A Federated Learning Approach for Graph Convolutional Neural Networks

Andrew Campbell, Hang Liu, Anna Scaglione, and Tong Wu
Department of Electrical and Computer Engineering, Cornell Tech, Cornell University, NY USA
Emails: ac2458@cornell.edu, hl2382@cornell.edu, as337@cornell.edu, tw385@cornell.edu

2024 IEEE 13th Sensor Array and Multichannel Signal Processing Workshop (SAM) | 979-8-3503-4481-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SAM60225.2024.10636596

***Abstract*—In this paper we cast the problem of training a graph neural network based on labeled graph data in a "federated learning" scenario where different agents have access to data from a subset of the network nodes. The learning problem is not decomposable, therefore it does not lend itself to a straightforward mapping onto a distributed multi-agent protocol. We propose a multi-agent federated learning scheme which leverages the local and sparse structure of graph filters to limit the information sharing while emulating the performance of centralized training. Even though we preserve data locality and agent communication is restricted to the neighborhood level, the proposed method still converges in simulation.**

***Index Terms*—Graph convolutional neural network, multi-agent federated learning.**

I. Introduction

Graph convolutional networks (GCNs) are powerful machine learning models when applied to networked data including, but not limited to, social networks, bioinformatics, drug discovery, power and transportation networks data, etc. By integrating graph information into deep learning models, GNNs achieve superior results in node classification, link prediction, and other tasks. A typical GNN model comprises one or more graph convolutional layers, which are Graph Filters of a predetermined order, where each layer updates vertex states before feeding them to the activation functions. This process iterates across layers, enhancing the model's ability to handle complex graph structures.

Given that in many of the applications the graph structures are extremely large, there has been a flurry of papers proposing various methods to parallelize the training (see e.g. [1] for a review). The motivation of this line of work is accelerating the training when the graphs are extremely large rather than keeping observations and graph data local. Instead, our work is motivated by the latter two reasons. This is the motivation behind the popularity of "federated learning" which offers a practical approach to training neural networks across multiple locations, with the key advantage being the preservation of data locality [2]. This method not only protects privacy but also reduces the need for sending data over the network [3], [4]. The scenario we are interested in, however, does not fit the federated learning model. In our model, agents can only access a subset of the samples for the labeled data that corresponds to the nodes in a sub-graph of the network. As a result the graph convolution function itself, as well as the "loss functions," are not decomposable. Under this regime we require new ways to handle the evaluation and training.

The realm of distributed GCNs has been extensively explored, with a comprehensive review presented in [5], covering both full-batch and mini-batch training methodologies. Notably, NeuGraph [6], introduced in 2019, stands as the pioneering effort in distributed GNN training. Subsequent studies, such as [7], [8], and [9], have ventured into the federated training of GCNs. These approaches typically envisage a scenario where each client manages a segment of a larger graph. Here, clients periodically perform local updates on their subset using a semi-supervised model, with these updates being intermittently consolidated at a central server. However, these models assume a distributed framework of GCN parameters across nodes, different from our approach which relies on graph signal processing and assigns unique parameters to a singular graph rather than to individual nodes. Consequently, the decomposition techniques employed in these studies fall short, as the localized datasets do not suffice for updating parameters that depend on the entirety of the graph.

To address this challenge, our contribution introduces a novel method that eschews the decomposition of the loss function in favor of decomposing the gradient of the loss function. This gradient is derived partially from an agent's own dataset, with additional contributions from other agents. A key discovery is that the communication between agents involves only the neural network's parameters—specifically, individual scalars—which do not reveal sufficient information to perfectly reconstruct the original datasets. Moreover, our approach leverages the average consensus method on the GCNs' parameters, further ensuring the integrity of the global results. This additional step fortifies our model's capability to handle large-scale, distributed networks efficiently while maintaining a high degree of privacy and data security.

The paper is structured as follows: Section II discusses the graph decomposition and distributed training structures of GCN. Following this, a specific case study is detailed in Section III. Finally, Section IV summarizes and concludes the paper.

II. Distributed GCN

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose nodes' measurement $\boldsymbol{x}$ and topology information is distributed among R agents, with each agent sub-graph $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$, with $\boldsymbol{x} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_R]^\top$.

