GIST, one may continue increasing the number sub-GCNs until all GPUs are occupied or model performance begins to decrease. The latter effect is more noticeable in large-scale experiments; see Tables 3 and 4.

Incorporating Local Iterations. GIST dictates that sub-GCNs be trained for ζ independent, local iterations between synchronization rounds [27]. Although increasing ζ reduces communication for a fixed number of epochs, this may come at the cost of degraded model performance. We train models using GIST with different settings for ζ; see Figure 4. The performance of GIST is relatively robust to the number of local iterations, but test accuracy decreases slightly as ζ increases. For small-scale datasets, using $\zeta = 20$ performs consistently well, but ζ can be further increased (e.g., $\zeta = 500$ or $\zeta = 5000$) on large-scale datasets without noticeable performance deterioration; see Section 4.2. Additionally, larger values of ζ, such as $\zeta = 100$, seem to perform best on OGBN-Arxiv.

GIST **Performance.** We observe that models trained with GIST often exceed the performance of models trained with standard, single-GPU methodology; see Figure 4 and Table 2. Therefore, training GCN models with GIST yields performance benefits *in terms of both speed and accuracy.* Intuitively, we hypothesize that the random feature partitioning within GIST, which loosely resembles dropout [38], provides regularization benefits during training. However, we leave an in-depth analysis of the performance benefits derived from GIST as future work.

Table 3: Performance of GraphSAGE and GAT models trained with `GIST` on Reddit. *`GIST` significantly accelerates the training of both GCN models and sometimes even leads to better accuracy.*

L	# Sub-GCNs	GraphSAGE			GAT		
		F1 Score	Time	Speedup	F1 Score	Time	Speedup
2	Baseline	96.09	105.78s	1.00×	89.57	4289.80s	1.00×
	2	96.40	70.29s	1.50×	90.28	2097.16s	2.05×
	4	96.16	68.88s	1.54×	90.02	1112.33s	3.86×
	8	95.46	76.68s	1.38×	89.01	640.44s	6.70×
3	Baseline	96.32	118.37s	1.00×	89.25	7241.23s	1.00×
	2	96.36	80.46s	1.47×	89.63	3432.80s	2.11×
	4	95.76	78.74s	1.50×	88.82	1727.73s	4.19×
	8	94.39	88.54s	1.34×	70.38	944.21s	7.67×
4	Baseline	96.32	120.74s	1.00×	88.36	9969.13s	1.00×
	2	96.01	91.75s	1.32×	87.97	4723.55s	2.11×
	4	95.21	78.74s	1.53×	78.42	2376.21s	4.21×
	8	92.75	88.71s	1.36×	66.30	1262.28s	7.90×

4.2 Large-Scale Experiments

Reddit Dataset. For the Reddit dataset, we perform tests with 256-dimensional GraphSAGE [16] and GAT [41] models with two to four layers. Models trained with standard, single-GPU methodology are used as baselines, and compared to models trained with `GIST` in terms of F1 score and training time. All tests are run for 80 epochs with no weight decay, using the Adam optimizer [22]. We find that $\zeta = 500$ achieves consistently high performance for models trained with `GIST` on Reddit. The training graph is partitioned into 15,000 sub-graphs during training and a batch size of 10 is used.

As shown in Table 3, utilizing `GIST` significantly accelerates GCN training, yielding a $1.32\times$ to $7.90\times$ speedup. `GIST` performs best in terms of F1 score with $m = 2$ sub-GCNs. Although $m = 4$ reduces training time, the F1 score also decreases slightly. Interestingly, the speedup provided by `GIST` is more significant for models and datasets with larger compute requirements. For example, the speedup achieved on GAT experiments is more significant than the speedup achieved on GraphSAGE experiments, which becomes especially noticeable for larger m (i.e., `GIST` provides a near-linear speedup with respect to m when training the GAT model). Additionally, larger speedups are observed when training GraphSAGE models with `GIST` on the Amazon2M dataset; see Table 4.

