Fully Distributed Online Training of Graph Neural Networks in Networked Systems

Rostyslav Olshevskyi[⋆], Zhongyuan Zhao[⋆], Kevin Chan[‡], Gunjan Verma[‡], Ananthram Swami[‡], Santiago Segarra[⋆]
[⋆]*Rice University, USA* [‡]*DEVCOM Army Research Laboratory, USA*

***Abstract*—Graph neural networks (GNNs) are powerful tools for developing scalable, decentralized artificial intelligence in large-scale networked systems, such as wireless networks, power grids, and transportation networks. Currently, GNNs in networked systems mostly follow a paradigm of 'centralized training, distributed execution', which limits their adaptability and slows down their development cycles. In this work, we fill this gap for the first time by developing a communication-efficient, fully distributed online training approach for GNNs applicable to large networked systems. For a mini-batch with B samples, our approach of training an L-layer GNN only adds L rounds of message passing to the LB rounds required by GNN inference, with doubled message sizes. Through numerical experiments in graph-based node regression, power allocation, and link scheduling in wireless networks, we demonstrate the effectiveness of our approach in training GNNs under supervised, unsupervised, and reinforcement learning paradigms.**

***Index Terms*—Distributed optimization, graph neural networks, wireless networks, distributed gradient descent.**

I. Introduction

Graph neural networks (GNNs) hold the promise of empowering networked artificial intelligence in communication networks, smart grids, and transportation networks, due to several unique features [1], [2]: 1) *permutation equivariance* as an important inductive bias for tasks in networks, 2) *local message passing (MP)* that naturally promotes distributed executions, and 3) *shared trained model among all nodes*, which allows GNNs to generalize and scale up to large, dynamic networks much easier than, e.g., multi-agent reinforcement learning (MARL) [3]. GNNs have been applied to enhance resource allocation and decision-making in wireless networks, such as power allocation, link scheduling, packet routing, network simulation and management, and computation offloading [4]–[10], by leveraging their ability to exploit the topological information of the connectivity and interference relationships among wireless devices.

Current applications of GNNs in networked systems follow a paradigm of '*centralized training, distributed execution*'. In particular, the centralized training of GNNs can only be done offline in simulated environments, requiring extensive efforts in data collection and environment modeling as well as computing resources. Moreover, the distribution and deployment of trained models may cause downtime and disruptions to the networked system. After deployment, the trained models would also likely experience distribution shifts due to mismatched training settings, changing real-world environments, and application scenarios. Therefore, fully distributed online training of GNNs could simplify the development of intelligent networked systems and improve their adaptivity.

Existing approaches in distributed machine learning [11] are inadequate for fully distributed online training of GNNs. For example, with MARL [3], each agent has a different trained model rather than a common trained model shared among all nodes as in GNNs. Although federated learning seeks to train a common model for many clients with the help of a central server [12], neither the training nor inference requires any interactions between these clients, which is in contrast with GNNs where the inference requires synchronized message exchanging between each node and its neighbors. Distributed optimization (DO) [13]–[16] leverages many connected workers to accelerate the training of a model by splitting the training dataset and exchanging gradients among the workers. However, each worker in DO can perform model inference and backpropagation individually, which is different from GNNs where both inference and backpropagation require synchronized collaboration among all nodes in the network. Existing works on distributed training of GNNs can be categorized as DO, where large graphs are divided into smaller subgraphs, which are distributed to different servers for memory and computing efficiencies [17], [18]. By contrast, in our *fully* distributed training, every node in the graph is its own computing server. Thus, communication across servers is required for both inference and training.

To fill the gap of training GNNs online in a fully distributed manner, we take the supervised learning in graph convolutional neural networks (GCNNs) [19], [20] as an example, transform it into a variation of the DO framework, and develop communication-efficient implementations based on local MP. These principles of distributed training can serve as the basis for other types of GNNs [1], such as edge-featured GNNs and graph attention networks (GATs), as well as unsupervised training and reinforcement learning for GNNs in sophisticated algorithmic frameworks [4], [8].

Contributions: Our contributions are as follows:

- We show that GNN training can be reformulated as a

Research was sponsored by the DEVCOM ARL Army Research Office and was accomplished under Cooperative Agreement Number W911NF-24-2-0008. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DEVCOM ARL Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Emails: [⋆]{rostyslav.olshevskyi, zhongyuan.zhao, segarra}@rice.edu, [‡]{kevin.s.chan.civ, gunjan.verma.civ, ananthram.swami.civ}@army.mil

Source code and data: https://github.com/RostyslavUA/fdTrainGNN

DO problem by decomposing the global objective, loss function, and gradient of GNNs into linear combinations of the corresponding local ones, and deriving a local form of backpropagation for GCNNs.
- We develop a communication-efficient approach for distributed training of GNNs, by incorporating distributed gradient descent schemes, rearranging gradient aggregation, and message piggybacking in mini-batch settings.
- Through numerical experiments, we demonstrate that our distributed training scheme not only achieves a convergence behavior very close to that of the classical, centrally-trained approach in supervised learning, but is also effective in more sophisticated ML pipelines such as graph-based algorithmic unfolding [4] and graph-based actor-critic reinforcement learning frameworks [8].

Notational convention: $(\cdot)^\top$, $\odot$, and $|\cdot|$ represent the transpose operator, Hadamard (element-wise) product operator, and the cardinality of a set (or dimensionality of a vector), respectively. $\mathbb{E}(\cdot)$ stands for expectation. Calligraphic symbols, e.g., $\mathcal{V}$, denote a set. Upright bold lower-case symbols, e.g., $\mathbf{z}$, denote a column vector and $\mathbf{z}_i$ denotes the i-th element of vector $\mathbf{z}$. Upright bold upper-case symbols, e.g., $\mathbf{Z}$ denote a matrix, of which the element at row i and column j is denoted by $\mathbf{Z}_{ij}$, the row i by $\mathbf{Z}_{i*}$, and the column j by $\mathbf{Z}_{*j}$.

