# GNN-based Neighbor Selection and Resource Allocation for Decentralized Federated Learning

Chuiyang Meng*, Ming Tang†, Mehdi Setayesh*, and Vincent W.S. Wong*
*Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, Canada
†Department of Computer Science and Engineering, Southern University of Science and Technology, China
email: {chuiyangmeng, setayeshm, vincentw}@ece.ubc.ca, tangm3@sustech.edu.cn

***Abstract*—Decentralized federated learning (DFL) enables clients to train a neural network model in a device-to-device (D2D) manner without central coordination. In practical systems, DFL faces challenges due to the dynamic topology changes, time-varying channel conditions, and limited computational capability of devices. These factors can affect the performance of DFL. To address the aforementioned challenges, in this paper, we propose a graph neural network (GNN)-based approach to minimize the total delay on training and improve the learning performance of DFL in D2D wireless networks. In our proposed approach, a multi-head graph attention mechanism is used to capture different features of clients and channels. We design a neighbor selection module which enables each client to select a subset of its neighbors for the participation of model aggregation. We develop a decoder which enables each client to determine its transmit power and CPU frequency. Experimental results show that our proposed algorithm can achieve a lower total delay on training when compared with three baseline schemes. Furthermore, the proposed algorithm achieves similar performance on the testing accuracy when compared with the full participation scheme.**

## I. Introduction

Federated learning (FL) is a distributed learning framework that allows clients to cooperatively train a global model under the coordination of a central server. The conventional centralized FL schemes are vulnerable to single-point failure since the central server has to be available for aggregating the local models sent by the clients. Moreover, the wireless links between the clients and the base station may sometimes be unreliable due to fading and interference, which increases the delay on training for FL. To address these challenges, decentralized federated learning (DFL) [1] has emerged as an alternative, leveraging device-to-device (D2D) wireless connections and decentralized stochastic gradient descent. DFL relies on the model exchange between clients without the assistance of a central server. DFL has two distinct advantages. First, it is robust to single-point failure due to its decentralized structure. Second, DFL can reduce network congestion by exploiting D2D communications between clients, rather than relying solely on uplink transmissions to the central server.

However, DFL faces several challenges. First, clients need to exchange model updates frequently in a decentralized manner, which can lead to a longer convergence time than the conventional centralized FL schemes. The long convergence time can increase the communication and computation resource usage. Second, the heterogeneity of devices and the decentralized structure of DFL can reduce the convergence rate. Clients with lower communication and computational capabilities may become stragglers, which take longer time than other clients to send their model updates to their neighbors. They may reduce the frequency of model aggregation. Furthermore, the decentralized structure of DFL results in varying local models among clients. It can degrade the learning performance. Third, due to mobility, clients can move around. The associated D2D network topology may change over time. This can further lead to a longer time for training.

Recent works on DFL mainly focus on the algorithm design [2] [3] under the full participation schemes, where each client performs aggregation by requesting the local models from all of its neighboring clients. However, aggregating all neighboring models can be time-consuming due to the straggler issue. It can lead to a longer convergence time. Hence, an alternative approach is to consider partial client participation such that each client only selects a subset of its neighboring clients for the participation of model aggregation. Wang *et al.* in [4] proposed a matching decomposition sampling strategy. However, the work did not consider the discrepancy among clients' local models. Due to the decentralized structure of DFL, the local data samples of neighboring clients may not represent the data distribution of all clients. Model discrepancy exists across clients and therefore the DFL performance may be affected. In addition, addressing the impact of DFL performance under dynamic network topologies remains unexplored. In this paper, we aim to propose an effective neighbor selection and resource allocation approach for DFL to improve the learning performance and reduce the total delay on training.

The recent advances in graph neural networks (GNNs) enable them to be used as a tool for wireless network design [5] [6]. GNNs can be implemented in a decentralized manner to capture the properties of graphs. Some recent works have explored the use of GNNs in FL. He *et al.* in [7] proposed a serverless FL scheme for graph data. Wang *et al.* in [8] proposed graph convolutional networks for client sampling in FL. However, the aforementioned works did not consider the impact of wireless links on the performance of DFL.

In this paper, we study the following problem: How to select neighbors and allocate resources efficiently in a D2D network to optimize the performance of DFL? The problem is challenging due to the impact of dynamic topology changes, time-varying channel conditions, and heterogeneity of devices. We propose a GNN-based approach to address the aforementioned

challenges. The contributions of this paper are as follows:

- We formulate a neighbor selection and resource allocation problem for a DFL framework, which enables cooperative model training among clients in a D2D network. The formulated problem jointly minimizes the total delay on training and the differences among local models of DFL.
- We design an attention-based GNN to capture the intrinsic properties of the D2D network and the local model parameters. To achieve a good learning performance and a low delay on training, the proposed GNN enables each client to select its neighbors for model aggregation. It also includes a module to determine the transmit power and CPU frequency of each client. By training the GNN with different network topologies and channel conditions, the proposed GNN can adapt to different network settings.
- We conduct experiments using CIFAR-10 dataset to compare our proposed algorithm with three baseline schemes, including full participation, random sampling, and a path-following FL algorithm [9]. Results show that our proposed algorithm achieves a lower total delay on training than the baselines and comparable average testing accuracy as the full participation scheme.

The rest of the paper is organized as follows. The system model and problem formulation are presented in Section II. The proposed algorithm is presented in Section III. Simulation results and performance comparison are shown in Section IV. The conclusion is given in Section V.

*Notations*: We use boldface upper case and lower case letters to denote matrices and vectors, respectively. $\mathbb{R}^{M\times N}$ denotes the set of $M\times N$ real-valued matrices. $\mathbb{1}$ denotes the indicator function, which is equal to 1 if the argument is true and equal to 0 otherwise. $|\cdot|$, $\|\cdot\|$, $\|$, $(\cdot)^\intercal$, and $\lfloor\cdot\rfloor$ denote the cardinality, norm, concatenation, transpose, and floor function, respectively.