Amazon2M Dataset. We follow the experimental settings of [9]. Experiments are performed with two, three, and four-layer GraphSAGE models [16] with hidden dimensions of 400 and 4096 (we refer to these models as "narrow" and "wide", respectively). We compare the performance (i.e., F1 score and wall-clock training time) of GCN models trained with standard, single-GPU methodology to that of models trained with `GIST`. The training graph is partitioned into 15,000 sub-graphs and a batch size of 10 is used. We find that using $\zeta = 5000$ performs consistently well. Models are trained for 400 total epochs with the Adam optimizer [22] and no weight decay.

Results on the Amazon2M dataset are given in Table 4. Narrow models trained with `GIST` have a lower F1 score in comparison to the baseline, but training time is significantly reduced. For example, when $L = 4$, the narrow GCN trained with `GIST` achieves an F1 score 1.84 below the baseline with $1.68\times$ lower runtime. For wider models, `GIST` provides a more significant speedup (i.e., up to $7.12\times$). The F1 performance of models trained with `GIST` also improves as the model becomes wider, revealing that `GIST` works best when used to train wider models. Interestingly, although GCN models trained using `GIST` with eight sub-GCNs perform worse than the baseline, the baseline models take significantly longer to achieve equal F1 score. For example, when $L = 2$, a wide GCN trained with `GIST` ($m = 8$) reaches an F1 score of 88.86 in ~4,000 seconds, *while models trained with standard methodology take ~10,000 seconds to achieve a comparable F1 score.*

4.3 Training Ultra-Wide GCNs

To illustrate the power of our proposed methodology, we leverage `GIST` to train models of shocking scale (i.e., "ultra-wide" models) on the Amazon2M dataset. Adopting the settings of Section 4.2, **we use `GIST` to train GraphSAGE models of varying widths as high as 32K, which exceeds the**

Table 4: Performance of GraphSAGE models trained with `GIST` on the Amazon2M dataset. *Models trained with `GIST` train more quickly and achieve comparable F1 score to those trained with standard methodology. The performance benefits of `GIST` become more pronounced for wider models.*

L	# Sub-GCNs	$d_i = 400$			$d_i = 4096$		
		F1 Score	Time	Speedup	F1 Score	Time	Speedup
2	Baseline	89.90	1.81hr	1.00×	91.25	5.17hr	1.00×
	2	88.36	1.25hr	1.45×	90.70	1.70hr	3.05×
	4	86.33	1.11hr	1.63×	89.49	1.13hr	4.57×
	8	84.73	1.13hr	1.61×	88.86	1.11hr	4.65×
3	Baseline	90.36	2.32hr	1.00×	91.51	9.52hr	1.00×
	2	88.59	1.56hr	1.49×	91.12	2.12hr	4.49×
	4	86.46	1.37hr	1.70×	89.21	1.42hr	6.72×
	8	84.76	1.37hr	1.69×	86.97	1.34hr	7.12×
4	Baseline	90.40	3.00hr	1.00×	91.61	14.20hr	1.00×
	2	88.56	1.79hr	1.68×	91.02	2.77hr	5.13×
	4	87.53	1.58hr	1.90×	89.07	1.65hr	8.58×
	8	85.32	1.56hr	1.93×	87.53	1.55hr	9.13×

Table 5: Performance of GraphSAGE models of different widths trained with `GIST` on Amazon2M. We report sub-graph F1 score (i.e., F1 evaluated over graph partitions) to avoid memory overflow during evaluation. Experiments marked with "OOM" cause an out-of-memory error during training. *By training models with `GIST`, we enable the use of ultra-wide models that achieve improved accuracy.*

