HOMOGENEOUS LEARNING: SELF-ATTENTION DECENTRALIZED DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning (FL) has been facilitating privacy-preserving deep learning in many walks of life such as medical image classification, network intrusion detection, and so forth. Whereas it necessitates a central parameter server for model aggregation, which brings about delayed model communication and vulnerability to adversarial attacks. A fully decentralized architecture like Swarm Learning allows peer-to-peer communication among distributed nodes, without the central server. One of the most challenging issues in decentralized deep learning is that data owned by each node are usually non-independent and identically distributed (non-IID), causing time-consuming convergence of model training. To this end, we propose a decentralized learning model called Homogeneous Learning (HL) for tackling non-IID data with a self-attention mechanism. In HL, training performs on each round's selected node, and the trained model of a node is sent to the next selected node at the end of each round. Notably, for the selection, the self-attention mechanism leverages reinforcement learning to observe a node's inner state and its surrounding environment's state, and find out which node should be selected to optimize the training. We evaluate our method with various scenarios for two different image classification tasks. The result suggests that HL can produce a better performance compared with standalone learning and greatly reduce both the total training rounds by 50.8% and the communication cost by 74.6% compared with random policy-based decentralized learning for training on non-IID data.

1 INTRODUCTION

Decentralized deep learning (DDL) is a concept to bring together distributed data sources and computing resources while taking the full advantage of deep learning models. Nowadays, DDL such as Federated Learning (FL) (Konečný et al., 2016) has been offering promising solutions to social issues surrounding data privacy, especially in large-scale multi-agent learning. These massively distributed nodes can facilitate diverse use cases, such as industrial IoT (Parimala et al., 2021), environment monitoring with smart sensors (Gao et al., 2020), human behavior recognition with surveillance cameras (Liu et al., 2020), connected autonomous vehicles control (Pokhrel & Choi, 2020; Liu et al., 2019), federated network intrusion detection (Sun et al., 2021; Rahman et al., 2020), and so forth.

Though FL has been attracting great attention due to the privacy-preserving architecture, recent years' upticks in adversarial attacks cause its hardly guaranteed trustworthiness. FL encounters various threats, such as backdoor attacks (McMahan et al., 2018; Cao et al., 2019; Nguyen et al., 2020), information stealing attacks (Duan et al., 2021), and so on. On the contrast, fully decentralized architectures like Swarm Learning (SL) (Warnat-Herresthal et al., 2021) leverages the blockchain, smart contract, and other state-of-the-art decentralization technologies to offer a more practical solution. Whereas, a great challenge of it has been deteriorated performance in model training with non-independent identically distributed (non-IID) data, leading to extremely increased time of model convergence.

Our contributions. We propose a self-attention decentralized deep learning model called Homogeneous Learning (HL). HL leverages a shared communication policy for adaptive model sharing among nodes. A starter node initiates a training task and by iteratively sending the trained model

and performing training on each round's selected node its model is updated for achieving the training goal. Notably, a node selection decision is made by reinforcement learning agents based on the current selected node's inner state and outer state of its surrounding environment to maximize a reward for moving towards the training goal. Finally, comprehensive experiments and evaluation results suggest that HL can accelerate the model training on non-IID data with 50.8% fewer total training rounds and reduce the associated communication cost by 74.6%.

Paper outline. This paper is organized as follows. Section 2 demonstrates the most recent work about DDL and methodologies for tackling data heterogeneity problems in model training. Section 3 presents the technical underpinnings of Homogeneous Learning, including the privacy-preserving decentralized learning architecture and the self-attention mechanism using reinforcement learning agents. Section 4 demonstrates experimental evaluations. Section 5 concludes the paper and gives out future directions of this work.

2 Related Work

Decentralized Deep Learning. In recent years, lots of DDL architectures have been proposed leveraging decentralization technologies such as the blockchain and ad hoc networks. For instance, Li et al. (2021) presented a blockchain-based decentralized learning framework based on the FISCO blockchain system. They applied the architecture to train AlexNet models on the FEMNIST dataset. Similarly, Lu et al. (2020) demonstrated a blockchain empowered secure data sharing architecture for FL in industrial IoT. Furthermore, Mowla et al. (2020) proposed a client group prioritization technique leveraging the Dempster-Shafer theory for unmanned aerial vehicles (UAVs) in flying ad-hoc networks. HL is a fully decentralized machine learning model sharing architecture based on decentralization technology such as token exchanges.