## II. System Model and Problem Formulation

We consider a D2D wireless network with $N$ clients in a given area. The DFL process iterates for $R$ training rounds. To capture the mobility of clients, we model the D2D network as a time-varying directed graph $\mathcal{G}^r(\mathcal{V},\mathcal{E}^r)$, where $\mathcal{V}$ and $\mathcal{E}^r$ denote the set of clients (i.e., nodes) and the set of communication links (i.e., edges) between clients in the $r$-th training round, respectively. We assume that clients can move freely at the end of each training round, while they remain stationary within each training round. There is a communication link between two clients when their signal-to-interference-plus-noise ratio (SINR) is greater than or equal to a threshold $\Omega$. The value of $\Omega$ controls the node degree in the D2D network. The link from client $i$ to client $j$ is denoted by $(i,j)\in\mathcal{E}^r$. We define $\mathbf{A}^r\in\{0,1\}^{N\times N}$ as the adjacency matrix in the $r$-th training round with $\mathbf{a}_i^r\in\mathbb{R}^{1\times N}, i\in\mathcal{V}$, denoting the $i$-th row of matrix $\mathbf{A}^r$. We have $a_{i,j}^r=1$ if there is a link between clients $i$ and $j$. Otherwise, $a_{i,j}^r=0$. We define the set of neighbors of client $i$ in the $r$-th training round as $\mathcal{N}_i^r=\{j\mid j\in\mathcal{V}, a_{i,j}^r=1\}$.

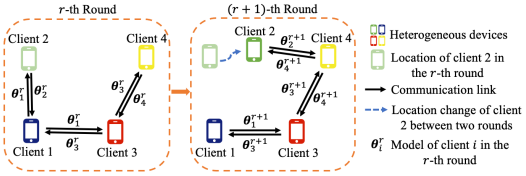


Fig. 1: An illustration of DFL executed under time-varying D2D network topologies. Each client shares its model with its neighbors after local training. In the $r$-th training round, client 2 sends its model $\boldsymbol{\theta}_2^r$ to client 1 and receives the model $\boldsymbol{\theta}_1^r$. In the $(r+1)$-th training round, client 2 moves to another location. It sends its model $\boldsymbol{\theta}_2^{r+1}$ to client 4 and receives model $\boldsymbol{\theta}_4^{r+1}$.

*1) Decentralized Federated Learning:* Each client $i\in\mathcal{V}$ has a local dataset $\mathcal{D}_i$ with $D_i$ data samples. Let parameters $\boldsymbol{\theta}_i^R\in\mathbb{R}^d$ denote the local model of client $i$ after $R$ training rounds. The loss function of the local model of client $i$ is $F_i(\boldsymbol{\theta}_i^R)=\frac{1}{D_i}\sum_{z\in\mathcal{D}_i}\ell(z;\boldsymbol{\theta}_i^R)$, where $\ell(z;\boldsymbol{\theta}_i^R)$ denotes the loss on data sample $z\in\mathcal{D}_i$ with model $\boldsymbol{\theta}_i^R$. Let $\boldsymbol{\theta}_G$ denote the global model (i.e., ground truth). It is defined as $\boldsymbol{\theta}_G=\frac{1}{\sum_{j=1}^N D_j}\sum_{i=1}^N D_i\boldsymbol{\theta}_i^R$. Note that acquiring $\boldsymbol{\theta}_G$ in practical systems may result in high communication delays. To minimize the empirical loss of the global model $\frac{1}{\sum_{j=1}^N D_j}\sum_{i=1}^N D_iF_i(\boldsymbol{\theta}_G)$, each client shares its models with its neighbors during model aggregation in each training round to achieve a high degree of consensus [10].

The DFL process is depicted in Fig. 1. In the $r$-th training round, client $i$ trains its model locally with $K$ stochastic gradient descent (SGD) steps. We define client $i$'s model at the $k$-th step as $\boldsymbol{\theta}_i^{r,k}, 1\le k\le K$. It is updated as follows:

$$\boldsymbol{\theta}_i^{r,k}=\boldsymbol{\theta}_i^{r,k-1}-\eta^r\nabla F_i(\boldsymbol{\theta}_i^{r,k-1}), \tag{1}$$

where $\eta^r$ and $\nabla F_i(\boldsymbol{\theta}_i^{r,k-1})$ denote the learning rate and the gradient of the loss function of client $i$ at the $(k-1)$-th SGD step in the $r$-th training round, respectively. After $K$ SGD steps in the $r$-th training round, client $i$'s local model is $\boldsymbol{\theta}_i^{r,K}$.

Then, clients exchange their models. In this work, each client selects a subset of its neighboring clients to participate in model aggregation (details to be presented in Section III). We define the neighbor selection decision variable as $s_{i,j}^r\in\{0,1\}$, which satisfies $s_{i,j}^r\le a_{i,j}^r, i,j\in\mathcal{V}$. The neighbor selection matrix for aggregation in the $r$-th training round is $\mathbf{S}^r\in\mathbb{R}^{N\times N}$ with $\mathbf{s}_i^r\in\mathbb{R}^{1\times N}$ denoting the neighbor selection row vector of client $i$. We define the set of neighbors that client $i$ has selected for aggregation in the $r$-th training round as $\hat{\mathcal{N}}_i^r=\{j\mid s_{i,j}^r=1, j\in\mathcal{V}\}$. $\hat{\mathcal{N}}_i^r$ is a set of indices of non-zero elements in vector $\mathbf{s}_i^r$ and set $\hat{\mathcal{N}}_i^r\subseteq\mathcal{N}_i^r$. Client $i$ then updates its local model by aggregating the local models of its selected neighbors in the $r$-th training round as follows:

$$\boldsymbol{\theta}_i^{r+1,0}(\mathbf{s}_i^r)\triangleq\sum\nolimits_{j\in\hat{\mathcal{N}}_i^r\bigcup\{i\}}\rho_{j,i}^r\boldsymbol{\theta}_j^{r,K}, \tag{2}$$

where $\rho_{j,i}^r\triangleq D_j/\sum_{n\in\hat{\mathcal{N}}_i^r\bigcup\{i\}}D_n$ is the weight of the model of client $j$ when client $i$ performs aggregation at the end of the $r$-th training round. $\boldsymbol{\theta}_i^{r+1,0}$ will then be used as the client $i$'s model for local training in the $(r+1)$-th training round.