L	# Sub-GCNs	F1 Score (Time)				
		$d_i = 400$	$d_i = 4096$	$d_i = 8192$	$d_i = 16384$	$d_i = 32768$
2	Baseline	89.38 (1.81hr)	90.58 (5.17hr)	OOM	OOM	OOM
	2	87.48 (1.25hr)	90.09 (1.70hr)	90.87 (2.76hr)	90.94 (9.31hr)	90.91 (32.31hr)
	4	84.82 (1.11hr)	88.79 (1.13hr)	89.76 (1.49hr)	90.10 (2.24hr)	90.17 (5.16hr)
	8	82.56 (1.13hr)	87.16 (1.11hr)	88.31 (1.20hr)	88.89 (1.39hr)	89.46 (1.76hr)
3	Baseline	89.73 (2.32hr)	90.99 (9.52hr)	OOM	OOM	OOM
	2	87.79 (1.56hr)	90.40 (2.12hr)	90.91 (4.87hr)	91.05 (17.7hr)	OOM
	4	85.30 (1.37hr)	88.51 (1.42hr)	89.75 (2.07hr)	90.15 (3.44hr)	OOM
	8	82.84 (1.37hr)	86.12 (1.34hr)	88.38 (1.37hr)	88.67 (1.88hr)	88.66 (2.56hr)
4	Baseline	89.77 (3.00hr)	91.02 (14.20hr)	OOM	OOM	OOM
	2	87.75 (1.79hr)	90.36 (2.77hr)	91.08 (6.92hr)	91.09 (26.44hr)	OOM
	4	85.32 (1.58hr)	88.50 (1.65hr)	89.76 (2.36hr)	90.05 (4.93hr)	OOM
	8	83.45 (1.56hr)	86.60 (1.55hr)	88.13 (1.61hr)	88.44 (2.30hr)	OOM

capacity of a single GPU by $8\times$. For $d_i > 4096$, evaluation must be performed on graph partitions (not the full graph) to avoid memory overflow. As such, the graph is partitioned into 5,000 sub-graphs during testing and F1 score is measured over each partition and averaged.

The performane of ultra-wide models is reported in Table 5. Without `GIST`, the best-performing two-layer GraphSAGE model is of dimension $d_i = 4096$, which achieves an F1 score of 90.58 in 5.2 hours. With `GIST` ($m = 2$), however, we achieve a higher F1 score of 90.87 in 2.8 hours (i.e., a $1.86\times$ speedup) using $d_i = 8192$. A model of such width cannot be trained with standard methodology, and larger hidden dimensions yield further improvements. Similar patterns are observed for deeper models trained with `GIST`. For example, the four-layer, 8192-dimensional GraphSAGE model trained with `GIST` ($m = 2$) achieves an F1 score of 91.08 in 7 hours – achieving similar F1 score with a 4196-dimensional model trained with standard methodology takes over 14 hours. However, we focus mainly on increasing model width, as deeper GCN models often suffer from oversmoothing [25].

By training GraphSAGE models with dimensions up to 32K, we find that larger hidden dimensions tend to yield better performance, revealing that wide, overparameterized GCN models are useful for large-scale datasets such as Amazon2M. The ability to train a model of such scale (i.e., orders of magnitude wider than existing work) to reach state-of-the-art performance on the Amazon2M dataset affirms the ability of `GIST` to enable large-scale experiments on graphs.

5 Conclusion

We present GIST, a distributed training approach for GCNs that enables the exploration of larger models and datasets. GIST is compatible with existing sampling approaches and leverages a feature-wise partition of model parameters to construct smaller sub-GCNs that are trained independently and in parallel. We have shown that GIST achieves remarkable speed-ups over large graph datasets and even enables the training of GCN models of unprecedented size. We hope GIST can empower the exploration of larger, more powerful GCN architectures within the graph community.

Acknowledgements

AK acknowledges funding by the NSF (CNS-2003137). This work was partially done as CW's and JY's class project for "COMP414: Optimization - Algorithms, Complexity and Approximations," Rice University, Fall 2020.

References

[1] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[2] Alexandru T Balaban. Applications of Graph Theory in Chemistry. *Journal of Chemical Information and Computer Sciences*, 1985.

[3] Tal Ben-Nun and Torsten Hoefler. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *ACM Computing Surveys (CSUR)*, 2019.

[4] Gil Benkö, Christoph Flamm, and Peter F Stadler. A graph-based toy model of chemistry. *Journal of Chemical Information and Computer Sciences*, 2003.

[5] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 2017.