Convergence Optimization. In a real-life application, usually data owned by different clients in such a decentralized system are skewed. For this reason, the model training is slow and even diverges. Methodologies for tackling such data heterogeneity such as FL, have been studied for a long time. For example, Sener & Savarese (2018) presented the K-Center clustering algorithm which aims to find a representative subset of data from a very large collection such that the performance of the model based on the small subset and that based on the whole collection will be as close as possible. Moreover, Wang et al. (2020) demonstrated reinforcement learning-based client selection in FL, which counterbalances the bias introduced by non-IID data thus speeding up the global model's convergence. Sun et al. (2021) proposed the Segmented-FL to tackle heterogeneity in massively distributed network intrusion traffic data, where clients with highly skewed training data are dynamically divided into different groups for model aggregation respectively at each round. Furthermore, Zhao et al. (2018) presented a data-sharing strategy in FL by creating a small data subset globally shared between all the clients. Likewise, Jeong et al. (2018) proposed the federated augmentation where each client augments its local training data using a generative neural network. Different from the aforementioned approaches, HL leverages a self-attention mechanism that optimizes the communication policy in DDL using reinforcement learning models. It is aimed to reduces computational and communication cost of decentralized training on skewed data.

3 Homogeneous Learning

3.1 Preliminary

Data Privacy and Decentralized Deep Learning. Centralized deep learning in high performance computing (HPC) environments has been facilitating the advancement in various areas such as drug discovery, disease diagnosis, cybersecurity, and so on. Despite its broad applications in many walks of life, the associated potential data exposure of training sources and privacy regulation violation have greatly decreased the practicality of such centralized learning architecture. In particular, with the promotion of GDPR (EU), data collection for centralized model training has become more and more difficult. For this reason, Google proposed federated learning (FL) to alleviate the limitation of model training on distributed data. FL allows a client to train its own model based on a local dataset and achieve a better performance by sharing the training result with others, whereas without sharing the raw training data. FL quickly acquired intense attention from lots of fields related to sensitive

data processing including medical image classification, face recognition, intrusion detection, finance data analysis, and so forth. Moreover, a fully decentralized deep learning architecture is peer-to-peer networking of nodes based on decentralization and security technologies such as the token-exchange, a service capable of validating and issuing security tokens to enable nodes to obtain appropriate access credentials for exchanging resources without the central server. In this case, each node owns a local training model and performs both the function of the client and the server based on a shared communication policy, which is different from FL where the central server plays the key role in model sharing.

We specifically consider a supervised learning task with C categories in the entire dataset D. Suppose that $f_\theta : x \to y$ denotes a neural network classifier with parameters θ, taking an input $x_i \in x$ and outputting a C-dimensional real-valued vector also known as the logit. This neural network gives a predicted label $y_i = arg\,max f_\theta(x_i)$ s.t. $y_i \in y$. We assume there are K nodes in the network. The kth node has its own dataset $D_{local}^{(k)} := \{(x_i, y_i)\}_{i=1}^{N^{(k)}}$, where x_i is the ith training sample, y_i is the corresponding label of x_i, $y_i \in \{1, 2, ..., C\}$ (a multi-classification learning task), and $N^{(k)}$ is the sample number in dataset $D_{local}^{(k)}$. $D := \{D_{local}^1, D_{local}^2, ..., D_{local}^i\}, N = \sum_{k=1}^{K} N^{(k)}$. The goal of the decentralized systems is to achieve a desired performance on the entire data through sharing local trained models $L_t^{(k)}$ at each round *t*.

Data Heterogeneity. The challenges related to heterogeneity of nodes in DDL refer to two categories, i.e., data heterogeneity and hardware heterogeneity. Notably, data heterogeneity results in time-consuming convergence or divergence of model learning. Let $p(x|y)$ be the common data distribution of the entire data *D*. We assume the common distribution $p(x|y)$ is shared by all nodes. Then, Node$k$ has $p_k(y)$. We first consider an independent and identically distributed (IID) setting, i.e., $p_i(x, y) = p(x|y)p_i(y)$ s.t. $p_i(y) = p_j(y)$ for all $i \neq j$. Under this assumption, the data distribution of the entire dataset can be represented by a node's local data distribution. Unfortunately, in real-life application, samples held by clients are usually skewed with various data distributions, i.e., $p_i(x, y) = p(x|y)p_i(y)$ s.t. $p_i(y) \neq p_j(y)$ for all $i \neq j$. Node1 follows $p1(x, y)$ and Node2 follows $p2(x, y)$. We further define and clarify such data heterogeneity as follows: for a node*k*'s local dataset, when its α samples are from a single main data class $c^{(k)}$ subject to $\alpha > \frac{N^{(k)}}{C}$ and the remaining samples are randomly drawn from the other *C-1* data classes, the heterogeneity level $H^{(k)}$of node*k* is formulated as $H^{(k)}(D_{local}^{(k)}) = -p(y = c) * log(p(y \neq c))$. Moreover, we assign a main data class $c^{(k)} = k\%C$ to node*k*.