The *consensus distance* measures the discrepancy between the locally aggregated model and the global model [10].

Reducing the consensus distance is crucial for improving the learning performance of DFL. We denote the consensus distance between the model of client $i$ and the global model after the $r$-th training round as $\mathcal{L}_i^{\mathrm{gap},r}(\mathbf{s}_i^r) = \|\boldsymbol{\theta}_i^{r+1,0}(\mathbf{s}_i^r) - \boldsymbol{\theta}_G\|_2^2$.

*2) Training Delay:* In DFL, each client needs to select its neighbors, CPU frequency, and transmit power. The chosen values affect the computation delay and communication delay. The computation delay of a client is the time that the client uses for local training. Let decision variable $f_i^r \in [f_i^{\min}, f_i^{\max}]$ denote the CPU frequency of client $i$ in the $r$-th training round, where $f_i^{\min}$ and $f_i^{\max}$ are the minimum and maximum CPU frequency of client $i$, respectively. Let $c_i$ denote the number of CPU cycles of client $i$ required to process one bit of data. Let $\epsilon_i$ denote the size of dataset $\mathcal{D}_i$ (in bits). The computation delay of client $i$ in the $r$-th training round is $T_i^{\mathrm{comp},r} = \frac{c_i \epsilon_i K}{f_i^r}$.

When client $i$ completes the local training in one training round, it sends its updated local model to its selected neighbors. Thus, the communication delay will be incurred. Let decision variable $p_i^r \in [0, p_i^{\max}]$ denote the transmit power of client $i$ in the $r$-th training round, where $p_i^{\max}$ denotes the maximum transmit power of client $i$. Let $g_{i,j}^r$ denote the channel gain from client $i$ to client $j$ in the $r$-th training round. At an arbitrary time instant $t$ within the $r$-th training round, we define the set of clients that are transmitting their updated models as $\mathcal{M}(t) \subset \mathcal{V}$. The SINR of the link from client $i$ to client $j$ at time instant $t$ in the $r$-th training round is

$$\psi_{i,j}^r(t) = \frac{|g_{i,j}^r|^2 p_i^r}{\sum_{k \in \mathcal{M}(t)\setminus\{i\}} |g_{k,j}^r|^2 p_k^r + \sigma^2}, \tag{3}$$

where $\sigma^2$ is the received noise power. The corresponding achievable data rate at time instant $t$ in the $r$-th training round is $C_{i,j}^r(t) = B\log_2(1 + \psi_{i,j}^r(t))$, where $B$ is the bandwidth.

We consider a synchronized DFL scheme, where all clients begin local training and perform model aggregation simultaneously[1]. Let $\tau^r$ denote the time that client $i$ begins to receive the model from client $j$ in the $r$-th training round. Let $\xi_{\mathrm{m}}$ denote the size of the model (in bits). The corresponding communication delay is denoted as $T_{j,i}^{\mathrm{tr},r}$. Due to selected neighbors' different transmit power and channel gains, they may complete sending model parameters to client $i$ at different time instants. Hence, the interference may change over time. Client $i$ can calculate the time instant of its selected neighboring client $j$ to obtain $T_{j,i}^{\mathrm{tr},r}$. In particular, $T_{j,i}^{\mathrm{tr},r}$ satisfies $\int_{\tau^r}^{\tau^r + T_{j,i}^{\mathrm{tr},r}} C_{j,i}^r(t)\, dt = \xi_{\mathrm{m}}$. A new training round begins only when all clients have received the updated models from their selected neighbors. The communication delay of client $i$ is defined as the total time it takes to receive the models from its selected neighbors in set $\hat{\mathcal{N}}_i^r$, which is $T_i^{\mathrm{comm},r} = \max_{j \in \hat{\mathcal{N}}_i^r}\{T_{j,i}^{\mathrm{tr},r}\}$.

Due to synchronous training, the total delay of the $r$-th training round is the sum of the maximum computation delay and maximum communication delay, which can be represented as a function of the neighbor selection matrix, transmit power, and CPU frequency of all clients. Let $\mathbf{p}^r \triangleq (p_1^r, p_2^r, \ldots, p_N^r)$ and $\mathbf{f}^r \triangleq (f_1^r, f_2^r, \ldots, f_N^r)$ denote the vectors of transmit power and CPU frequency in the $r$-th training round, respectively. The delay on training in the $r$-th training round is given by $T^r(\mathbf{p}^r, \mathbf{f}^r, \mathbf{S}^r) = \max_{i \in \mathcal{V}}\{T_i^{\mathrm{comp},r}\} + \max_{i \in \mathcal{V}}\{T_i^{\mathrm{comm},r}\}$.

We aim to jointly optimize the neighbor selection and resource allocation decisions to minimize the total delay on training and the consensus distance of DFL. Due to time-varying topologies, the neighbor selection and resource allocation decisions are independent across training rounds. Thus, we decouple the optimization problem of all training rounds into $R$ optimization problems, one for each training round. In the $r$-th training round, the problem can be formulated as

$$\begin{aligned}
\underset{\mathbf{p}^r, \mathbf{f}^r, \mathbf{S}^r}{\text{minimize}} \quad & T^r(\mathbf{p}^r, \mathbf{f}^r, \mathbf{S}^r) + \frac{\lambda}{N}\sum\nolimits_{i=1}^{N} \mathcal{L}_i^{\mathrm{gap},r}(\mathbf{s}_i^r) && \text{(4a)}\\
\text{subject to} \quad & s_{i,j}^r \le a_{i,j}^r, \quad i,j \in \mathcal{V}, && \text{(4b)}\\
& 0 \le p_i^r \le p_i^{\max}, \quad i \in \mathcal{V}, && \text{(4c)}\\
& f_i^{\min} \le f_i^r \le f_i^{\max}, \quad i \in \mathcal{V}, && \text{(4d)}
\end{aligned}$$

where $\lambda$ is a non-negative coefficient. Problem (6) aims to determine the neighbor selection matrix $\mathbf{S}^r$ and the vectors of transmit power $\mathbf{p}^r$ and CPU frequency $\mathbf{f}^r$. Constraint (4b) ensures that client $i$ can send its model to its neighbor $j$ only if there is a communication link between them. Constraints (4c) and (4d) ensure that each client adheres to the transmit power and CPU frequency limit. Solving this problem is challenging due to the coupling of the neighbor selection and resource allocation decisions. In the next section, we propose a GNN approach to solve problem (6) in a decentralized manner.