This work was supported in part by the DoD-ARO under Grant No. W911NF2210228 and in part by the National Science Foundation (NSF) under Grant NSF ECCS # 2210012.

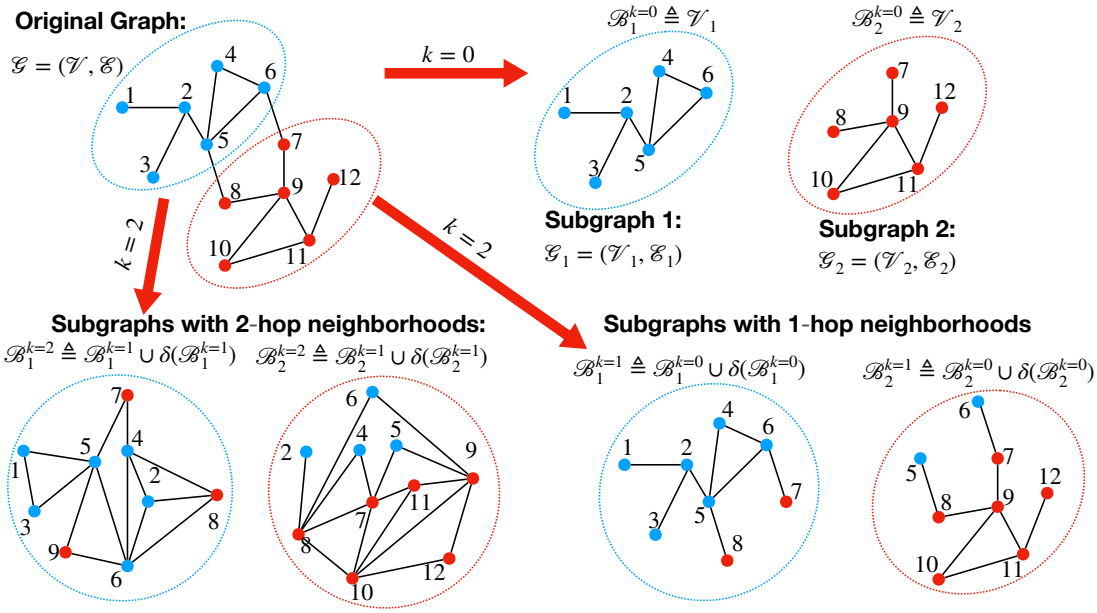

Fig. 1: Graph Decomposition of Distributed GCN.

The set of nodes $\mathcal{V}$ is the union of all sub-graphs, $\bigcup_{r=1}^{R} \mathcal{V}_r$, and the nodes in different sub-graphs are disjoint sets, i.e., $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$ for all distinct $i, j \in [R]$. In the following, to help understand the basic algorithm and avoid a cumbersome notation, we start explaining how a two-agent model $R = 2$ would work. In this case, the two agents observe two distinct subsets of graph signal values $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Again, for simplicity, in Section II-A we first show how, with message passing among the two agents, it is possible to produce the outputs of a given two-layer Chebyshev GNN, with only the first layer parametrized as a Graph Filter and the second layer fully connected. Then, in Sections II-B we show how the loss function stochastic gradient evaluation can be decomposed and how updates can be performed with message passing.

A. Framework for Distributed GCN: Evaluation

When we break down the graph into two parts by setting $R = 2$, the resulting decomposition of $\mathbf{S}$ is as follows:

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}. \tag{1}$$

The graph convolutions for the first and second graphs:

$$\mathbf{S}_1 = \begin{bmatrix} \mathbf{S}_{11} \\ \mathbf{S}_{21} \end{bmatrix}, \mathbf{S}_2 = \begin{bmatrix} \mathbf{S}_{12} \\ \mathbf{S}_{22} \end{bmatrix} \tag{2}$$

For simplicity of notation, we denote the second-order graph convolutions as follows[1]:

$$\mathbf{S}^2 = \begin{bmatrix} \mathbf{S}_{11}^2 & \mathbf{S}_{12}^2 \\ \mathbf{S}_{21}^2 & \mathbf{S}_{22}^2 \end{bmatrix}, \mathbf{S}_1^2 = \begin{bmatrix} \mathbf{S}_{11}^2 \\ \mathbf{S}_{21}^2 \end{bmatrix}, \mathbf{S}_2^2 = \begin{bmatrix} \mathbf{S}_{12}^2 \\ \mathbf{S}_{22}^2 \end{bmatrix}, \tag{3}$$

See Fig 1 for a visualization of this partitioning.