II. Problem Formulation

Consider a connected and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times g_0}$ collects node features (e.g., the type or queueing state of a transmitter), and vector $\mathbf{y} \in \mathbb{R}^{|\mathcal{V}|}$ collects node-wise labels (e.g., optimal transmit power). An L-layer GCNN is a parameterized function $\hat{\mathbf{y}} = \Psi_{\mathcal{G}}(\mathbf{X}; \boldsymbol{\theta})$ defined on $\mathcal{G}$, where vector $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathcal{V}|}$ is the node-wise prediction, and $\boldsymbol{\theta}$ collects all the trainable parameters. Centralized training of the GCNN with supervised learning can be formulated as minimizing the expected loss over the distribution of node-featured graphs and the corresponding label vectors, $(\mathcal{G}, \mathbf{X}, \mathbf{y}) \in \Omega$, as

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathbb{R}^{|\boldsymbol{\theta}|}}{\operatorname{argmin}} J(\boldsymbol{\theta}) \tag{1a}$$

$$\text{s.t. } J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathcal{G},\mathbf{X},\mathbf{y}) \in \Omega} \left[\ell(\mathbf{y}, \mathcal{G}, \mathbf{X}; \boldsymbol{\theta}) \right] , \tag{1b}$$

$$\ell(\boldsymbol{\theta}) = \ell(\mathbf{y}, \mathcal{G}, \mathbf{X}; \boldsymbol{\theta}) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 , \tag{1c}$$

$$\hat{\mathbf{y}} = \Psi_{\mathcal{G}}(\mathbf{X}; \boldsymbol{\theta}), \tag{1d}$$

where (1c) defines the mean-squared-error (MSE) loss function. Problem (1) can be solved with stochastic gradient descent (SGD) with a learning rate of α,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \widehat{\nabla J(\boldsymbol{\theta})}, \ \widehat{\nabla J(\boldsymbol{\theta})} = \nabla \ell(\boldsymbol{\theta}) = \left[\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right]^\top \in \mathbb{R}^{|\boldsymbol{\theta}|} . \tag{2}$$

The local implementation of a GCNN comprises synchronized parallel executions of the same parameterized local function on every node, where each local function performs L iterations of a local neighborhood aggregation followed by a dense layer. Thus, the L-layer GCNN can be denoted as $\hat{\mathbf{y}} = \Psi_{\mathcal{G}}(\mathbf{X}; \{\boldsymbol{\theta}^i\}_{i \in \mathcal{V}})$, where $\boldsymbol{\theta}^i = \boldsymbol{\theta}$ is a local copy of the global parameters on node $i \in \mathcal{V}$. The backward pass of a GCNN first computes the partial derivatives $\partial \ell(\boldsymbol{\theta}) / \partial \boldsymbol{\theta}^i$ for all $i \in \mathcal{V}$, and then the total derivative as

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i \in \mathcal{V}} \frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^i} . \tag{3}$$

For the centralized training of a GCNN, the operation in (3) is straightforward since the global loss function $\ell(\boldsymbol{\theta})$ and all local gradients reside on the same server. However, for GCNNs in fully distributed systems, where each node $i \in \mathcal{V}$ is an individual device in the network, online training becomes challenging as it requires a central server to host the global loss function in (1c), perform the summation operation in (3), SGD in (2), and redistribution of $\boldsymbol{\theta}$ to $\boldsymbol{\theta}^i$ for all $i \in \mathcal{V}$.

To transform the centralized training in (1) into a distributed problem, we define the local loss $\ell_i(\boldsymbol{\theta})$ and local objective $J_i(\boldsymbol{\theta})$ for all $i \in \mathcal{V}$ as follows,

$$\ell(\boldsymbol{\theta}) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \ell_i(\boldsymbol{\theta}) \text{ , where } \ell_i(\boldsymbol{\theta}) = (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 , \tag{4}$$

$$J(\boldsymbol{\theta}) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} J_i(\boldsymbol{\theta}) \text{ , where } J_i(\boldsymbol{\theta}) = \mathbb{E}_{\Omega} \left[\ell_i(\boldsymbol{\theta}) \right] . \tag{5}$$

With (4), (5), and a small $\epsilon > 0$, we reformulate (1) for a sampling distribution $\Omega_{\mathcal{V}}$ conditioned on a vertex set $\mathcal{V}$ as

$$\{\boldsymbol{\theta}^i\}^*_{i \in \mathcal{V}} = \underset{\boldsymbol{\theta}^i \in \mathbb{R}^{|\boldsymbol{\theta}|}, i \in \mathcal{V}}{\operatorname{argmin}} \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} J_i(\boldsymbol{\theta}) \tag{6a}$$

$$\text{s.t. } J_i(\boldsymbol{\theta}) = \mathbb{E}_{\Omega_{\mathcal{V}}} \left[\ell_i(\boldsymbol{\theta}) \right] , \tag{6b}$$

$$\ell_i(\boldsymbol{\theta}) = (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 , \forall\, i \in \mathcal{V} , \tag{6c}$$

$$\hat{\mathbf{y}} = \Psi_{\mathcal{G}}(\mathbf{X}; \{\boldsymbol{\theta}^i\}_{i \in \mathcal{V}}), \ (\mathcal{G}, \mathbf{X}, \mathbf{y}) \in \Omega_{\mathcal{V}} , \tag{6d}$$

$$\boldsymbol{\theta} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \boldsymbol{\theta}^i, \ \|\boldsymbol{\theta}^i - \boldsymbol{\theta}\|_2^2 < \epsilon, \ \forall\, i \in \mathcal{V}. \tag{6e}$$