[6] Tom B. Brown et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[7] Jianfei Chen, Jun Zhu, and Le Song. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[8] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

[10] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised Cross-lingual Representation Learning at Scale. *arXiv preprint arXiv:1911.02116*, 2019.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

[12] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-Scale Learnable Graph Convolutional Networks. *arXiv preprint arXiv:1808.03965*, 2018.

[13] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. Integrated Model, Batch and Domain Parallelism in Training Neural Networks. *arXiv preprint arXiv:1712.04432*, 2017.

[14] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2005.

[15] S. Günther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger. Layer-Parallel Training of Deep Residual Neural Networks. *arXiv preprint arXiv:1812.04352*, 2018.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[17] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes, June 2019.

[18] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv e-prints*, page arXiv:2005.00687, May 2020.

[19] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive Sampling Towards Fast Graph Representation Learning. *arXiv preprint arXiv:1809.05343*, 2018.

[20] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.

[21] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 1998.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, 2016.

[24] Andrew C. Kirby, Siddharth Samsi, Michael Jones, Albert Reuther, Jeremy Kepner, and Vijay Gadepally. Layer-Parallel Training with GPU Concurrency of Deep Residual Neural Networks via Nonlinear Multigrid. *arXiv preprint arXiv:2007.07336*, 2020.

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[26] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[27] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't Use Large Mini-Batches, Use Local SGD. *arXiv preprint arXiv:1808.07217*, 2018.

[28] Dean Lusher, Johan Koskinen, and Garry Robins. *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press, 2013.

[29] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2015.

[30] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep Double Descent: Where Bigger Models and More Data Hurt. *arXiv preprint arXiv:1912.02292*, 2019.

[31] Mark EJ Newman, Duncan J Watts, and Steven H Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 2002.

[32] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[33] J. Gregory Pauloski, Zhao Zhang, Lei Huang, Weijia Xu, and Ian T. Foster. Convolutional Neural Network Training with Distributed K-FAC. *arXiv preprint arXiv:2007.00784*, 2020.

[34] Tony Peng and Michael Sarazen. The staggering cost of training sota ai models, June 2019.

[35] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29:93–93, 2008.

[36] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

[37] Shaohuai Shi, Zhenheng Tang, Xiaowen Chu, Chengjian Liu, Wei Wang, and Bo Li. A Quantitative Survey of Communication Optimizations in Distributed Deep Learning. *arXiv preprint arXiv:2005.13247*, 2020.

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014.

[39] Sebastian U. Stich. Local SGD converges fast and communicates little. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[40] Sanket Tavarageri, Srinivas Sridharan, and Bharat Kaul. Automatic Model Parallelism for Deep Neural Networks with Compiler and Hardware Support. *arXiv preprint arXiv:1906.08168*, 2019.

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

[42] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2127–2135, 2020.

[43] Kwangmin Yu, Thomas Flynn, Shinjae Yoo, and Nicholas D'Imperio. Layered sgd: A decentralized and synchronous sgd algorithm for scalable deep neural network training. *arXiv preprint arXiv:1906.05936*, 2019.

[44] Binhang Yuan, Anastasios Kyrillidis, and Christopher M. Jermaine. Distributed Learning of Deep Neural Networks using Independent Subnet Training. *arXiv preprint arXiv:1810.01392*, 2019.

[45] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. *arXiv preprint arXiv:1907.04931*, 2019.

[46] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[47] Z. Zhang, L. Yin, Y. Peng, and D. Li. A quick survey on large scale distributed deep learning systems. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018.

[48] Wentao Zhu, Can Zhao, Wenqi Li, Holger Roth, Ziyue Xu, and Daguang Xu. LAMP: Large Deep Nets with Automated Model Parallelism for Image Segmentation. *arXiv preprint arXiv:2006.12575*, 2020.

[49] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, pages 2595–2603, 2010.

[50] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. *arXiv preprint arXiv:1911.07323*, 2019.