## III. Algorithm Design

We propose an attention-based GNN algorithm. The attention mechanism can help each client learn the features of its neighboring clients and determine the importance scores of them for selection. The proposed algorithm has four modules: a client encoder, a parameter encoder, a neighbor selection module, and a decoder, shown in Fig. 2. The client encoder encodes the client features (i.e., maximum transmit power, maximum and minimum CPU frequencies) and edge features (i.e., channel gains) to capture the intrinsic properties of the D2D networks. The parameter encoder generates embeddings of the model parameters. The neighbor selection module determines the set of neighboring clients for model aggregation. The decoder determines the transmit power and CPU frequency of client $i$. Let $\tilde{\mathcal{N}}_i^r = \mathcal{N}_i^r \bigcup \{i\}$ denote the modified neighboring set in the $r$-th training round. $\mathbf{x}_i^{\mathrm{ndin},r} = (p_i^{\max}, f_i^{\min}, f_i^{\max}) \in \mathbb{R}^3$ denotes the client $i$'s feature vector. The edge feature $x_{i,j}^{\mathrm{ein},r}$ is the absolute value of the channel gain of link $(i,j) \in \mathcal{E}^r$ in the $r$-th training round, i.e., $|g_{i,j}^r|$.

*1) Client Encoder:* We consider both node and edge features. The node features describe the communication and computational capabilities of devices. The edge features characterize the channel gain which affects the communication delay. The client encoder utilizes these two features to capture

[1]Similar to [11], each client can have a scheduler which sends synchronization messages to control the synchronization process. The size of a synchronization message is small when compared with the size of model update. Hence, the communication delay of synchronization is negligible.

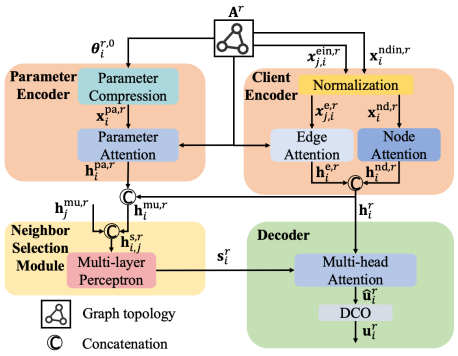


Fig. 2: A schematic of our proposed algorithm.

the intrinsic properties of the network topology. It generates the edge-enhanced embeddings with the following steps.

First, we use the $z$-score normalization to stabilize the gradients of GNN. Since each client aggregates features of different magnitudes from its neighbors, local normalization is applied to ensure that the mean and standard deviation of the features are zero and one, respectively. The $m$-th normalized node feature is $x_{i,m}^{\mathrm{nd},r} = (x_{i,m}^{\mathrm{ndin},r} - \mu_{i,m}^{\mathrm{nd},r})/\sigma_{i,m}^{\mathrm{nd},r}, 1 \le m \le 3$, where $\mu_{i,m}^{\mathrm{nd},r}$ and $\sigma_{i,m}^{\mathrm{nd},r}$ are the mean and standard deviation of the $m$-th feature of client $i$ and its neighbors. The neighboring edges of edge $(j,i)$ are the set of edges from client $i$'s neighbors to client $i$, which are defined as $(k,i) \in \mathcal{E}^r, k \in \mathcal{N}_i^r \backslash \{j\}$. The normalized edge feature is $x_{j,i}^{\mathrm{e},r} = (|g_{j,i}^r| - \mu_{j,i}^{\mathrm{e},r})/\sigma_{j,i}^{\mathrm{e},r}$, where $\mu_{j,i}^{\mathrm{e},r}$ and $\sigma_{j,i}^{\mathrm{e},r}$ are the mean and standard deviation of $(j,i)$ and its neighboring edge features, respectively.

Second, the node and edge features are sent to the node attention layer and edge attention layer to generate the node and edge embeddings, respectively. In the graph attention mechanism, the node attention score $\alpha_{i,j}^r$ assesses the importance of node $j$'s feature $\mathbf{x}_j^{\mathrm{nd},r}$ to node $i$'s feature $\mathbf{x}_i^{\mathrm{nd},r}$. Only the neighboring nodes and the corresponding edges are considered when calculating the attention scores. The multi-head attention mechanism is employed that uses a predefined $K_h$ independent sets of weights to encode the features. Then, we compute the node attention scores to generate the node embeddings. The $k_h$-th set of node attention scores are

$$\alpha_{i,j}^{r,k_h} = \frac{\exp(\sigma((\mathbf{a}^{\mathrm{nd},k_h})^\intercal [\mathbf{W}^{\mathrm{nd},k_h}\mathbf{x}_i^{\mathrm{nd},r} \| \mathbf{W}^{\mathrm{nd},k_h}\mathbf{x}_j^{\mathrm{nd},r}]))}{\sum_{k \in \tilde{\mathcal{N}}_i^r} \exp(\sigma((\mathbf{a}^{\mathrm{nd},k_h})^\intercal [\mathbf{W}^{\mathrm{nd},k_h}\mathbf{x}_i^{\mathrm{nd},r} \| \mathbf{W}^{\mathrm{nd},k_h}\mathbf{x}_k^{\mathrm{nd},r}]))}, \tag{5}$$