Based on (3) and (4), the total derivative becomes

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} \frac{\partial \ell_j(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^i} . \tag{7}$$

Consequently, the SGD in (2) can be implemented in a distributed manner as

$$\boldsymbol{\theta}^i \leftarrow \boldsymbol{\theta}^i - \alpha \widehat{\nabla J(\boldsymbol{\theta})} , \forall\, i \in \mathcal{V} , \tag{8a}$$

$$\widehat{\nabla J(\boldsymbol{\theta})} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \widehat{\nabla J_i(\boldsymbol{\theta})} , \tag{8b}$$

$$\widehat{\nabla J_i(\boldsymbol{\theta})} = \left[\sum_{j \in \mathcal{V}} \frac{\partial \ell_j(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^i} \right]^\top , \forall\, i \in \mathcal{V} , \tag{8c}$$

given that $\boldsymbol{\theta}^i = \boldsymbol{\theta}$, for all $i \in \mathcal{V}$ upon initialization.

Unlike the centralized training in (1), the distributed optimization in (6) no longer requires a server to host the global loss function $\ell(\boldsymbol{\theta})$ or global objective function $J(\boldsymbol{\theta})$. Moreover, as long as (8b) and (8c) can be computed in a fully

distributed manner, the centralized SGD in (2) and (3) can be attained via (8) without the need of a central server.

Key departure from classical DO. It is essential to notice that in a local implementation of an L-layer GCNN where every node $i \in \mathcal{V}$ has a copy of the parameters $\boldsymbol{\theta}^i$, the estimate $\hat{\mathbf{y}}_i$ at node i is *not only a function of* $(\mathbf{X}_{i*}, \boldsymbol{\theta}^i)$ but also those of its L-hop neighbors,

$$\hat{\mathbf{y}}_i = f_{\text{local}}\left(\left\{\mathbf{X}_{j*}; \boldsymbol{\theta}^j\right\}_{j\in\mathcal{N}_{\mathcal{G}}^L(i)\cup\{i\}}\right) ,$$

where $\mathcal{N}_{\mathcal{G}}^L(i)$ denotes the set of L-hop neighbors of node i on graph $\mathcal{G}$. This follows immediately from the fact that $\hat{\mathbf{y}}_i$ depends on messages that node i receives from its neighbors j, and these messages are functions of their parameters $\{\boldsymbol{\theta}^j\}$. This dependence cascades over the L layers, as later illustrated in (10) for the specific case of GCNNs. Consequently, unlike in classical DO [13]–[16], node i *cannot immediately compute the local gradient* since $\frac{\partial \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^i} \neq \frac{1}{|\mathcal{V}|}\frac{\partial \ell_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^i}$. In this context, the local gradient computation requires further consideration. In the next section, we introduce our fully distributed solution to implement (8). Our objective is twofold: first, we want to minimize the global objective function in (6a) in a fully distributed manner. Second, we aim to minimize the communication costs of our fully distributed training.

III. Fully Distributed Training of GNNs

Our solution comprises three components: 1) fully-distributed backpropagation for local gradient estimation (8c) in Section III-A; 2) joint implementation of (8b) and (8a) based on distributed gradient descent approaches in Section III-B; and 3) systematic schemes to reduce the communication rounds for mini-batch training in Section III-C.

A. Fully Distributed Backpropagation

The lth layer of the GCNN $\hat{\mathbf{y}} = \Psi_{\mathcal{G}}(\mathbf{X}; \boldsymbol{\theta})$ introduced in Section II, where $l \in \{1, \ldots, L\}$, can be expressed as

$$\mathbf{X}^l = \sigma_l\left(\mathbf{X}^{l-1}\boldsymbol{\Theta}_0^l + \mathbf{S}\mathbf{X}^{l-1}\boldsymbol{\Theta}_1^l\right) ,\ l \in \{1, \ldots, L\}, \quad (9)$$

where $\mathbf{X}^l \in \mathbb{R}^{|\mathcal{V}|\times g_l}$ collects the output node features of layer l, matrices $\boldsymbol{\Theta}_0^l, \boldsymbol{\Theta}_1^l \in \mathbb{R}^{g_{l-1}\times g_l}$ are the trainable parameters of the lth layer, σ_l is an element-wise activation function, and $\mathbf{S} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the graph shift operator. $\mathbf{S}$ can be selected as the adjacency matrix $\mathbf{A}$, the graph Laplacian $\mathbf{L}$, or their normalized versions [19]. For the GCNN, the input node feature $\mathbf{X}^0 = \mathbf{X}$, and prediction $\hat{\mathbf{y}} = \mathbf{X}^L$ where $g_L = 1$.

The local form of the GCNN layer in (9) on node $i \in \mathcal{V}$ is

$$\mathbf{X}_{i*}^l = \sigma_l(\mathbf{H}_{i*}^l) ,\ \mathbf{H}_{i*}^l = \mathbf{X}_{i*}^{l-1}\boldsymbol{\Theta}_0^{li} + \sum_{j\in\mathcal{N}_{\mathcal{G}}^+(i)} \mathbf{S}_{ij}\mathbf{X}_{j*}^{l-1}\boldsymbol{\Theta}_1^{li} , \quad (10)$$

where $\mathbf{X}_{i*}^l \in \mathbb{R}^{1\times g_l}$ captures the output features of layer l on node i, $\boldsymbol{\Theta}_0^{li}$ and $\boldsymbol{\Theta}_1^{li}$ are local copies at node i of $\boldsymbol{\Theta}_0^l$ and $\boldsymbol{\Theta}_1^l$, and $\mathcal{N}_{\mathcal{G}}^+(i) = \mathcal{N}_{\mathcal{G}}(i) \cup \{i\}$ with $\mathcal{N}_{\mathcal{G}}(i) = \mathcal{N}_{\mathcal{G}}^1(i)$. The distributed execution of a GCNN layer can be implemented in two steps: first, node $i \in \mathcal{V}$ exchanges its input node feature $\mathbf{X}_{i*}^{l-1}$ with its neighbors $j \in \mathcal{N}_{\mathcal{G}}(i)$; second, each node locally computes (10). The L-layer GCNN requires L rounds of MP.