A Comparison between Other Methods

A.1 Comparison to Local SGD

The major benefit of GIST arises from its communication-efficient methodology for disjointly partitioning the weights of a single, large GCN model across multiple machines for distributed training. A simple version of local SGD [27] could also be implemented for distributed training of GCNs by training the full model on each separate worker for a certain number of local iterations and intermittently averaging local updates. Similarly to GIST, this method would allow GCN training to be distributed across multitple GPUs, thus enabling some larger-scale experiments to be explored. However, GIST has superior communication-efficiency in comparison to local SGD, as only a fraction of model parameters are communicated to each machine. Furthermore, because each sub-GCN is smaller than the global model, GIST achieves an additional speedup during the local training of each sub-GCN in comparison to locally training the full model. To verify the efficiency benefits of GIST, we perform a direct comparison to local SGD over the Reddit dataset using a 2-layer GCN model with a hidden dimension of 256.

Table A1: Training metrics for two-layer GCN models with hidden dimension 256 trained using local SGD and `GIST` on the Reddit dataset. All models are trained with 100 local iterations for both `GIST` and local SGD.

# Machines	Method	F1 Score	Training Time
2	Local SGD	96.37	137.17s
	`GIST`	**96.40**	**108.67s**
4	Local SGD	95.00	127.63s
	`GIST`	**96.16**	**116.56s**
8	Local SGD	93.40	129.58s
	`GIST`	**95.46**	**123.83s**

`GIST` and local SGD are compared directly within Table A1. As can be seen, the combined effect of communicating fewer parameters and conducting training on smaller models allow `GIST` to accelerate training in comparison to local SGD. For example, in two machine setting, `GIST` completes training 20% faster than local SGD, even while achieving slightly improved test accuracy. In addition to providing a speedup in comparison to local SGD, `GIST` also provides higher test accuracy in all cases. Therefore, it is clear from these experiments that the `GIST` algorithm provides performance benefits on multiple fronts and is a viable alternative to local SGD for the distributed training of GCN models.

A.2 Comparison to Ensembles of Sub-GCNs

Table A2: Training metrics for two-layer GCN models with hidden dimension of 256 on the Reddit dataset. Models are trained both with `GIST` and as ensembles of shallow sub-GCNs. The ensembles of shallow sub-GCNs are formed by partitioning the global model into sub-GCNs once, training them independently, and never aggregating their parameters into the global model.

# Machines	Method	F1 Score	Inference Time (s)
2	Ensemble	96.31	3.59
	`GIST`	**96.40**	**1.81**
4	Ensemble	96.10	6.38
	`GIST`	**96.16**	**1.81**
8	Ensemble	95.28	11.95
	`GIST`	**95.46**	**1.81**

As previously mentioned, increasing the number of local iterations (i.e., ζ in Algorithm 1) decreases communication requirements given a fixed amount of training, thus yielding an acceleration. When taken to the extreme (i.e., as $\zeta \to \infty$), one could minimize communication requirements by independently training sub-GCNs and never aggregating their parameters into the global model, thus forming an ensemble of sub-GCNs. To test whether such an ensemble is a viable alternative to `GIST`, the performance of such ensembles of sub-GCNs, in comparison to models trained with `GIST`, is presentetd in Table A2 for the Reddit dataset.[8] As can be seen, although training ensembles of sub-GCNs may minimize communication during training, such an approach yields inferior performance in comparison to `GIST`. Furthermore, because an entire ensemble of GCN models must be maintained, the inference time of the resulting model is significantly increased, which becomes more extreme as the number of sub-GCNs is increased (i.e., because more models exist within the ensemble).

B Full Results on Reddit dataset

Here, we present all experimental settings that were tested on the Reddit dataset. These results are listed in Table A3. For each experiment, we include the optimal learning rate that was used to achieve

[8]For each sub-GCN, we measure validation accuracy throughout training and add the highest-performing model into the ensemble.

Table A3: Performance of all experimental settings for 256-dimensional GraphSAGE [16] models trained with `GIST` on the Reddit dataset. All `GIST` experiments used weight decay of 0 and were trained for 80 epochs in total.