Communication Overhead. Though communication overhead in a decentralized learning system also involves the payload size such as different numbers of model parameters (He et al., 2020; Singh et al., 2019), it is out of the scope of this research where we focus on different communication distances between distributed nodes. In particular, for every two nodes *i* and *j*, a relative communication distance $d_{i,j}$ is defined in the symmetrical matrix $Dis_{i\times j}$, where the bidirectional distances between two nodes are equal and the distance to a node itself $d_{i,j|i=j}$ is zero. Furthermore, each distance $d_{i,j|i\neq j}$ in the matrix is a random numerical value between 0 and β (Equation 1).

$$Dis_{i\times j} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,j} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ d_{i,1} & d_{i,2} & \cdots & d_{i,j} \end{pmatrix} \quad subject\,to: \; d_{i,j|i=j} = 0,\, d_{i,j} = d_{j,i}, d_{i,j|i\neq j} \in (0, \beta] \quad (1)$$

Where $d_{i,j}$ represents the relative distance from node *i* to node *j*.

3.2 Technical Fundamentals of Homogeneous Learning

We propose a novel decentralized deep learning architecture called Homogeneous Learning (HL) (Fig. 1). HL leverages reinforcement learning (RL) agents to learn a shared communication policy of node selection, thus contributing to fast convergence of model training and reducing communication cost as well. In HL, each node has two machine learning (ML) models, namely a local ML task model $L^{(k)}$ for the multi-classification task and an RL model L^{DQN} for the node selection in peer-to-peer communications.

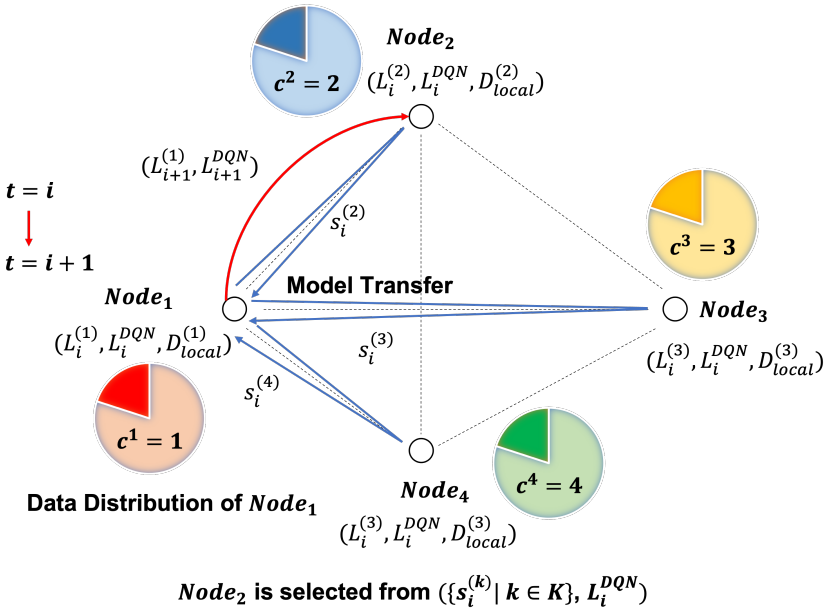

Figure 1: Homogeneous learning: self-attention decentralized deep learning.

3.2.1 Local ML Task Model

Task Model Architecture. We assume the K nodes in HL share the same model architecture as a local ML task model. Let y_i be the layer$i$'s output of $L^{(k)}$. $y_i = f_i(W_i y_{i-1}), i = 1, ..., p, y_0 = x$, where f_i is the activation function, W_i is the weight matrix of layer$i$, y_{i-1} represents the output of the previous layer, and p is the number of layers in $L^{(k)}$. Notably, we employ a three-layer convolutional neural network (CNN) with an architecture as follows: the first convolutional layer of the CNN model has a convolution kernel of size 5×5 with a stride of 1 and it takes one input plane and it produces 20 output planes, followed by a ReLU activation function; the second convolutional layer takes 20 input planes and produces 50 output planes and it has a convolution kernel of size 5×5 with a stride of 1, followed by ReLU; the output is flattened followed by a linear transformation of a fully connected layer, which takes as input the tensor and outputs a tensor of size C representing the C categories. Moreover, the categorical cross-entropy is employed to compute a loss ℓ. After that, we apply as a learning function the Adam to update the model, i.e., $L_{t+1}^{(k)} \leftarrow L_t^{(k)} - \eta \cdot \nabla \ell(L_t^{(k)}, D_{local}^{(k)})$, where $L_t^{(k)}$is node$k$'s local ML task model at the round t and η is the learning rate.

3.2.2 Reinforcement Learning Model

Besides the local ML task model, each node$k$ in HL is also associated with a reinforcement learning (RL) model L^{DQN}. The goal of the RL model is to learn a communication policy for the node selection in decentralized learning. There are three main components of the RL model, namely, the state s, the action a, and the reward r. Then, based on the input state s, the RL model outputs an action a for the next node selection, and at the same time, updates itself by correlating the attained reward r with the performed action a. As a result, the recursive self-improvement of the RL model allows a node to constantly explore the relation between the system's performance and the selection policy (the self-attention mechanism in HL), contributing to faster convergence of model learning.