where $\sigma(\cdot)$ is a nonlinear activation function, which is a leaky rectified linear unit in our proposed algorithm. $\mathbf{a}^{\mathrm{nd},k_h} \in \mathbb{R}^{2d_{\mathrm{ndo}}}$ is the $k_h$-th set of weights of the fully-connected layer. $\mathbf{W}^{\mathrm{nd},k_h} \in \mathbb{R}^{d_{\mathrm{ndo}} \times 3}$ is the $k_h$-th set of weights for the node feature. $d_{\mathrm{ndo}}$ is the dimension of the node embedding $\mathbf{h}_i^{\mathrm{nd},r}, i \in \mathcal{V}$. Based on $\alpha_{i,j}^{r,k_h}$, $\mathbf{h}_i^{\mathrm{nd},r}$ is obtained by

$$\mathbf{h}_i^{\mathrm{nd},r} = \sigma\left(\frac{1}{K_h}\sum\nolimits_{k_h=1}^{K_h}\sum\nolimits_{j \in \tilde{\mathcal{N}}_i^r} \alpha_{i,j}^{r,k_h}\mathbf{W}^{\mathrm{nd},k_h}\mathbf{x}_j^{\mathrm{nd},r}\right). \tag{6}$$

Let $\mathbf{A}^{\mathrm{nd},r}$, $\mathbf{W}^{\mathrm{nd},r}$, $\mathbf{X}^{\mathrm{nd},r}$ denote the $K_h$ sets of weights of fully-connected layers, $K_h$ sets of weights for node features, and all client features, respectively. We define matrix $\tilde{\mathbf{A}}^r = \mathbf{A}^r + \mathbf{I}_N$, where $\mathbf{I}_N$ denotes the $N \times N$ identity matrix. Let $\tilde{\mathbf{a}}_i^r$ denote the $i$-th row vector of matrix $\tilde{\mathbf{A}}^r$. We represent the node embedding generation process in (5) and (6) as a function of $\mathbf{A}^{\mathrm{nd},r}$, $\mathbf{W}^{\mathrm{nd},r}$, $\mathbf{X}^{\mathrm{nd},r}$ and $\tilde{\mathbf{a}}_i^r$, which is $\mathbf{h}_i^{\mathrm{nd},r} = \mathrm{Att}(\mathbf{A}^{\mathrm{nd},r}, \mathbf{W}^{\mathrm{nd},r}, \mathbf{X}^{\mathrm{nd},r}, \tilde{\mathbf{a}}_i^r)$, where $\mathbf{A}^{\mathrm{nd},r} \in \mathbb{R}^{2d_{\mathrm{ndo}} \times K_h}$, $\mathbf{W}^{\mathrm{nd},r} \in \mathbb{R}^{d_{\mathrm{ndo}} \times 3 \times K_h}$, and $\mathbf{X}^{\mathrm{nd},r} \in \mathbb{R}^{3 \times N}$. Let $\mathbf{h}_i^{\mathrm{e},r} \in \mathbb{R}^{d_{\mathrm{eo}}}$ denote the embedding of the edge $(j,i)$, $j \in \mathcal{N}_i^r$. $d_{\mathrm{eo}}$ is the dimension of $\mathbf{h}_i^{\mathrm{e},r}$. By using the attention mechanism, $\mathbf{h}_i^{\mathrm{e},r}$ can be expressed as $\mathbf{h}_i^{\mathrm{e},r} = \mathrm{Att}(\mathbf{A}^{\mathrm{e},r}, \mathbf{W}^{\mathrm{e},r}, \mathbf{x}^{\mathrm{e},r}, \mathbf{a}_i^r)$, where $\mathbf{A}^{\mathrm{e},r} \in \mathbb{R}^{2d_{\mathrm{eo}} \times K_h}$, $\mathbf{W}^{\mathrm{e},r} \in \mathbb{R}^{d_{\mathrm{eo}} \times 1 \times K_h}$. $\mathbf{x}^{\mathrm{e},r} \in \mathbb{R}^{|\mathcal{E}^r|}$ includes the edge features of all clients. The edge-enhanced node embedding of client $i$ is obtained by concatenating the node and edge embeddings, which is $\mathbf{h}_i^r = (\mathbf{h}_i^{\mathrm{nd},r} \| \mathbf{h}_i^{\mathrm{e},r})$.

*2) Parameter Encoder:* The value of the loss function of client $i$ in DFL is a function of the model parameters of client $i$ and its neighbors. To obtain the information of local models, we design a parameter encoder to generate the embeddings of model parameters. It uniformly samples parameters based on a ratio $\kappa$ and uses the compressed parameters as input for the parameter attention layer. Let $\mathbf{X}^{\mathrm{pa},r} \in \mathbb{R}^{\lfloor \kappa d \rfloor \times N}$ denote the compressed model parameters of all clients. Let $\mathbf{h}_i^{\mathrm{pa},r} \in \mathbb{R}^{d_{\mathrm{pao}}}$ denote the parameter embedding of client $i$. $d_{\mathrm{pao}}$ is the dimension of $\mathbf{h}_i^{\mathrm{pa},r}$. $\mathbf{h}_i^{\mathrm{pa},r}$ can be obtained by $\mathbf{h}_i^{\mathrm{pa},r} = \mathrm{Att}(\mathbf{A}^{\mathrm{pa},r}, \mathbf{W}^{\mathrm{pa},r}, \mathbf{X}^{\mathrm{pa},r}, \tilde{\mathbf{a}}_i^r)$, where $\mathbf{A}^{\mathrm{pa},r} \in \mathbb{R}^{2d_{\mathrm{pao}} \times K_h}$, $\mathbf{W}^{\mathrm{pa},r} \in \mathbb{R}^{d_{\mathrm{pao}} \times \lfloor \kappa d \rfloor \times K_h}$. We concatenate $\mathbf{h}_i^r$ and $\mathbf{h}_i^{\mathrm{pa},r}$ as $\mathbf{h}_i^{\mathrm{mu},r} = (\mathbf{h}_i^r \| \mathbf{h}_i^{\mathrm{pa},r})$. It is sent to the neighbor selection module as input.