# Layers	ζ	# Sub-GCNs	LR	Best Test F1	Time (s)
2	100	2	0.01	96.43	109.57
		4	0.01	95.97	116.38
		8	0.01	94.41	123.14
	500	2	0.01	96.40	81.44
		4	0.005	96.16	78.75
		8	0.01	95.46	90.49
	1000	2	0.005	96.42	77.28
		4	0.01	96.20	73.86
		8	0.01	95.24	80.67
	1500	2	0.01	96.31	77.24
		4	0.01	96.10	71.34
		8	0.01	95.24	87.52
3	100	2	0.01	96.34	128.19
		4	0.01	95.33	140.45
		8	0.001	92.47	149.79
	500	2	0.005	96.36	95.41
		4	0.005	95.76	89.25
		8	0.01	94.39	97.82
	1000	2	0.005	96.39	91.01
		4	0.01	95.90	85.04
		8	0.01	95.01	90.59
	1500	2	0.005	96.18	86.46
		4	0.005	95.94	80.92
		8	0.01	95.01	91.67
4	100	2	0.005	96.13	145.27
		4	0.01	94.17	162.10
		8	0.01	84.07	177.06
	500	2	0.005	96.01	105.88
		4	0.005	95.21	99.92
		8	0.01	92.75	108.11
	1000	2	0.005	96.32	99.94
		4	0.005	95.73	96.63
		8	0.01	92.26	100.11
	1500	2	0.005	96.11	95.18
		4	0.01	95.80	92.39
		8	0.01	92.26	96.78

the recorded results. Baseline experiments are excluded because all relevant baseline results are provided in the main text.

C Full Results on Small Datasets

Here, we present several tables which contain the full results for all experimental settings that were tested on small-scale datasets (i.e., CORA, CITESEER, and PUBMED). These results are provided in Tables A4, A5, A6, and A7. For each experiment, the optimal learning rate, which yielded top-performing results, is also listed. In the main text, the metric that is reported is the best test accuracy, but in these tables we also list the validation and final test accuracies for completeness.

Table A4: Performance of single-GPU baseline models of different sizes on small-scale datasets.

Dataset	Depth	Width	LR	Best Test Acc.	Best Val Acc.	Last Test Acc.
CORA	2	16	0.05	79.00	77.40	75.40
	2	64	0.05	81.72	80.92	78.56
	2	128	0.05	82.30	80.96	79.36
	2	256	0.05	82.96	80.84	80.54
	2	512	0.05	81.54	79.88	79.44
	2	1024	0.01	81.18	80.32	79.94
	3	16	0.05	78.86	77.12	76.28
	3	64	0.05	81.08	79.24	76.70
	3	128	0.05	81.52	79.36	77.56
	3	256	0.01	81.64	79.48	78.60
	3	512	0.01	82.28	80.16	78.30
	3	1024	0.01	82.24	80.52	79.78
	4	16	0.1	78.70	77.68	75.92
	4	64	0.005	80.68	78.64	75.08
	4	128	0.01	80.76	79.64	76.98
	4	256	0.005	81.14	80.32	78.44
	4	512	0.005	81.88	79.80	77.62
	4	1024	0.005	79.86	78.16	76.62
CITESEER	2	16	0.05	72.53	71.46	67.50
	2	64	0.1	76.00	75.50	72.62
	2	128	0.05	76.68	76.24	72.87
	2	256	0.05	77.05	76.02	74.13
	2	512	0.05	76.10	75.78	74.14
	2	1024	0.05	76.10	75.88	73.94
	3	16	0.1	72.00	72.28	68.77
	3	64	0.05	75.08	74.30	70.75
	3	128	0.01	75.02	75.62	72.18
	3	256	0.005	75.19	75.32	73.16
	3	512	0.01	76.00	75.10	72.14
	3	1024	0.05	75.56	74.86	72.33
	4	16	0.1	71.55	70.86	69.10
	4	64	0.005	74.52	74.06	68.36
	4	128	0.01	74.26	73.88	70.67
	4	256	0.005	74.59	74.46	71.81
	4	512	0.005	74.66	73.42	71.09
	4	1024	0.005	72.39	71.96	69.50
PUBMED	2	16	0.05	74.22	73.65	69.79
	2	64	0.1	77.20	76.79	74.29
	2	128	0.1	77.45	77.23	75.09
	2	256	0.05	77.90	77.33	75.07
	2	512	0.1	77.33	76.99	75.52
	2	1024	0.05	77.18	77.12	75.11
	3	16	0.1	74.12	74.44	71.37
	3	64	0.1	76.01	75.68	72.63
	3	128	0.05	76.01	75.95	73.35
	3	256	0.05	75.90	76.07	73.63
	3	512	0.05	75.97	75.84	72.87
	3	1024	0.01	75.99	76.07	73.53
	4	16	0.1	74.12	74.44	71.37
	4	64	0.1	76.01	75.68	72.63
	4	128	0.005	75.34	75.57	72.59
	4	256	0.05	75.90	76.07	73.63
	4	512	0.05	75.97	75.84	72.87
	4	1024	0.01	75.99	76.07	73.53