Every round t, a RL model observes the state s_t from two different sources, i.e., model parameters $s_t^{(k)}$ of the selected node$k$ and parameters of models in the surrounding environment $\{s_t^{(i)} | i \in K, i \neq k\}$. In particular, we employ a deep Q-network (DQN), which approximates a state-value function in a Q-learning framework with a neural network. Let y_i^{DQN} be the layer$i$'s output of L^{DQN}. $y_i^{DQN} = f_i^{DQN}(W_i^{DQN} y_{i-1}^{DQN}), i = 1, ..., q, y_0^{DQN} = s$, where f_i^{DQN} is the activation function of layer$i$, W_i^{DQN} is the weight matrix of layer$i$, y_{i-1}^{DQN}represents the output of the previous layer,

and q is the number of layers in L^{DQN}. Notably, a DQN model consisting of three fully connected layers is applied (Fig. 2). The two hidden layers consist of 500 and 200 neurons respectively, using as an activation function the ReLU. The output layer with a linear activation function consists of K neurons that output the rewards of selecting each node*k* respectively, $k \in \{1, 2, ..., K\}$. Furthermore, at each round *t*, the node with the largest reward will be selected. $a_t = arg\,max f_{L^{DQN}}(s_t)$, where L^{DQN} denotes weights of the DQN model. Consequently, the RL model selects and sends the trained local model $L_{t+1}^{(k)}$ of node*k* to the next node a_t. As such, the local ML task model $L_{t+1}^{(a_t)}$ of node a_t is updated to $L_{t+1}^{(k)}$.

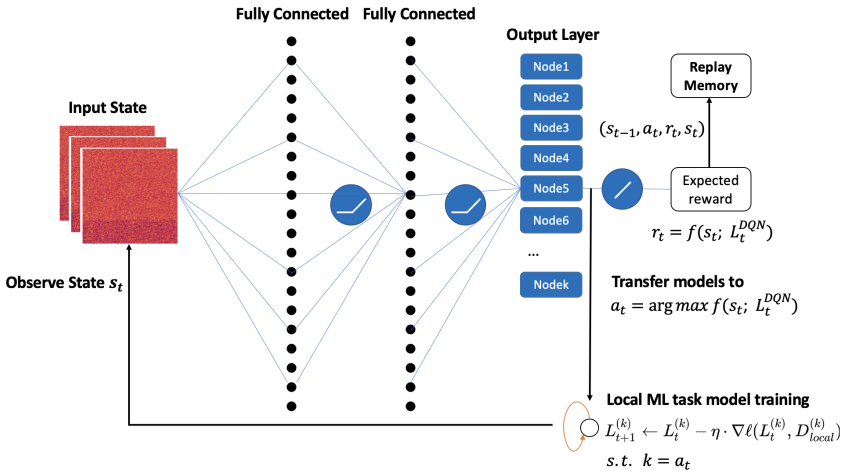

Figure 2: Next-node selection based on the RL model of HL.

To understand the training of the RL model, we first define the input state s_t. The state s_t is a concatenated vector of the flattened model parameters of all nodes in the systems. $s_t = \{s_t^{(k)} | k \in K\}$. To efficiently represent the state and compute the RL model prediction, we adopt the principal component analysis (PCA) to reduce the dimension of the state s_t from an extremely large number (e.g. 33580 dimensions for MNIST) to *K*, where *K* is the number of nodes. *K* is adopted due to the minimum possible dimension of a PCA-based output vector is the number of input samples. Then, we define the output reward r_t. Every round t, a trained ML task model is evaluated on a hold-out validation set D_{val}, and the reward r_t can be computed from the validation accuracy $ValAcc_t$, the communication distance between the current node k and the next selected node a_t, and a penalty of minus one for taking each training step. $r_t = 32^{ValAcc_t - GoalAcc} - d_{k,a_t} - 1$, where $GoalAcc$ denotes the desired performance on the validation set and d_{k,a_t} is the communication distance drawn from the distance matrix $Dis_{i \times j}$. We employ an exponentially increasing function $32^{(\cdot)}$ to distinguish between different validation results when the ML task model is close to convergence when only small variance is observed in the results. In addition, an episode reward R is the accumulative reward of the current reward and discounted future rewards in the whole training process of HL. $R = \sum_{t=1}^{T} \gamma^{t-1} r_t$, where *T* is the total training rounds of HL in one episode.

With DQN, we often use experience replay during training. A RL model's experience at each time step *t* is stored in a data set called the replay memory. Let e_t be the model's experience at time *t*. $e_t = (s_{t-1}, a_t, r_t, s_t)$, where r_{t+1} is the reward given the previous state-action pair (s_{t-1}, a_t) and s_t is the state of the ML task models after training. We assume a finite size limit M of the replay memory, and it will only store the last *M* experiences. Moreover, to facilitate constant exploration of a RL model, epsilon is a factor to control the probability of the next node being selected by the RL model. In particular, for each round, a random numerical value between 0 and 1 is obtained and compared with the current epsilon value $Epsilon_{ep}$ where *ep* denotes the current episode. Then if the randomly picked value is greater than $Epsilon_{ep}$, the next node will be selected by the RL model. Otherwise, a random action of node selection will be performed. For either case, an experience sample $e_t = (s_{t-1}, a_t, r_t, s_t)$ will be stored in the replay memory. The decentralized learning terminates when either the model achieves the desired performance on the validation set or exceeds a maximum number of rounds T_{max}, the learning progress of which is called an episode of HL. For

each episode, we apply the epsilon decay ρ to gradually increase the possibility of the RL model's decision-making.$Epsilon_{ep+1} = Epsilon_{ep} \cdot e^{-\rho}$, where $Epsilon_{ep+1}$ is the computed epsilon for the next episode and e is the Euler's number that is approximately equal to 2.718.