[1]Here we abuse the notation, by denoting the blocks of the square of the GSO matrix $\mathbf{S}^2$ as $\mathbf{S}_{ij}^2$, while the expression of the corresponding blocks is clearly not the square of the original ones.

Therefore, with $K = 2$, the graph signals $\boldsymbol{v}$ is:

$$\boldsymbol{v} = \sum_{k=0}^{K=2} h_k \mathbf{S}^k \boldsymbol{x} = \boldsymbol{v}_1 + \boldsymbol{v}_2 \tag{4}$$
$$= (h_0 \mathbf{I}_1 + h_1 \mathbf{S}_1 + h_2 \mathbf{S}_1^2)\boldsymbol{x}_1 + (h_0 \mathbf{I}_2 + h_1 \mathbf{S}_2 + h_2 \mathbf{S}_2^2)\boldsymbol{x}_2,$$

where h_0, h_1, h_2 are trainable parameters. Assuming a two-layer neural network configuration, the second layer is a fully-connected affine layer:

$$\boldsymbol{y} = \mathbf{W}\sigma(\boldsymbol{v}_1 + \boldsymbol{v}_2), \mathbf{W} \in \mathbb{R}^{M \times N}, \boldsymbol{y} \in \mathbb{R}^{M \times 1}, \tag{5}$$

where $\boldsymbol{y}$ denotes the regression labels. From (5) it is clear that the output of the graph filter (first layer) can be computed through a network exchange that allows computing the sum $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and by only knowing information about the network 2-hop connections. That is, it is sufficient to compute $h_0\mathbf{I} + h_1\mathbf{S}_r + h_2\mathbf{S}_r^2$ for $r = 1, 2$. Upon obtaining $\boldsymbol{v}_1 + \boldsymbol{v}_2$, one can apply the activation function and use the same set of weights $\mathbf{W}$ to generate the output of the second layer. Note that for the sake of evaluation, it is simply necessary to know the Graph Filter columns corresponding to entries of the graph signal that are accessed by each agent. However, for the sake of training, it is clear that both agents will have to learn and agree on the same set of parameters h_0, h_1, h_2 and $\mathbf{W}$. The extension to a multi-agent setup comes naturally: it is sufficient to replace the exact evaluation of the vector $\boldsymbol{v}$ at each agent with a scaled convex combination of the neighbors' terms $\boldsymbol{v}_r$:

$$\hat{\boldsymbol{v}}_r^{(0)} = \boldsymbol{v}_r = \sum_{k=0}^{2} h_k \mathbf{S}_r^k \boldsymbol{x}_r \tag{6}$$

$$\hat{\boldsymbol{v}}_r^{(t,i)} = R \sum_{r'=1}^{N(r)} a_{r,r'} \boldsymbol{v}_{r'}^{(t,i)}, \tag{7}$$

where $N(r)$ denotes the neighbour set of node r containing r itself, and the matrix $\mathbf{A} = [a_{rr'}]$ contains the mixing

coefficients. Here i refers to the consensus round while t refers to the training iteration. We can consider the simplest case where the graph is strongly connected and $\mathbf{A}$ is a doubly stochastic matrix; then asymptotically for all nodes $r = 1, \ldots, R$, $\hat{\boldsymbol{v}}_r^{(t,\infty)} = \sum_{r=1}^{R} \boldsymbol{v}_r^{(t)}$. It is well known that this communication model is an instance of the average consensus algorithm and that the latter allows for many variations, with asynchronous communications, switching typologies, etc. [10]. To understand how to minimize the loss with a stochastic gradient descent approach it is useful to first understand how it is possible to decompose the loss function. We focus on the Mean Squared Error (MSE) loss function, leaving other cases as future work.