Table A5: Performance of narrow three-layer GCN model (i.e., hidden dimension of 64) trained with `GIST` on small-scale datasets. These tests do not split the input layer, but they do split the output layer.

Dataset	ζ	# Sub-GCN	LR	Best Test Acc.	Best Val. Acc.	Last Test Acc.
CORA	1	2	0.1	80.24	78.92	76.64
	1	4	0.01	80.56	79.24	78.34
	1	8	0.01	76.34	74.60	69.98
	5	2	0.005	80.18	79.04	76.50
	5	4	0.01	79.34	77.56	77.86
	5	8	0.005	74.14	72.00	72.86
	10	2	0.05	80.06	79.00	77.80
	10	4	0.01	78.76	76.80	77.86
	10	8	0.005	73.22	72.20	73.18
	20	2	0.005	79.84	78.44	77.02
	20	4	0.005	79.24	77.68	78.84
	20	8	0.005	75.88	74.76	75.26
	30	2	0.05	79.44	78.88	78.14
	30	4	0.005	79.16	77.92	78.12
	30	8	0.05	76.16	74.40	60.72
CITESEER	1	2	0.05	76.28	76.10	70.87
	1	4	0.01	76.63	75.70	73.16
	1	8	0.01	74.82	73.44	70.23
	5	2	0.1	75.96	75.42	71.15
	5	4	0.005	75.55	74.98	73.72
	5	8	0.005	72.61	71.50	71.74
	10	2	0.1	76.11	74.96	73.43
	10	4	0.01	75.37	74.16	73.31
	10	8	0.005	71.94	71.08	71.92
	20	2	0.01	74.82	73.64	70.90
	20	4	0.1	74.99	74.20	74.10
	20	8	0.005	72.75	71.98	72.43
	30	2	0.1	74.62	74.06	73.01
	30	4	0.05	73.93	73.36	72.68
	30	8	0.005	71.76	70.80	71.65
PUBMED	1	2	0.1	77.02	77.20	69.97
	1	4	0.05	76.61	76.67	73.75
	1	8	0.01	74.32	73.93	70.37
	5	2	0.1	76.27	76.20	72.60
	5	4	0.01	75.99	75.76	74.90
	5	8	0.01	72.57	72.05	70.75
	10	2	0.1	76.31	75.71	73.80
	10	4	0.01	75.62	75.03	73.73
	10	8	0.005	70.97	70.80	70.79
	20	2	0.01	75.41	75.00	72.08
	20	4	0.1	75.40	74.83	74.43
	20	8	0.005	72.63	72.55	71.93
	30	2	0.1	75.26	74.88	73.72
	30	4	0.1	74.82	74.72	71.06
	30	8	0.01	72.02	71.60	69.62

Table A6: Performance of wide three-layer GCN model (i.e., hidden dimension 256) trained with GIST on small-scale datasets. These tests do not split the input layer, but they do split the output layer. Tests that do not split the output layer performed similarly.