Furthermore, at the end of each episode *ep*, the RL model is trained on a small subset of samples randomly drawn from the replay memory. Let ℓ be the mean squared error loss of L_t^{DQN}. We adopt as a learning function the Adam. Then, the optimization of the DQN model is formulated in (5). The updated DQN model is shared with the next selected node. As such, the RL model performs better and better in predicting the expected rewards of selecting each node for the next round, which results in the increase of the episode reward *R* by selecting the node with the largest expected reward at each round *t*.

$$\ell(L_t^{DQN}) = \sum_{i=1}^{B} \ell(r_t + \beta \max_{a_{i+1}} f(a_{i+1}, s_i; L_t^{DQN}), f(a_i, s_{i-1}; L_t^{DQN})) \tag{2}$$

$$\theta^* = arg\min_{\theta} \ell(\theta),\ subject\,to\ \theta = L_t^{DQN}$$

Where a_{i+1} denotes the predicted next step's action that maximizes the future reward, β denotes the discount factor of the future reward, f denotes a feed-forward function that produces the output of the RL model with respect to a_i based on s_i, and *B* denotes the batch size.

Finally, the model training of HL is formulated as Algorithm 1.

Algorithm 1 Model Training of Homogeneous Learning

Training:
initialize L_0^{dqn}
for each episode $ep = 1, 2, ...$ **do**
    initialize $L_0^{(k)}$
    $a_0 = k$
    for each step $t = 0, 1, 2, ...$ **do**
        while $ValAcc_t < GoalAcc$ and $t < T_{max}$ **do** ▷ T_{max} is the maximum steps per episode
            $ValAcc_{t+1}, a_t, L_{t+1}^{(k)} = HL(L_t^{(k)}, L_t^{DQN})$
            Send $\{L_{t+1}^{(k)}, L_{t+1}^{DQN}\}$ to a_t for the next step's model update
        end while
    end for
    $L_{ep+1}^{DQN} = arg \min_{L_{ep}^{DQN}} \sum_{i=1}^{B} \ell(r_t + \beta \max_{a_{i+1}} f(a_{i+1}, s_{i+1}; L_{ep}^{DQN}), f(a_i, s_i; L_{ep}^{DQN}))$
    $Epsilon_{ep+1} = Epsilon_{ep} \cdot e^{-\rho}$
end for

function HL($L_t^{(k)}, L_t^{DQN}$) $L_{t+1}^{(k)} = Train(L_t^{(k)}, D_{local}^{(k)})$ ▷ $D_{local}^{(k)}$ is Node$k$'s local training data
$ValAcc_{t+1} = Acc(D_{val}; L_{t+1}^{(k)})$
$s_t^{(k)} = L_{t+1}^{(k)}$
$s_t^{(i)} = L_t^{(i)}$ subject to $i \in K, i \neq k$
$s_t = \{s_t^{(k)}, s_t^{(i)} | \, i \in K,\ i \neq k\}$
$a_t = arg\,max f(s_t; L_t^{DQN})$
$r_t = 32^{ValAcc_t - GoalAcc} - d_{k,a_t} - 1$
Add $\{s_{t-1}, a_t, r_t, s_t\}$ to the replay memory
return $ValAcc_{t+1}, a_t, L_{t+1}^{(k)}$
end function

4 EVALUATION

4.1 EXPERIMENT SETUP

4.1.1 DATASET.

We applied MNIST (LeCun et al., 2010), a handwritten digit image dataset containing 50,000 training samples and 10,000 test samples labeled as 0-9, and Fashion-MNIST(Xiao et al., 2017), an image collection of 10 types of clothing containing 50,000 training samples and 10,000 test samples labeled as shoes, t-shirts, dresses, and so on. The size of grayscale images in these two datasets is 28×28. Then, we considered both a 10-node scenario and a 100-node scenario training on the MNIST dataset and the Fashion-MNIST dataset respectively. The machine learning library we used to build the system is Tensorflow.

4.1.2 BASELINE MODELS.

To compare the performance of the proposed approach, we considered three different model training methods as baselines, which are centralized learning with all data collected from all nodes, decentralized learning with a random node selection policy, and standalone learning of the starter node without model sharing. In detail, for each method, we applied the same local ML task model architecture and associated model training hyperparameters using the training set data from MNIST and Fashion-MNIST respectively. The goal of model training is to achieve a validation accuracy of 0.80 for the MNIST classification task and 0.70 for the Fashion-MNIST classification task based on the hold-out test set of the corresponding dataset.