*3) Neighbor Selection Module:* When performing training in DFL, the full participation scheme may incur high communication delays. Thus, we propose a neighbor selection module to determine which neighbors to be selected by each client. The inputs to the module consist of the concatenated embedding of a client and the embeddings of its neighbors. They are concatenated and sent to the module sequentially. The concatenated embedding of client $i$ and its neighbor $j$ is $\mathbf{h}_{i,j}^{\mathrm{s},r} = (\mathbf{h}_i^{\mathrm{mu},r} \| \mathbf{h}_j^{\mathrm{mu},r})$.

We apply a multi-layer perceptron module to determine whether a particular neighbor is selected. The output selection is $b_{i,j}^r = \sigma_{\mathrm{s}}(\mathbf{W}_i^{\mathrm{s},r}\mathbf{h}_{i,j}^{\mathrm{s},r})$, where $\mathbf{W}_i^{\mathrm{s},r} \in \mathbb{R}^{2(d_{\mathrm{ndo}}+d_{\mathrm{eo}}+d_{\mathrm{pao}})}$ denotes the weights of the neighbor selection module. $\sigma_{\mathrm{s}}(\cdot)$ is the sigmoid function that generates the normalized probability vector. The module determines whether to select a neighbor for model aggregation. For each neighboring client $j \in \mathcal{N}_i^r$, client $i$ determines the selection decision $s_{i,j}^r$ based on a pre-defined threshold $\gamma \in (0,1)$. Since bi-directional model exchanges are required for model aggregation in DFL, we have

$$s_{i,j}^r = \mathbb{1}(b_{i,j}^r \ge \gamma \text{ and } b_{j,i}^r \ge \gamma). \tag{7}$$

After all neighbors' embeddings have been sent to the module, client $i$ can generate the selection vector $\mathbf{s}_i^r$. The neighbor selection matrix $\mathbf{S}^r$ can then be constructed.

*4) Decoder:* The decoder determines the transmit power and CPU frequency for client $i$ by using the edge-enhanced embeddings generated by the encoders and the neighbor selection vector generated by the neighbor selection module,

respectively. We define $\hat{\mathbf{S}}^r = \mathbf{S}^r + \mathbf{I}_N$. $\hat{\mathbf{u}}_i^r = (\hat{p}_i^r, \hat{f}_i^r)$ denotes the resource allocation decision of client $i$. It is obtained by

$$\hat{\mathbf{u}}_i^r = \mathrm{Att}(\hat{\mathbf{A}}^{\mathrm{nd},r}, \hat{\mathbf{W}}^{\mathrm{nd},r}, \mathbf{H}^r, \hat{\mathbf{s}}_i^r), \tag{8}$$

where $\hat{\mathbf{A}}^{\mathrm{nd},r} \in \mathbb{R}^{4\times K_h}$, $\hat{\mathbf{W}}^{\mathrm{nd},r} \in \mathbb{R}^{2\times(d_{\mathrm{ndo}}+d_{\mathrm{eo}})\times K_h}$. $\mathbf{H}^r \in \mathbb{R}^{(d_{\mathrm{ndo}}+d_{\mathrm{eo}})\times N}$ contains the edge-enhanced embeddings of all clients. Note that $\hat{\mathbf{u}}_i^r$ may be infeasible. Hence, we add a differentiable convex optimization (DCO) layer [12] to the last part of the decoder to generate the feasible output $\mathbf{u}_i^r = (p_i^r, f_i^r)$ by solving the following problem:

$$\begin{aligned} \underset{\mathbf{u}_i^r}{\text{minimize}} \quad & \|\mathbf{u}_i^r - \hat{\mathbf{u}}_i^r\|_2^2 \\ \text{subject to} \quad & \text{constraints (4c) and (4d).} \end{aligned} \tag{9}$$

*5) Loss Function and Training Algorithm:* We define the loss function as a weighted sum of the delay and the consensus distance. Given $\mathbf{s}_i^r$, the aggregated compressed parameters of client $i$ in the $r$-th training round can be represented by $\mathbf{x}_i^{\mathrm{pa},r+1}(\mathbf{s}_i^r) = \sum_{j\in\hat{\mathcal{N}}_i^r\bigcup\{i\}} \rho_{j,i}^r \mathbf{x}_j^{\mathrm{pa},r}$. It is used to approximate the consensus distance between the full model parameters of client $i$ and the average of all model parameters after neighbor selection. We define $\bar{\mathbf{x}}^{\mathrm{pa},r} = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i^{\mathrm{pa},r}$. In order to obtain $\bar{\mathbf{x}}^{\mathrm{pa},r}$ for training, we need to collect $\mathbf{x}_i^{\mathrm{pa},r}, i \in \mathcal{V}$. Since we use uniformly sampled parameters, the consensus distance is given by $\mathcal{L}_i^{\mathrm{gap},r}(\mathbf{s}_i^r) \approx \|\mathbf{x}_i^{\mathrm{pa},r+1}(\mathbf{s}_i^r) - \bar{\mathbf{x}}^{\mathrm{pa},r}\|_2^2/\kappa$. Since $\mathcal{L}_i^{\mathrm{gap},r}$ requires only $\mathbf{x}_i^{\mathrm{pa},r+1}, i \in \mathcal{V}$, the communication delay during GNN training can be minor. After we have obtained the transmit power and CPU frequency, the total delay of client $i$ in the $r$-th training round is given by $T_i^r = T_i^{\mathrm{comp},r} + T_i^{\mathrm{comm},r}$. The loss function of client $i$ is