To find the messages passed in backpropagation, we define $\mathbf{Z}^l = \partial\ell(\boldsymbol{\theta})/\partial\mathbf{X}^l$ and $\mathbf{Q}^l = \partial\ell(\boldsymbol{\theta})/\partial\mathbf{H}^l$ for $l \in \{1, \ldots, L\}$, where $\mathbf{Z}^l, \mathbf{Q}^l \in \mathbb{R}^{g_l\times|\mathcal{V}|}$. For node $i \in \mathcal{V}$, we have

$$\mathbf{Z}_{*i}^l = \frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{X}_{i*}^l} \in \mathbb{R}^{g_l\times 1},\ \mathbf{Z}_{*i}^L = \frac{\partial\ell(\boldsymbol{\theta})}{\partial\hat{\mathbf{y}}_i} = 2(\hat{\mathbf{y}}_i - \mathbf{y}_i) , \quad (11)$$

$$\mathbf{Q}_{*i}^l = \frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{X}_{i*}^l}\frac{\partial\mathbf{X}_{i*}^l}{\partial\mathbf{H}_{i*}^l} = \mathbf{Z}_{*i}^l \odot \sigma_l'(\mathbf{H}_{i*}^l) \in \mathbb{R}^{g_l\times 1}, \quad (12)$$

where $\sigma_l'(\cdot)$ is the element-wise derivative function of the activation $\sigma_l(\cdot)$. Based on (10), the local partial derivatives for the trainable parameters at node $i \in \mathcal{V}$ are:

$$\frac{\partial\ell(\boldsymbol{\theta})}{\partial\boldsymbol{\Theta}_0^{li}} = \frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{H}_{i*}^l}\frac{\partial\mathbf{H}_{i*}^l}{\partial\boldsymbol{\Theta}_0^{li}} = \mathbf{Q}_{*i}^l\mathbf{X}_{i*}^{l-1} \in \mathbb{R}^{g_l\times g_{l-1}} , \quad (13a)$$

$$\frac{\partial\ell(\boldsymbol{\theta})}{\partial\boldsymbol{\Theta}_1^{li}} = \frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{H}_{i*}^l}\frac{\partial\mathbf{H}_{i*}^l}{\partial\boldsymbol{\Theta}_1^{li}} = \mathbf{Q}_{*i}^l\sum_{j\in\mathcal{N}_{\mathcal{G}}^+(i)}\mathbf{S}_{ij}\mathbf{X}_{j*}^{l-1} . \quad (13b)$$

According to the chain rule, we can find $\mathbf{Z}_{*i}^{l-1} \in \mathbb{R}^{g_{l-1}\times 1}$ as

$$\begin{aligned}\mathbf{Z}_{*i}^{l-1} &= \frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{X}_{i*}^{l-1}} = \sum_{j\in\mathcal{V}}\frac{\partial\ell(\boldsymbol{\theta})}{\partial\mathbf{H}_{j*}^l}\frac{\partial\mathbf{H}_{j*}^l}{\partial\mathbf{X}_{i*}^{l-1}} \\ &= \left(\boldsymbol{\Theta}_0^{li} + \mathbf{S}_{ii}\boldsymbol{\Theta}_1^{li}\right)\mathbf{Q}_{*i}^l + \sum_{j\in\mathcal{N}_{\mathcal{G}}(i)}\mathbf{S}_{ji}\boldsymbol{\Theta}_1^{lj}\mathbf{Q}_{*j}^l\end{aligned} \quad (14)$$

since, based on (10), we have the following

$$\left[\frac{\partial\mathbf{H}_{j*}^l}{\partial\mathbf{X}_{i*}^{l-1}}\right]^\top = \begin{cases}\boldsymbol{\Theta}_0^{li} + \mathbf{S}_{ii}\boldsymbol{\Theta}_1^{li} & \text{if } j = i ,\\ \mathbf{S}_{ji}\boldsymbol{\Theta}_1^{lj} & \text{if } j \in \mathcal{N}_{\mathcal{G}}(i) ,\\ \mathbf{0}_{g_{l-1}\times g_l} & \text{if } j \notin \mathcal{N}_{\mathcal{G}}^+(i).\end{cases}$$

Equations (11)-(14) show the local form of backpropagation for all layers $l \in \{1, \ldots, L\}$. Notice that the sum operation in (13b) is already done in the forward pass in (10). Only the second term in (14) requires an additional round of MP, i.e., each node i broadcasts $\boldsymbol{\Theta}_1^{li}\mathbf{Q}_{*i}^l$ to all its neighbors as $\mathbf{S}_{ji}$ can be found from the forward pass. Based on (11)-(14), $L-1$ rounds of MP are required to estimate the local gradient $\widehat{\nabla J_i(\boldsymbol{\theta})}$ in (8c).