Regarding the standalone method, we utilized the early stopping to monitor the validation loss of the model at each epoch with a patience of five, which automatically terminated the training process when there appeared no further decrease in the validation loss of the model for the last five epochs. In the centralized and standalone learning, evaluation was performed at each epoch of the training. Moreover, in decentralized learning, due to multiple models in a system, the evaluation was performed on the trained local model of each step's selected node.

4.1.3 HOMOGENEOUS LEARNING SETTINGS.

Homogeneous Learning (HL) of $K = \{10, 100\}$ nodes with a heterogeneity level $H = \{0.24, 0.56, 0.90\}$ ($p(y = c) = \{0.6, 0.8, 0.9\}$) was adopted. Each node *k* owned a total of 500 skewed local training data with a heterogeneity level of *H*. In addition, to generate the distance matrix, the relative communication cost represented by the distance between two different nodes $d_{i,j|i\neq j}$ takes a random numerical value between 0 and 0.1. A random seed of 0 was adopted for the reproducibility of the distance matrix (See A.1). For the local ML task model training, we adopted an epoch of one with a batch size of 32. A further discussion on the selection of these two hyperparameters can be found in the section A.2. The Adam was applied as an optimization function with a learning rate of 0.001.

4.2 NUMERICAL RESULTS

4.2.1 LEARNING A COMMUNICATION POLICY BASED ON DEEP Q-NETWORKS

As aforementioned, each node has a specific main data class c. We considered a starter node with a main data class of digit '0' in the case of MNIST and a class of T-shirt in the case of Fashion-MNIST. Then, starting from the starter node, a local ML task model was trained on the current node's local data and sent to the next step's node decided by either the RL model or a random action depending on the epsilon at the current episode. We adopted an initial epsilon of one and a decay rate of 0.02. Moreover, the RL model was updated at the end of each episode using the hyperparameters defined in Table 1. We applied a maximum replay memory size of 50,000 and a minimum size of 128, where the training of the DQN model started only when there were more than 128 samples in the replay memory and the oldest samples would be removed when samples were more than the maximum capacity. Furthermore, in every episode, an agent randomly drew 32 samples from the memory to update its model, with a total of 120 episodes. The maximum training step is 35 in the case of MNIST and 100 in the case of Fashion-MNIST.

Table 1: Hyperparameters in Homogeneous Learning

ML TASK MODEL		RL MODEL	
Epoch	1	Episode	120
Batch size	32	Future reward discount	0.9
Learning rate	0.001	Epsilon decay	0.02
Optimization function	Adam	Epoch	1
Maximum step	35 (MNIST)/100 (Fashion-MNIST)	Batch size	16
		Learning rate	0.001

For each episode, we computed the step rewards and the episode reward for the model training to achieve the performance goal. With the advancement of episodes, the communication policy evolved to improve the episode reward thus benefiting better decision-making of the next-node selection. Figure 3.a illustrates the episode reward and the mean reward over the last 10 episodes during HL and the corresponding total training rounds for each method when training on MNIST. Figure 3.b illustrates the episode reward results of HL when training on Fashion-MNIST.

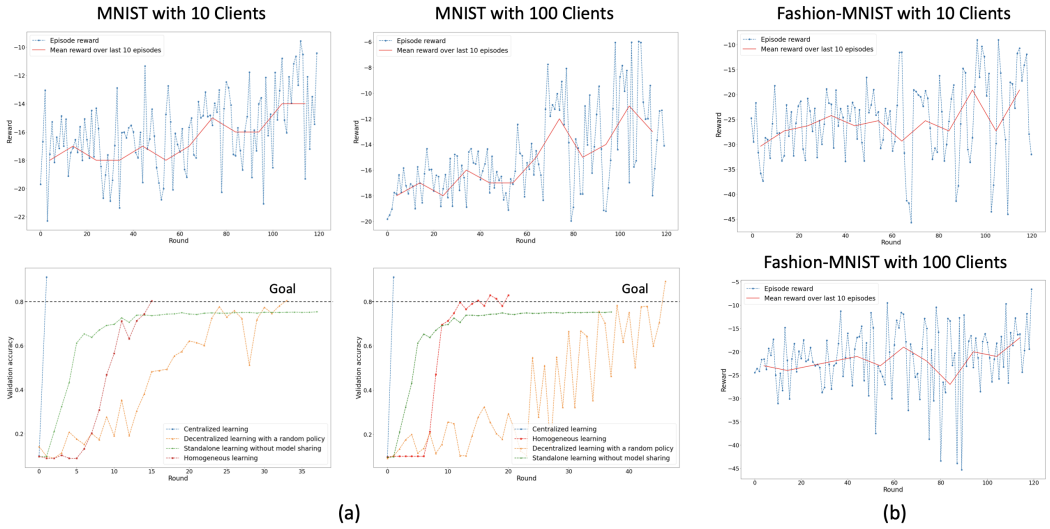

Figure 3: (a) With the increase of episodes, the mean reward over last 10 episodes is gradually increasing. The DQN model learned a better communication policy by training on samples from the replay memory, contributing to the systems' performance in total training rounds. (b) Episode reward results for the 10-node and 100-node scenarios when applying the Fashion-MNIST dataset. Compared with MNIST, the policy learning was more unstable with the same hyperparameter setting, however, it showed an increasing episode reward.