$$\mathcal{L}_i^r = T_i^r + \lambda \mathcal{L}_i^{\mathrm{gap},r}. \tag{10}$$

To ensure that the proposed algorithm can adapt to different network settings, we train our GNN in an offline and unsupervised manner. We generate a set of D2D network scenarios with topology changes and time-varying channel conditions. Due to the small size of the neighbors' node features, compressed model parameters, and channel conditions, each client collects these features in each scenario with negligible communication delay. It then determines the decision and sends the decision to its neighbors and the loss of each client can be calculated. Let $\mathbf{W}^{\mathrm{s},r} = \{\mathbf{W}_i^{\mathrm{s},r}, i \in \mathcal{V}\}$ denote the set of weights of the neighbor selection module. Let $\mathbf{\Phi}^r = \{\mathbf{A}^{\mathrm{nd},r}, \mathbf{A}^{\mathrm{e},r}, \mathbf{A}^{\mathrm{pa},r}, \hat{\mathbf{A}}^{\mathrm{nd},r}, \mathbf{W}^{\mathrm{nd},r}, \mathbf{W}^{\mathrm{e},r}, \mathbf{W}^{\mathrm{s},r}, \mathbf{W}^{\mathrm{pa},r}, \hat{\mathbf{W}}^{\mathrm{nd},r}\}$ denote the set of parameters of GNN. Then $\mathbf{\Phi}^r$ is updated based on each loss. The adjacency matrix serves as a mask matrix. It ensures that the weights of non-neighboring clients are not considered during the update for a particular client. Let $\mathbf{\Phi}^{\star,r}$ denote the parameters of GNN after training. The training algorithm is given in Algorithm 1.

## IV. Performance Evaluation

We consider a scenario where the clients are randomly located in a $2 \times 2\ \mathrm{km}^2$ coverage area. To model the mobility of clients, each client can move in an arbitrary direction. The traveling distance of each client $i \in \mathcal{V}$ between two training rounds follows a uniform distribution $\triangle d_i \sim \mathcal{U}[0,\ 100\,\mathrm{m}]$. To depict the heterogeneity of devices, the maximum transmit power of each device $i$ follows $p_i^{\max} \sim \mathcal{U}[6.667\,\mathrm{mW},\ 13.333\,\mathrm{mW}]$. The minimum and maximum CPU frequency of each device $i$ follow $f_i^{\min} \sim \mathcal{U}[0.067\,\mathrm{GHz},\ 0.133\,\mathrm{GHz}]$, $f_i^{\max} \sim \mathcal{U}[1.667\,\mathrm{GHz},\ 2.333\,\mathrm{GHz}]$, respectively. We initialize the transmit power and CPU frequency of each client to the maximum value and implement our trained GNN. We consider Rayleigh fading channel model. Each client can independently generate their decisions and obtain the delay by recording the wall clock time. Unless stated otherwise, the number of clients $N$ is set to 30. A list of key simulation parameters is given in Table I. We use CIFAR-10 as the dataset and allocate the same number of data samples to the local dataset of each client in all cases. We adopt the convolutional neural networks with three convolutional layers, two fully-connected layers, and a dropout layer in DFL. We compare the performance of our algorithm with the following baseline schemes:

- **Random sampling DFL (RS-DFL) scheme:** Neighbors are randomly selected by each client based on a certain ratio $\zeta$, where $0 < \zeta < 1$. Then, the resources are allocated using our proposed client encoder and decoder.
- **Full participation DFL (FP-DFL) scheme:** Each client performs local aggregation based on the model obtained from all of its neighbors. The power and CPU frequencies are allocated to the clients using our pre-trained GNN.
- **Path-following FL (PF-FL) scheme [9]:** Problem (6) is solved by using the centralized path-following algorithm in [9] without the energy consumption constraint. The algorithm uses the full participation scheme and iteratively solves the problem by transforming the nonconvex problem into a computationally tractable convex form.

In Fig. 3, we compare the average testing accuracy of all clients under different schemes. Results show that our proposed algorithm outperforms the RS-DFL scheme when $\zeta = 0.4$ and 0.6. It achieves an average testing accuracy

**Algorithm 1** Training Algorithm

**Input:** Normalized client features $\mathbf{X}^{\mathrm{nd},r}$, normalized edge features $\mathbf{x}^{\mathrm{e},r}$, compressed model parameters $\mathbf{X}^{\mathrm{pa},r}$, total number of training epochs $N_{\mathrm{ep}}$, pre-defined threshold $\gamma$, and initialized parameters of GNN $\mathbf{\Phi}^r$.

```
for n = 1 to N_ep do
    Select a batch of network scenarios for training.
    for i ∈ V do
        Obtain s_i^r and û_i^r based on eqns. (7) and (8).
        Determine u_i^r by solving problem (9).
        Calculate the loss based on eqn. (10).
    end for
    Update Φ^r using the SGD optimizer.
end for
```

**Output:** Trained parameters of GNN $\mathbf{\Phi}^{\star,r}$.

Table I: List of key simulation parameters

| Para. | Value | Para. | Value | Para. | Value | Para. | Value |
|---|---|---|---|---|---|---|---|
| $B$ | 0.1 MHz | $\Omega$ | $-25$ dB | $N_{\mathrm{ep}}$ | 200 | $\xi_{\mathrm{m}}$ | 160 kB |
| $K_h$ | 4 | $K$ | 20 | $\gamma$ | 0.4 | $\lambda$ | 0.5 |
| $\sigma^2$ | $3.98\times10^{-13}$ mW | $c_i$ | 20 | $\kappa$ | 0.03 | $R$ | 300 |
| $d_{\mathrm{ndo}}$ | 3 | $d_{\mathrm{eo}}$ | 2 | $d_{\mathrm{pao}}$ | 20 | $\epsilon_i$ | 2.61 MB |

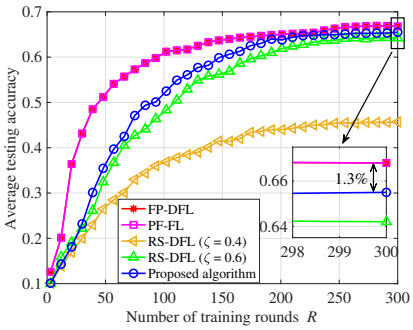


Fig. 3: Comparison of the average testing accuracy versus the number of training rounds.