B. Distributed Stochastic Gradient Descent

A naive approach for implementing (8b) is to perform $K \geq 1$ rounds of distributed consensus on the local gradient estimates $\{\widehat{\nabla J_i(\boldsymbol{\theta})}\}_{i\in\mathcal{V}}$. The kth round of distributed consensus on a set of node-specific vectors $\{\mathbf{x}^j(k)\}_{j\in\mathcal{V}}$ can be expressed as

$$\mathbf{x}^i(k+1) = \sum_{j\in\mathcal{N}_{\mathcal{G}}^+(i)}\mathbf{W}_{ij}\mathbf{x}^j(k) ,\ k \in \{1, \ldots, K\}, \quad (15)$$

where matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ collects the consensus weights. Denoting the degree of node i by $d(i)$, a good candidate for $\mathbf{W}$ is the Metropolis-Hasting weights [21]

$$\mathbf{W}_{ij} = \begin{cases}\frac{1}{1+\max\{d(i),d(j)\}} & \{i,j\} \in \mathcal{E} ,\\ 1 - \sum_{v\in\mathcal{N}_{\mathcal{G}}(i)}\mathbf{W}_{iv} & i = j ,\\ 0 & \text{otherwise.}\end{cases} \quad (16)$$

Algorithm 1 Distributed Training of GCNNs

Input: $\mathcal{V}, \Omega, \alpha, B$
Output: $\{\boldsymbol{\theta}^i\}^*_{i\in\mathcal{V}}, \boldsymbol{\theta}^*$

1: Initialize $\boldsymbol{\theta}$ randomly, $t=0$, $\{\boldsymbol{\theta}^i(t)\}_{i\in\mathcal{V}}=\{\boldsymbol{\theta}\}_{i\in\mathcal{V}}$, $\alpha_t=\alpha$
2: **while** not converged **do**
3: Draw graph $\mathcal{G}(t)=(\mathcal{V},\mathcal{E}(t))$ from Ω conditioned on $\mathcal{V}$
4: Compute $\mathbf{W}$ using (16)
5: **for** $b=1,\ldots,B$ **do**
6: Draw node features and labels $(\mathbf{X}(b),\mathbf{y}(b))\sim\Omega_{\mathcal{G}(t)}$
7: **for all** $i\in\mathcal{V}$ **do**
8: Compute $\hat{\mathbf{y}}_i(b)$ using (10) for all $l\in\{1,\ldots,L\}$.
9: Compute $\widehat{\nabla J_i(\boldsymbol{\theta})}(b)$ using (11)–(14)
10: **end for**
11: **end for**
12: Update $\{\boldsymbol{\theta}^i(t+1)\}_{i\in\mathcal{V}}$ using (17)
13: $t=t+1$, update α_t, e.g., via exponential decay
14: **end while**
15: Return $\{\boldsymbol{\theta}^i\}^*_{i\in\mathcal{V}}=\{\boldsymbol{\theta}^i(t)\}_{i\in\mathcal{V}}$, $\boldsymbol{\theta}^*=\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}}\boldsymbol{\theta}^i(t)$

However, this naive approach has a high communication cost since a large K is required for the convergence of the consensus operation in each gradient step.

Notice that the formulation in (6a) can be considered as a form of DO but where the local gradient estimate $\widehat{\nabla J_i(\boldsymbol{\theta})}$ is obtained as described in Section III-A. Thus, we can employ efficient approaches such as D-SGD [14], D-Adam [15], and D-AMSGrad [16]. In the context of the popular mini-batch gradient descent, where the network topology is assumed to be static during a mini-batch, each local update in DO comprises one round of distributed consensus on $\boldsymbol{\theta}^i$ and an application of local gradient,

$$\boldsymbol{\theta}^i(t+1)=\sum_{j\in\mathcal{N}^+_{\mathcal{G}}(i)}\mathbf{W}_{ij}\boldsymbol{\theta}^j(t)-\alpha_t f\left(\left\{\widehat{\nabla J_i(\boldsymbol{\theta})}(b)\right\}_{b=1}^{B}\right)\ , \quad (17)$$

where the function $f(\cdot)$ aggregates local gradients over a mini-batch of B samples. To estimate the gradients of B samples, $\{\widehat{\nabla J(\boldsymbol{\theta})}(b)\}_{b=1}^B$, we require B forward passes of the GCNN for the inference of $\{\hat{\mathbf{y}}(b)\}_{b=1}^B$ and B passes of backpropagation. For D-SGD, function $f(\cdot)$ is simply a summation

$$f\left(\left\{\widehat{\nabla J_i(\boldsymbol{\theta})}(b)\right\}_{b=1}^{B}\right)=\sum_{b=1}^{B}\widehat{\nabla J_i(\boldsymbol{\theta})}(b)\ .$$

However, for D-Adam and D-AMSGrad, $f(\cdot)$ is based on the momentum of the local gradients from graph data samples.

The entire procedure of DO-based distributed training on a network of a fixed set of vertices $\mathcal{V}$ with dynamic topology, i.e., edge set $\mathcal{E}(t)$, is illustrated in Algorithm 1, where Ω is the sampling distribution for $(\mathcal{G},\mathbf{X},\mathbf{y})$, and $\Omega_{\mathcal{V}}$ (or $\Omega_{\mathcal{G}}$) is the conditional distribution for a given $\mathcal{V}$ (or $\mathcal{G}$).

C. Communication-efficient Mini-Batch Training

If we approach (8b) naively by running distributed consensus on local gradients for each sample $(\mathbf{X}(b),\mathbf{y}(b))$, the whole mini-batch requires $B(2L-1+K)$ rounds of MP. Moreover, messages exchanged in distributed consensus are of large size, e.g., $|\boldsymbol{\theta}|$. We can reduce the communication cost by i) re-using information from the forward pass, ii) rearranging the mini-batch update, and iii) piggybacking messages for backward and forward passes, as we discuss next.

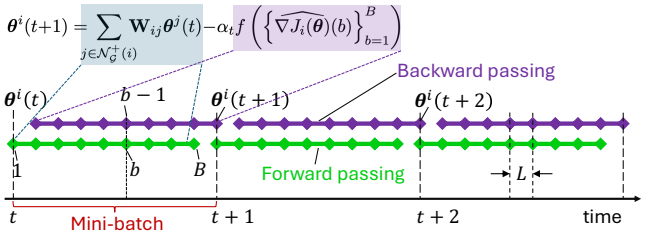

Fig. 1: Timeline of fully-distributed training of GCNN in mini-batches. By piggybacking messages in the backward pass of sample $b-1$ into the messages of the forward pass of sample b, a mini-batch requires only $L(B+1)$ rounds of MP. Notice that most communication and computation for the consensus step and local gradient aggregation (line 12 in Algo 1) can be piggybacked to the messages of B forward passes and carried out in parallel with the processing of data samples (lines $5-11$ in Algo 1).