4.2.2 Comparisons Regarding Computational and Communication Cost

To compare the performance between HL and the aforementioned three baseline models, we performed a comprehensive evaluation against the metrics of computational cost and communication cost in the case of MNIST with the 10-node scenario. For each method, we performed 10 individual experiments with different random seeds. Here, the computational cost refers to the required total rounds for a system to achieve the training goal, and the communication cost refers to the total communication distance for the model sharing in decentralized learning.

Computational Cost. As shown in Figure 3.a, due to limited local training data, the standalone learning appeared to be extremely slow after the validation accuracy reached 0.70. Finally, it terminated with a final accuracy of around 0.75 due to the early-stopping strategy. Moreover, by comparing the decentralized learning methods with and without the self-attention mechanism, the result

suggests that our proposed method of HL can greatly reduce the total training rounds. In addition, though centralized learning shows the fastest convergence, it suffers from problems of data privacy.

Communication Cost. In decentralized learning, to train a model, each selected node trains the current ML task model on its local dataset and sends the trained model to another node, and the communication cost refers to the network traffic payload for sending the model. We studied the relative cost by introducing the communication distance between nodes, which is defined as the total communication distance for model sharing in HL, i.e., from the starter node to the last selected node.

We performed ten individual experiments for each method and used as final results the best cases of node selection over the last five episodes when decisions were almost made by the agent and a learned communication policy was prone to be stable. Figure 4.a illustrates the experiment results of the total training rounds and the communication cost. The bottom and top of the error bars represent the 25_{th} and 75_{th} percentiles respectively, the line inside the box shows the median value, and outliers are shown as open circles. Finally, the evaluation result shows that HL can greatly reduce the training rounds by 50.8% and the communication cost by 74.6%.

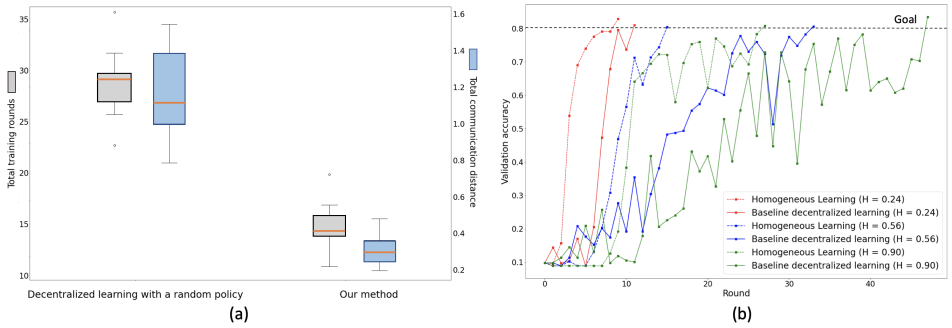

Figure 4: (a) Performance comparison between the random policy-based decentralized learning and HL. Each error bar illustrates 10 individual experiments' results. (b) Computational performance comparison with various heterogeneity levels of training data.

4.2.3 Comparisons Regarding Various Heterogeneity Levels

We further studied the performance of the proposed method with different heterogeneity levels $H = \{0.24, 0.56, 0.90\}$ ($p(y = c) = \{0.6, 0.8, 0.9\}$). We evaluated the performance in the 10-node scenario training on MNIST. In addition, for the case of $H = 0.90$, we applied a maximum training step of 80 instead due to a more challenging convergence of the ML task model using the highly skewed local training data. Figure 4.b illustrates a computational performance comparison between the proposed HL and the baseline decentralized learning with a classical random policy.

5 Conclusion

Decentralized deep learning (DDL) leveraging distributed data sources contributes to a better neural network model while safeguarding data privacy. Despite the broad applications of DDL models such as federated learning and swarming learning, the challenges regarding edge heterogeneity especially the data heterogeneity have greatly limited their scalability. In this research, we proposed a self-attention decentralized deep learning method of Homogeneous Learning (HL) that recursively updates a shared communication policy by observing the system's state and the gained reward for taking an action based on the observation. We comprehensively evaluated the proposed method by comparing with three baseline models for two different image classification tasks in both a 10-node scenario and a 100-node scenario, applying as criteria the computational and communication cost. The evaluation result shows that HL can greatly reduce the cost when training on skewed decentralized data with various heterogeneity levels. In future, a decentralized model leveraging various communication policies at the same time to achieve diverse goals is considered for the further study of this research.

REFERENCES

General data protection regulation. https://gdpr-info.eu. Accessed: 2021-09-22.