B. Graph and Gradient Decomposition

Let us consider a regression problem where the label data are also partitioned $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2]^\top$. This is the example that we are going to consider in our simulations, where each of the agents is trying to reconstruct some missing samples that pertain to its own nodal values. Let $\boldsymbol{w}_r = \text{vec}(\boldsymbol{W}_r)$ contain the coefficients of the vectorized sub-matrix $\boldsymbol{W}_r$. The equations above imply that the loss function can be written as follows:

$$\begin{aligned} \ell &= \|\mathbf{W}_1\sigma(\boldsymbol{v}_1 + \boldsymbol{v}_2) - \boldsymbol{y}_1\|_2^2 + \|\mathbf{W}_2\sigma(\boldsymbol{v}_1 + \boldsymbol{v}_2) - \boldsymbol{y}_2\|_2^2 \\ &= \ell_1 + \ell_2 \\ \Rightarrow \quad \frac{\partial \ell}{\partial h_k} &= \sum_{r=1}^{R} \frac{\partial \ell_r}{\partial h_k}, \quad \frac{\partial \ell}{\partial \boldsymbol{w}_k} = \frac{\partial \ell_r}{\partial \boldsymbol{w}_r}. \end{aligned} \tag{8}$$

The gradient of $\frac{\partial \ell_r}{\partial h_k}$ is:

$$\begin{aligned} \frac{\partial \ell_r}{\partial h_k} &= \frac{\|\mathbf{W}_1\sigma(\boldsymbol{v}) - \boldsymbol{y}_1\|_2^2}{\partial h_k} \\ &= \left[\frac{\partial \sigma(\boldsymbol{v})}{\partial h_k}\right]^T [2 \overbrace{\mathbf{W}_1^T(\mathbf{W}_1\sigma(\boldsymbol{v}) - \boldsymbol{y}_1)}^{\mathbb{R}^{N\times 1}}], \end{aligned} \tag{9}$$

where:

$$\frac{\partial \sigma(\boldsymbol{v})}{\partial h_k} = \frac{\partial \sigma(\boldsymbol{v})}{\partial \boldsymbol{v}} \odot \frac{\partial \boldsymbol{v}}{\partial h_k} = \sum_{r=1}^{R} \frac{\partial \sigma(\boldsymbol{v})}{\partial \boldsymbol{v}} \odot \frac{\partial \boldsymbol{v}_r}{\partial h_k}. \tag{10}$$

The gradient of the ReLU activation function $\sigma'(\boldsymbol{w})$, is:

$$\sigma'(\boldsymbol{v}) := \frac{\partial \sigma(\boldsymbol{v})}{\partial \boldsymbol{v}} = \begin{cases} 0 & \text{if } v_i < 0 \\ 1 & \text{if } v_i > 0 \end{cases} \tag{11}$$

which depends on the sum of all the $\boldsymbol{v}_r$ with

$$\frac{\partial \boldsymbol{v}_r}{\partial h_k} = \mathbf{S}_r^k \boldsymbol{x}_r \tag{12}$$

Let us define:

$$u_{rk} := [\sigma'(\boldsymbol{v}) \odot \mathbf{S}_r^k \boldsymbol{x}_r]^T 2\mathbf{W}_r^T(\mathbf{W}_r\sigma(\boldsymbol{v}) - \boldsymbol{y}_r), \tag{13}$$

the gradient of the total loss with respect to h_k can be expressed as:

$$\frac{\partial \ell}{\partial h_k} = \sum_{r=1}^{R} u_{rk}. \tag{14}$$

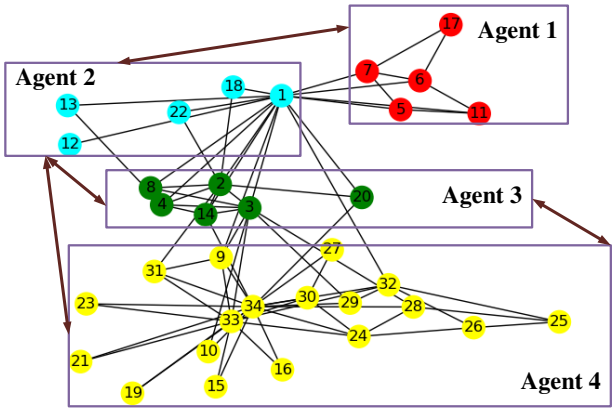

Fig. 2: Illustration of Zachary's Karate Club network, where nodes are grouped by color to indicate their respective agents and black lines denote edges between the nodes. Double-headed arrows between pairs of agents represent communication links that enable the information exchange across the agents.