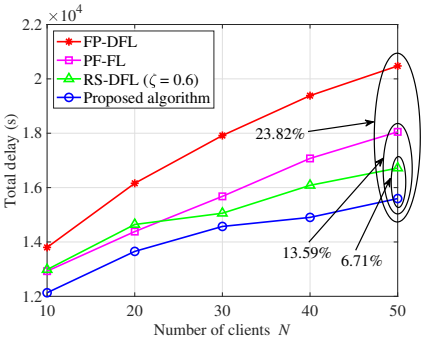


Fig. 4: Comparison of the total delay on training versus the number of clients.

comparable to FP-DFL, with a gap of only 1.3%. The PF-FL and FP-DFL schemes have the same learning performance since they both apply the full participation scheme.

In Fig. 4, we compare the total delay on training of our proposed algorithm with the baselines. We choose $\zeta = 0.6$ for RS-DFL since Fig. 3 shows that RS-DFL can achieve similar accuracy to our proposed algorithm at this sampling ratio. Fig. 4 shows that our proposed algorithm can achieve a lower total delay than the other baselines. When $N$ is equal to 50, our proposed algorithm can achieve a total delay of 23.82% and 13.59% lower than FP-DFL and PF-FL schemes, respectively. This indicates the effectiveness of partial participation. In addition, the total delay of our proposed algorithm is 6.71% lower than the RS-DFL scheme. This is because the neighboring clients are properly selected by the proposed algorithm.

In Fig. 5, we show the impact of parameter $\gamma$ of the neighbor selection module on the total delay on training and learning performance of DFL. The value of $\gamma$ controls the average fraction of selected neighbors (i.e. $\frac{1}{N}\sum_{i=1}^{N}\frac{\|\mathbf{s}_i^\tau\|_1}{\|\mathbf{a}_i^\tau\|_1}$). When $\gamma =$ 0.1, 0.4, and 0.9, the average fraction of selected neighbors is 0.95, 0.67, and 0.12, respectively. Increasing $\gamma$ within a certain range can reduce the delay while maintaining the average testing accuracy. When $\gamma$ varies from 0.1 to 0.4, the average testing accuracy remains stable and the training delay decreases. This is because selecting fewer but proper neighbors for each client leads to a lower delay and a relatively good learning performance of DFL. Further increasing $\gamma$ results in a significant degradation in the average testing accuracy since the limited number of selected neighbors for each client hinders a good approximation of the global model.

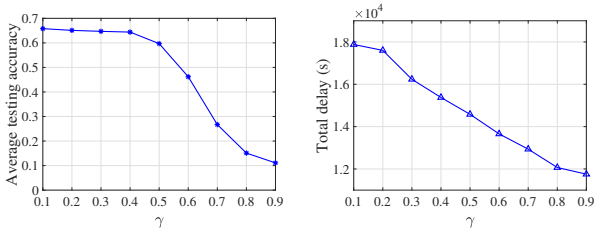


Fig. 5: Effect of $\gamma$ on (a) the average testing accuracy and (b) the total delay on training of the proposed algorithm.

## V. Conclusion

In this paper, we proposed a GNN-based approach to minimize the total delay on training and optimize the learning performance of DFL in D2D networks. We developed an attention-based GNN to determine the neighbor selection and resource allocation decisions. Experimental results showed that when compared with three baseline schemes, our proposed algorithm achieves a relatively good learning performance but with a significantly lower total delay on training. These results showed the importance of neighbor selection in DFL. For future work, we will consider minimizing the delay and energy consumption of DFL in non-IID data settings.

## References


[1] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "BrainTorrent: A peer-to-peer environment for decentralized federated learning," *ArXiv preprint ArXiv 2106.02743*, May 2019.

[2] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of local-update SGD algorithms," *Journal of Machine Learning Research*, vol. 22, no. 213, pp. 1–50, Jan. 2021.

[3] M. Bornstein, T. Rabbani, E. Wang, A. S. Bedi, and F. Huang, "SWIFT: Rapid decentralized federated learning via wait-free model communication," in *Proc. of Int'l Conf. Learn. Representations (ICLR)*, Kigali, Rwanda, May 2023.

[4] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "MATCHA: Speeding up decentralized SGD via matching decomposition sampling," in *Proc. of Indian Control Conf. (ICC)*, Hyderabad, India, Dec. 2019.

[5] M. Lee, G. Yu, and G. Y. Li, "Graph embedding-based wireless link scheduling with few training samples," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 4, pp. 2282–2294, Apr. 2021.

[6] M. Lee, G. Yu, and H. Dai, "Decentralized inference with graph neural networks in wireless communication systems," *IEEE Trans. Mob. Comput.*, vol. 22, no. 5, pp. 2582–2598, May 2023.

[7] C. He, E. Ceyani, K. Balasubramanian, M. Annavaram, and S. Avestimehr, "SpreadGNN: Serverless multi-task federated learning for graph neural networks," in *Proc. of Int'l Conf. on Machine Learn. (ICML)*, Jul. 2021.

[8] S. Wang, M. Lee, S. Hosseinalipour, and C. Brinton, "Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation," in *Proc. of IEEE INFOCOM*, May 2021.

[9] V.-D. Nguyen, S. K. Sharma, T. X. Vu, S. Chatzinotas, and B. Ottersten, "Efficient federated learning algorithm for resource allocation in wireless IoT networks," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3394–3409, Mar. 2021.

[10] L. Wang, Y. Xu, H. Xu, M. Chen, and L. Huang, "Accelerating decentralized federated learning in heterogeneous edge computing," *IEEE Tran. Mob. Comput.*, vol. 22, no. 9, pp. 5001–5016, Sep. 2023.

[11] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and Y. J. Guo, "Multi-timescale decentralized online orchestration of software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2716–2730, Dec. 2018.

[12] A. Agrawal, B. Amos, S. T. Barratt, S. P. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Proc. of Conf. on Neural Info. Process. Syst. (NeurIPS)*, Vancouver, Canada, Dec. 2019.