Dataset	ζ	# Sub-GCN	LR	Best Test Acc.	Best Val. Acc.	Last Test Acc.
CORA	1	2	0.05	79.32	78.84	76.26
	1	4	0.01	79.80	78.68	76.28
	1	8	0.05	79.40	77.88	75.64
	5	2	0.1	79.80	78.44	75.54
	5	4	0.05	79.56	77.48	77.28
	5	8	0.05	78.20	76.60	77.56
	10	2	0.1	79.48	78.44	77.70
	10	4	0.01	78.80	77.80	76.88
	10	8	0.005	77.66	76.84	77.44
	20	2	0.1	79.24	78.08	77.16
	20	4	0.1	78.30	77.32	76.92
	20	8	0.01	77.22	75.24	76.94
	30	2	0.05	78.26	77.40	76.32
	30	4	0.01	78.18	76.76	78.08
	30	8	0.01	76.44	74.68	76.44
CITESEER	1	2	0.05	75.38	75.26	69.32
	1	4	0.1	75.89	75.12	68.35
	1	8	0.05	75.93	75.14	71.85
	5	2	0.1	75.63	74.42	69.60
	5	4	0.1	75.39	75.06	72.20
	5	8	0.05	75.06	74.42	72.91
	10	2	0.1	74.90	74.26	72.35
	10	4	0.1	74.93	74.62	73.22
	10	8	0.005	74.06	73.90	72.78
	20	2	0.1	74.18	73.54	71.47
	20	4	0.1	73.55	72.96	72.86
	20	8	0.005	73.51	73.20	72.66
	30	2	0.05	73.35	73.36	71.84
	30	4	0.1	72.95	72.82	72.71
	30	8	0.005	72.33	72.00	72.33
PUBMED	1	2	0.05	76.31	76.56	70.95
	1	4	0.05	76.51	76.55	71.55
	1	8	0.05	76.03	75.92	72.05
	5	2	0.1	76.38	75.69	71.15
	5	4	0.1	75.89	75.88	72.98
	5	8	0.05	74.91	74.97	72.98
	10	2	0.1	75.45	75.12	72.89
	10	4	0.1	75.45	75.45	73.93
	10	8	0.005	74.58	74.44	73.56
	20	2	0.1	74.89	74.61	72.72
	20	4	0.05	73.91	73.45	73.24
	20	8	0.05	73.99	73.81	73.21
	30	2	0.05	74.22	74.33	72.93
	30	4	0.1	73.87	74.05	73.33
	30	8	0.005	72.84	72.91	72.75

Table A7: Performance of wide three-layer GCN model (i.e., hidden dimension 256) on small-scale datasets trained with `GIST`. These tests split the input layer and the output layer.

Dataset	ζ	# Sub-GCN	LR	Best Test Acc.	Best Val. Acc.	Last Test Acc.
CORA	1	2	0.05	80.64	79.40	76.64
	1	4	0.05	79.60	78.36	76.28
	1	8	0.05	73.70	72.72	60.48
	10	2	0.1	79.18	79.24	76.86
	10	4	0.01	76.38	75.04	75.60
	10	8	0.01	62.08	60.36	61.56
	20	2	0.05	78.90	78.32	77.78
	20	4	0.01	76.56	74.80	76.56
	20	8	0.01	57.00	54.40	57.00
CITESEER	1	2	0.1	71.02	75.64	70.72
	1	4	0.05	75.82	74.68	72.43
	1	8	0.01	72.47	71.28	66.80
	10	2	0.1	74.96	75.26	72.60
	10	4	0.01	73.10	72.94	72.05
	10	8	0.01	63.77	63.00	63.16
	20	2	0.1	74.14	73.88	72.80
	20	4	0.01	72.67	71.36	71.63
	20	8	0.01	58.67	57.44	58.67
PUBMED	1	2	0.05	76.57	76.51	72.54
	1	4	0.1	75.52	75.67	71.55
	1	8	0.05	73.03	72.96	65.15
	10	2	0.1	75.52	75.73	73.53
	10	4	0.01	74.05	73.83	73.14
	10	8	0.01	65.51	65.55	64.81
	20	2	0.1	74.83	74.88	73.61
	20	4	0.01	73.45	73.08	72.51
	20	8	0.01	62.12	61.12	59.41