The terms in (14) can also be evaluated through consensus averaging, which is preferable to exchange the full information about $\mathbf{S}_r^k \boldsymbol{x}_r$, for scalability and privacy reasons. Similar to (6), the consensus aggregation at agent r in the t-th training round is given by

$$\hat{u}_{rk}^{(t,i)} = R \sum_{r'=1}^{N(r)} a_{r,r'} u_{r'k}^{(t,i)}, \forall k. \tag{15}$$

In addition, the updates on the parameters $\boldsymbol{w}_r$ can be calculated locally using the values of $\sigma(\boldsymbol{v})$ that can be computed by the consensus algorithm operating on $\boldsymbol{v}_r$.

C. Multi-Agent Federated Learning Algorithm

The proposed multi-agent training algorithm first initializes the local model $\boldsymbol{w}_r^{(0)}, [h_0^{(0)}, \ldots, h_K^{(0)}]$ for every agent r. Throughout the training iterations, labeled as $t = 1, 2, \cdots$, our algorithm consists of the following steps for each agent r:

1) **Local forward computation**: Compute $\boldsymbol{v}_r^{(t)} = \sum_{k=0}^{K} h_k^{(t)} \mathbf{S}_r^k \boldsymbol{x}_r$;
2) **Local model aggregation**: Communicate $\boldsymbol{v}_r^{(t)}$ with the neighbors in $N(r)$ and collect $\boldsymbol{v}_{r'}^{(t)}$ from all the neighbors $\forall r' \in N(r)$ to compute $\hat{\boldsymbol{v}}_r^{(t)}$ in (6);
3) **Local backward computation**: Compute $\sigma(\hat{\boldsymbol{v}}_r^{(r)})$, followed by calculating the gradients $u_{rk}^{(t)}$ and $\frac{\partial \ell_r}{\partial \boldsymbol{w}_r}$;
4) **Local gradient aggregation**: Communicate $u_{rk}^{(t)}$ to the neighbors and average $u_{r'k}^{(t)}$ received from $\forall r' \in N(r)$ to compute $\frac{\partial \ell}{\partial h_k}$ in (15);
5) **Model update**: Perform a gradient descent step to update $\boldsymbol{w}_r^{(t+1)}$ and $[h_0^{(t+1)}, \ldots, h_K^{(t+1)}]$.

III. Numerical Results

In this section, we evaluate the performance of our algorithm on a missing data recovery task. We simulate graph-

Fig. 3: Heat map on the normalized values of graph signals within the range of $[0, 1]$. Left: the ground-truth data. Middle: the model input with 30% of the data missing. Right: the interpolation result achieved by our algorithm.

structure data generated over Zachary's Karate Club network [11]. The network comprises 34 nodes, each generating 1000 independent and identically distributed (i.i.d.) filtered graph signals $\{\boldsymbol{g}[t]\}_{t=1}^{1000}$ through a diffusion-dynamic graph filter given by [12]

$$\boldsymbol{g}[t] = (\mathbf{I} + 0.1\mathbf{S})^{-1}\boldsymbol{x}[t] + \mathbf{n}[t], \forall t, \tag{16}$$

where $\mathbf{S}$ is the graph Laplacian matrix, $\boldsymbol{x}[t]$ is the excitation signal with i.i.d. entries uniformly distributed over $[-1, 1]$, and $\boldsymbol{n}[t]$ representing the Gaussian measurement noise with entries drawn from $\mathcal{N}(0, 0.01)$. These filtered graph signals $\{\boldsymbol{g}[t]\}_{t=1}^{1000}$ serve as the training outputs in our missing data recovery model. Neither the graph filter nor the excitation is known to the agents, but the graph filter introduces correlation among the graph signal observations that enables the interpolation of the missing values. The training inputs, denoted by $\{\boldsymbol{m}[t]\}_{t=1}^{1000}$, are generated by randomly removing a single entry of $\{\boldsymbol{g}[t]\}$.