Information reuse: In the GCNN forward pass for sample b, node $i\in\mathcal{V}$ can save the intermediate variables $\mathbf{H}^l_{i*}(b)$, $\mathbf{X}^l_{i*}(b)$ and $\sum_{j\in\mathcal{N}^+_{\mathcal{G}}(i)}\mathbf{S}_{ij}\mathbf{X}^{l-1}_{j*}(b)$ in (10) for all $l\in\{1,\ldots,L\}$, which can be reused in (12) and (13) for the following backward pass, without retransmission and recomputing.

Mini-batch rearrangement: Instead of running expensive distributed consensus for every sample, we can first aggregate B local gradients at each node and then run K rounds of distributed consensus once per mini-batch. The latter is mathematically equivalent to the former under basic SGD,

$$\sum_{b=1}^{B}\left[\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}}\widehat{\nabla J_i(\boldsymbol{\theta})}(b)\right]=\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}}\left[\sum_{b=1}^{B}\widehat{\nabla J_i(\boldsymbol{\theta})}(b)\right]\ ,$$

but cuts the total rounds per mini-batch to $B(2L-1)+K$. Furthermore, the DO-based gradient descent in (17) requires only 1 round of consensus per mini-batch, where each node i only needs to broadcast $\boldsymbol{\theta}^i(t)\in\mathbb{R}^{|\boldsymbol{\theta}|}$ to its neighbors $j\in\mathcal{N}_{\mathcal{G}}(i)$ once. Since local aggregation $f(\cdot)$ needs no MP, the required communication rounds is further reduced to $B(2L-1)+1$ per mini-batch. In addition, for momentum-based DO, consensus on momentum parameters is also needed to ensure convergence, as in D-AMSGrad [16].

Piggybacking: Since it is more efficient to transmit a larger message than multiple smaller messages in wireless networks due to the signaling overhead of each transmission, we can piggyback the message of the backward pass for sample $b-1$ to the forward pass of the next sample b, as shown in Fig. 1. For example, in the MP of layer $l<L$ in the forward pass for sample b, each node $i\in\mathcal{V}$ sends a message containing $\mathbf{X}^{l-1}_{i*}(b)\in\mathbb{R}^{1\times g_l}$ and $[\boldsymbol{\Theta}^{\bar{l}i}_1\mathbf{Q}^{\bar{l}i}_{*i}(b-1)]^\top\in\mathbb{R}^{1\times g_{\bar{l}}}$, where $\bar{l}=L-l+1$, to all its neighbors $j\in\mathcal{N}_{\mathcal{G}}(i)$, which can be achieved by a single broadcast transmission with an omnidirectional antenna. When $l=L$ or $b=1$, only $\mathbf{X}^{l-1}_{i*}(b)$ is exchanged. Once per mini-batch, each node i also needs to send $d(i)\in$

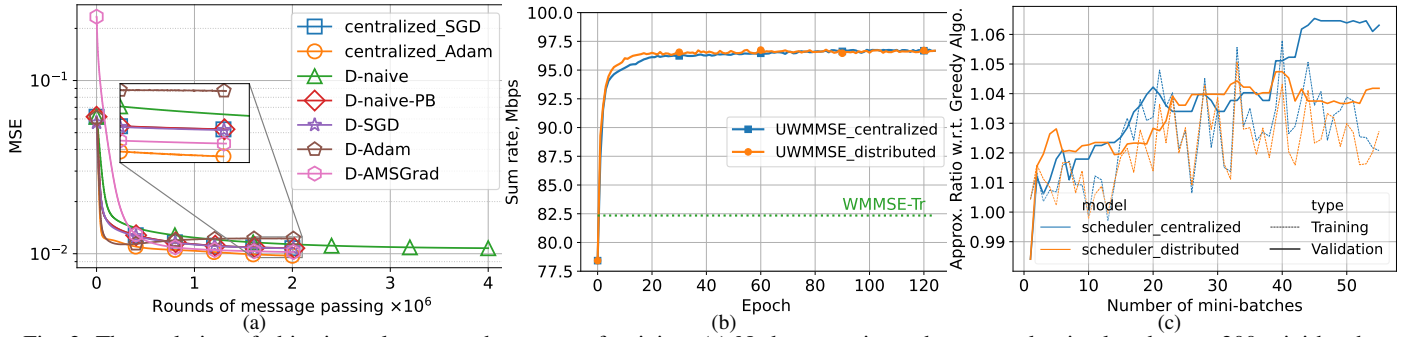

Fig. 2: The evolution of objective values over the course of training: (a) Node regression, where a marker is placed every 200 mini-batches. (b) Power allocation for a network of 25 transmitter-receiver pairs. (c) Distributed link scheduling in conflict graphs of 100 nodes (links).

TABLE I: Communication cost of a mini-batch in GCNN training

Accumulated Measures	**Total Msg. Rounds**	**Message size**
Forward pass (FWD) only	LB	g_l
FWD + D-naive (grad. consensus per sample) + reuse	$B(2L-1)$ $+BK$	g_l, $\|\boldsymbol{\theta}\|$
FWD + Grad. consensus per batch + Info. reuse	$2BL-B$ $+K$	g_l, $\|\boldsymbol{\theta}\|$
FWD + Grad. consensus per batch + reuse + piggyback	$LB+L-1$ $+K$	g_l+g_{L-l+1}, $\|\boldsymbol{\theta}\|$
FWD + Dist. Opt. in (17) + reuse + piggyback	$LB+L-1$	$g_l+g_{L-l+1}+$ $\|\boldsymbol{\theta}\|/LB$

$\mathbb{R}$ and $\boldsymbol{\theta}^i(t)$ to all its neighbors $j \in \mathcal{N}_{\mathcal{G}}(i)$, which can be broken up and piggybacked in the messages of $b=1$ and $l=L, b>1$. As a result, the total rounds of MP for a mini-batch are reduced to $LB+L-1$, where $L-1$ rounds are for the backward pass of the last sample $b=B$. The detailed communication costs for these approaches are listed in Table I.

IV. Numerical Simulations

To evaluate the effectiveness of our proposed distributed training scheme, we compare it with centralized training (and naive distributed training) under supervised learning for node regression on a synthetic dataset (Section IV-A), unsupervised learning in UWMMSE wireless power allocation [4] (Section IV-B), and graph-based actor-critic reinforcement learning for distributed link scheduling [8] (Section IV-C).

A. Supervised Learning on Synthetic Graph Data

In this experiment, we train a 2-layer GCNN to predict a set of graph data $\{(\mathbf{X}(n), \mathbf{y}(n)) \in \Omega_{\mathcal{G}}\}_{n=1}^{N}$ under a given random graph $\mathcal{G}$ drawn from a Barabási-Albert model ($m=2$) with $|\mathcal{V}|=100$ nodes. The input $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times 10}$ comprises a combination of continuous, binary, and discrete (one-hot) features drawn from uniform and normal distributions. The labels are generated by a non-linear process as $\mathbf{y}=\Phi_{\mathcal{G}}(\mathbf{X};\boldsymbol{\omega})+\mathbf{n}$, where $\Phi_{\mathcal{G}}(\cdot)$ is a 2-layer GCNN parameterized by a set of random weights $\boldsymbol{\omega}$, and $\mathbf{n}$ is Gaussian noise with a variance of 0.01. With a set of training samples $N=1000$, batch size of $B=100$, a learning rate $\alpha=10^{-3}$, and a total of 1000 epochs, we train a GCNN $\Psi_{\mathcal{G}}(\cdot;\boldsymbol{\theta})$ with supervised learning using different approaches.

The MSE loss as a function of MP rounds for 1000 epochs under 7 different training approaches are presented in Fig. 2(a), where the differences in message sizes are ignored. For centralized training (with SGD and Adam optimizers), only the cost of forward passing is considered such that processing each data sample requires $L=2$ rounds of communication. The number of consensus rounds was set to $K=1$ for naive distributed approaches (D-naive and D-naive-PB, where PB stands for piggybacking). After 1000 epochs, the MSE losses under all methods approach the noise floor of 0.01. We consider centralized training with the SGD optimizer as the baseline. The centralized Adam converges to the lowest MSE due to its momentum-based strategy. D-SGD [14] and D-naive-PB converge almost as well as the baseline, whereas D-naive takes twice as many rounds to achieve the same MSE. D-Adam [15] shows signs of overfitting after quick initial convergence due to the divergence of local momentum across nodes. Among the distributed approaches, D-AMSGrad [16] achieves the best convergence by incorporating distributed consensus on the local momentum terms. Although D-naive-PB with $K=1$ requires a similar total number of rounds of MP as D-AMSGrad, there is one round of MP with large messages of size $|\boldsymbol{\theta}|$ at the end of each mini-batch, which may be translated to more rounds of exchanging normal-sized messages. This result shows that D-AMSGrad is a good candidate for distributed training of GCNNs.

B. Unsupervised Learning for UWMMSE Power Allocation

A 4-layer neural architecture (UWMMSE) is constructed by unfolding 4 iterations of the weighted minimum mean-squared error (WMMSE) algorithm for power allocation, in which two constants are parameterized by two 2-layer GCNNs [4]. Trained with unsupervised learning, the UWMMSE seeks to maximize the total throughput (sum rate) of 25 transmitter-receiver pairs by computing their transmit powers based on channel state information (CSI). These 50 transceivers are randomly scattered in a 2D square with a width of 1km, and operating on a 5MHz band at the center frequency of 900MHz, with a maximum power of 5W. The training dataset contains 640,000 realizations of CSI matrices under 100 realizations of transceiver locations, generated from the urban macro path loss model [22] with Rayleigh fading, representing a wireless network with node mobility. We evaluate centralized (Adam) and distributed (D-Adam) training with a batch size of $B=64$, and a learning rate of $\alpha=10^{-2}$. As shown in Fig. 2(b), UWMMSE outperforms the baseline WMMSE-Tr (WMMSE

truncated to 4 iterations) in sum rate as training proceeds, showing the value of the learnable unfolded architecture. The effectiveness of distributed training is demonstrated by its similar sum rate as that of the centralized training.

C. Graph-based Policy Gradient Descent for Link Scheduling

We train GCNNs applied to distributed link scheduling in wireless multihop networks with orthogonal access [5], [8]. This task is formulated as finding the maximum weighted independent set (MWIS) on the conflict graph of a wireless network, in which each vertex represents a wireless link (with weights representing the utility of scheduling that link), and an edge indicates that two links cannot be activated at the same time. The MWIS problem is known to be NP-hard [23], and heuristics are used in practical link schedulers. In this application, an actor GCNN is trained to indirectly improve the quality of the solution by modifying the input vertex weights of a distributed local greedy solver (LGS) [24], which guarantees that the solution is always an independent set.