As shown in Fig. 2, we partition the network into four agents by the K-Means clustering algorithm on the Fiedler vector of the graph Laplacian matrix. We assume that each agent gets full access to the portion of graph signals within their respective sub-graph. However, communication between agents is contingent upon the existence of connecting edges among nodes across these sub-graphs. The agents aim to develop a two-layer GCN presented in Section II-A with $K = 2$ for interpolating the missing data in $\{\boldsymbol{m}[t]\}_{t=1}^{1000}$. To this end, the agents train the GCN model by using the algorithm proposed in Section II-C with the goal of minimizing the MSE loss. In each training iteration, we calculate the gradient with respect to one training input-output pair $(\boldsymbol{m}[t], \boldsymbol{g}[t])$ and perform single-round model and gradient aggregation for each agent by (6) and (15) with the following mixing matrix

$$\mathbf{A} = \begin{bmatrix} 0.75 & 0.25 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0.25 & 0.25 & 0.5 \end{bmatrix}.$$

In Fig. 3, we present the interpolation performance of our algorithm on a training sample with 30% of its data missing. The results demonstrate that the model trained by our algorithm can effectively interpolate the missing data in the graph signals. We note that our training is achieved without exchanging the local data itself but by sharing only partial model parameters and local gradients among the agents. Notably, this induces aggregation error during the consensus step which results in a slower convergence as demonstrated by Fig 4. Fig. 4 plots the training loss of our algorithm over training iterations. We include the loss of centralized training for comparison, which assumes the existence of a central server responsible for training the entire GCN model with unrestricted access to the complete dataset. We see that our distributed training algorithm attains a comparable convergence rate with that of the centralized baseline.

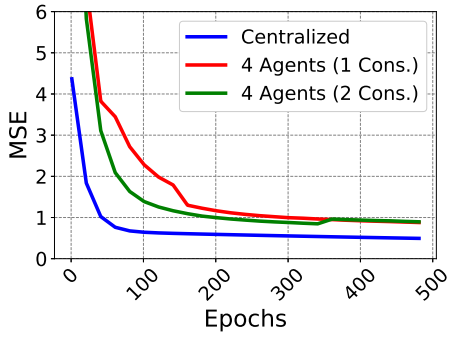

Fig. 4: MSE loss versus training iteration. The red curve includes a single round of consensus aggregation while the green curve contains two rounds.

IV. Conclusion

In this work, we introduced a distributed GCN framework for model training involving graph-structured data. This framework decomposes the gradient computation of the global model through local aggregation of intermediate model parameters and gradients during the forward and backward propagation processes. We proposed a multi-agent distributed training algorithm that replaces traditional inter-agent data sharing by local model aggregation, thereby enhancing data privacy and security. Through numerical experiments conducted on a missing data recovery task, we demonstrated that our distributed algorithm achieves comparable training efficiency to centralized training methods.

REFERENCES

[1] 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," *arXiv preprint arXiv:2211.00216*, 2022.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat.,*, Apr. 2017, pp. 1273–1282.

[3] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-iid data: A survey," *Neurocomputing*, vol. 465, pp. 371–390, Nov. 2021.

[4] H. Yang, M. Fang, and J. Liu, "Achieving linear speedup with partial worker participation in non-iid federated learning," *Proc. Int. Conf. Learn. Represent. (ICLR)*, pp. 1–23, 2021.

[5] 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," *Proc. IEEE*, vol. 111, no. 12, pp. 1572–1606, Dec. 2023.

[6] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "NeuGraph: Parallel deep neural network computation on large graphs," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 443–458.

[7] C. He *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *arXiv preprint arXiv:2104.07145*, 2021.

[8] K. Zhang, C. Yang, X. Li, L. Sun, and S. M. Yiu, "Subgraph federated learning with missing neighbor generation," *Proc. Conf. Neural Inf. Process. Syst.*, vol. 34, pp. 6671–6682, 2021.

[9] Y. Yao, W. Jin, S. Ravi, and C. Joe-Wong, "FedGCN: Convergence and communication tradeoffs in federated training of graph convolutional networks," *arXiv preprint arXiv:2201.12433*, 2022.

[10] M. Mirzaei, H. Atrianfar, N. Mehdipour, and F. Abdollahi, "Asynchronous consensus of continuous-time lagrangian systems with switching topology and non-uniform time delay," *Robotics and Autonomous Systems*, vol. 83, pp. 106–114, 2016.

[11] W. Zachary, "An information flow model for conflict and fission in small groups," *J. Anthropol. Res.*, vol. 33, no. 4, pp. 452–473, 1977.

[12] R. Ramakrishna, H.-T. Wai, and A. Scaglione, "A user guide to low-pass graph signal processing and its applications: Tools and applications," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 74–85, Nov. 2020.