We follow the configuration and training process in [8], which involves alternatively training a 5-layer GCNN (twin) and a 3-layer actor GCNN with a set of random graphs drawn from the Erdős-Rényi model with 100 nodes and average node degree ranging from 2 to 25. Within each mini-batch ($B = 100$), the graph remains the same, and the vertex weights are drawn online from $\mathbb{U}(0, 1.0)$, emulating the online training of GCNN in a dynamic network with a topology that changes per mini-batch. The actor is trained in a fully distributed manner, while the critic GCNN is trained in a centralized manner. We use the Adam optimizer with the learning rate $\alpha = 5 \times 10^{-5}$, and the remaining hyperparameters are as in [8]. In Fig. 2(c), the ratio between the total utility achieved by the GCNN-LGS w.r.t. that of the basic LGS as a function of the number of mini-batches is presented. Although the GCNN-LGS trained in a fully distributed manner underperforms its centralized counterpart, it still outperforms the baseline LGS, demonstrating the effectiveness of our distributed training approach in a more challenging graph-based ML pipeline.

V. Conclusion and Future Steps

We presented a methodology for online training of GNNs applied to fully distributed networked systems. This approach was illustrated with examples of GCNNs, including local implementations of inference and backpropagation, as well as communication-efficient mini-batch training based on information reuse, distributed gradient descent algorithms, and message piggybacking. The effectiveness of our approach was demonstrated numerically in supervised, unsupervised, and reinforcement learning for GCNNs in wireless ad-hoc networks. Future work includes theoretical proofs of convergence, and studying the impacts of communication errors and network mobility on training performance, and potential improvements.

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. on Neural Networks and Learning Systems*, 2020.

[2] E. Chien, M. Li, A. Aportela, K. Ding, S. Jia, S. Maji, Z. Zhao, J. Duarte, V. Fung, C. Hao, *et al.*, "Opportunities and challenges of graph neural networks in electrical engineering," *Nature Reviews Electrical Engineering*, vol. 1, no. 8, pp. 529–546, 2024.

[3] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, 2019.

[4] A. Chowdhury, G. Verma, C. Rao, A. Swami, and S. Segarra, "Unfolding WMMSE using graph neural networks for efficient power allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 6004–6017, 2021.

[5] Z. Zhao, G. Verma, C. Rao, A. Swami, and S. Segarra, "Link scheduling using graph neural networks," *IEEE Trans. Wireless Commun.*, vol. 22, no. 6, pp. 3997–4012, 2023.

[6] Z. Zhao, B. Radojičić, G. Verma, A. Swami, and S. Segarra, "Biased backpressure routing using link features and graph neural networks," *IEEE Trans. Mach. Learn. Commun. Netw.*, vol. 2, pp. 1424–1439, 2024.

[7] Z. Zhao, J. Perazzone, G. Verma, and S. Segarra, "Congestion-aware distributed task offloading in wireless multi-hop networks using graph neural networks," in *IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, pp. 8951–8955, 2024.

[8] Z. Zhao, A. Swami, and S. Segarra, "Graph-based deterministic policy gradient for repetitive combinatorial optimization problems," in *Intl. Conf. Learn. Repres. (ICLR)*, 2023.

[9] Y. Shen, J. Zhang, S. H. Song, and K. B. Letaief, "Graph neural networks for wireless communications: From theory to practice," *IEEE Trans. Wireless Commun.*, vol. 22, no. 5, pp. 3554–3569, 2023.

[10] B. Li, G. Verma, T. Efimov, A. Kumar, and S. Segarra, "GLANCE: Graph-based learnable digital twin for communication networks," *arXiv preprint arXiv:2408.09040*, 2024.

[11] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, Mar. 2020.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Intl. Conf. Artif. Intel. Stat. (AISTATS)*, vol. 54, pp. 1273–1282, PMLR, Apr 2017.

[13] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Auto. Control*, vol. 54, no. 1, pp. 48–61, 2009.

[14] B. Swenson, R. Murray, H. V. Poor, and S. Kar, "Distributed stochastic gradient descent: Nonconvexity, nonsmoothness, and convergence to local minima," *J. Mach. Learn. Res.*, vol. 23, no. 328, pp. 1–62, 2022.

[15] P. Nazari, D. A. Tarzanagh, and G. Michailidis, "DADAM: A consensus-based distributed adaptive gradient method for online optimization," *IEEE Trans. Signal Process.*, vol. 70, pp. 6065–6079, 2022.

[16] X. Chen, B. Karimi, W. Zhao, and P. Li, "On the convergence of decentralized adaptive gradient methods," in *Asian Conf. Mach. Learn. (ACML)*, vol. 189 of *Proceedings of Machine Learning Research*, pp. 217–232, PMLR, Dec 2023.

[17] H. Lin, M. Yan, X. Ye, D. Fan, S. Pan, W. Chen, and Y. Xie, "A comprehensive survey on distributed training of graph neural networks," *Proceedings of the IEEE*, vol. 111, no. 12, pp. 1572–1606, 2023.

[18] Y. Shao, H. Li, X. Gu, H. Yin, Y. Li, X. Miao, W. Zhang, B. Cui, and L. Chen, "Distributed graph neural network training: A survey," *ACM Comput. Surv.*, vol. 56, Apr. 2024.

[19] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, 2019.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Intl. Conf. Learn. Repres. (ICLR)*, 2017.

[21] L. Xiao, S. Boyd, and S. Lall, "Distributed average consensus with time-varying metropolis weights," *Automatica*, vol. 1, pp. 1–4, 2006.

[22] 3rd Generation Partnership Project (3GPP), "Technical report 38.901: Study on channel model for frequencies from 0.5 to 100 GHz," tech. rep., ETSI, 2020. Release 16.

[23] C. Joo, G. Sharma, N. B. Shroff, and R. R. Mazumdar, "On the complexity of scheduling in wireless networks," *EURASIP J.Wireless Commun. and Netw.*, vol. 2010, p. 418934, Sep 2010.

[24] C. Joo and N. B. Shroff, "Local greedy approximation for scheduling in multihop wireless networks," *IEEE Trans. on Mobile Computing*, vol. 11, no. 3, pp. 414–426, 2012.