Di Cao, Shan Chang, Zhijian Lin, Guohua Liu, and Donghong Sun. Understanding distributed poisoning attack in federated learning. In *25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019*, pp. 233–239. IEEE, 2019. doi: 10.1109/ICPADS47876.2019.00042. URL https://doi.org/10.1109/ICPADS47876.2019.00042.

Moming Duan, Duo Liu, Xianzhang Chen, Renping Liu, Yujuan Tan, and Liang Liang. Self-balancing federated learning with global imbalanced data in mobile systems. *IEEE Trans. Parallel Distributed Syst.*, 32(1):59–71, 2021. doi: 10.1109/TPDS.2020.3009406. URL https://doi.org/10.1109/TPDS.2020.3009406.

Yujia Gao, Liang Liu, Binxuan Hu, Tianzi Lei, and Huadong Ma. Federated region-learning for environment sensing in edge computing system. *IEEE Transactions on Network Science and Engineering*, 7(4):2192–2204, 2020. doi: 10.1109/TNSE.2020.3016035.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/a1d4c20b182ad7137ab3606f0e3fc8a4-Abstract.html.

Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *CoRR*, abs/1811.11479, 2018. URL http://arxiv.org/abs/1811.11479.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Netw.*, 35(1): 234–241, 2021. doi: 10.1109/MNET.011.2000263. URL https://doi.org/10.1109/MNET.011.2000263.

Boyi Liu, Lujia Wang, Ming Liu, and Chengzhong Xu. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *CoRR*, abs/1901.06455, 2019. URL http://arxiv.org/abs/1901.06455.

Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(08): 13172–13179, Apr. 2020. doi: 10.1609/aaai.v34i08.7021. URL https://ojs.aaai.org/index.php/AAAI/article/view/7021.

Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. Blockchain and federated learning for privacy-preserved data sharing in industrial iot. *IEEE Transactions on Industrial Informatics*, 16(6):4177–4186, 2020. doi: 10.1109/TII.2019.2942190.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJ0hF1Z0b.

Nishat I. Mowla, Nguyen H. Tran, Inshil Doh, and Kijoon Chae. Federated learning-based cognitive detection of jamming attack in flying ad-hoc network. *IEEE Access*, 8:4338–4350, 2020. doi: 10.1109/ACCESS.2019.2962873.

T. Nguyen, P. Rieger, Markus Miettinen, and A. Sadeghi. Poisoning attacks on federated learning-based iot intrusion detection system. 2020.

Mani Parimala, R. M. Swarna Priya, Quoc-Viet Pham, Kapal Dev, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Thien Huynh-The. Fusion of federated learning and industrial internet of things: A survey. *ArXiv*, abs/2101.00798, 2021.

Shiva Raj Pokhrel and Jinho Choi. Federated learning with blockchain for autonomous vehicles: Analysis and design challenges. *IEEE Transactions on Communications*, 68(8):4734–4746, 2020. doi: 10.1109/TCOMM.2020.2990686.

Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Network*, 34(6):310–317, 2020. doi: 10.1109/MNET.011.2000286.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1aIuk-RW.

Abhishek Singh, Praneeth Vepakomma, Otkrist Gupta, and Ramesh Raskar. Detailed comparison of communication efficiency of split learning and federated learning. *CoRR*, abs/1909.09145, 2019. URL http://arxiv.org/abs/1909.09145.

Yuwei Sun, Hiroshi Esaki, and Hideya Ochiai. Adaptive intrusion detection in the networking of large-scale lans with segmented federated learning. *IEEE Open J. Commun. Soc.*, 2:102–112, 2021. doi: 10.1109/OJCOMS.2020.3044323. URL https://doi.org/10.1109/OJCOMS.2020.3044323.

Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pp. 1698–1707. IEEE, 2020. doi: 10.1109/INFOCOM41043.2020.9155494. URL https://doi.org/10.1109/INFOCOM41043.2020.9155494.

Stefanie Warnat-Herresthal, Hartmut Schultze, Krishnaprasad Lingadahalli Shastry, Sathyanarayanan Manamohan, Saikat Mukherjee, Vishesh Garg, Ravi Sarveswara, Kristian Händler, Peter Pickkers, N. Ahmad Aziz, Sofia Ktena, Christian Siever, Michael Kraut, Milind Desai, Bruno Monnet, Maria Saridaki, Charles Martin Siegel, Anna Drews, Melanie Nuesch-Germano, Heidi Theis, Mihai G. Netea, Fabian Theis, Anna C. Aschenbrenner, Thomas Ulas, Monique M.B. Breteler, Evangelos J. Giamarellos-Bourboulis, Matthijs Kox, Matthias Becker, Sorin Cheran, Michael S. Woodacre, Eng Lim Goh, Joachim L. Schultze, and German COVID-19 OMICS Initiative (DeCOI). Swarm learning for decentralized and confidential clinical machine learning, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018. URL http://arxiv.org/abs/1806.00582.

A Appendix

A.1 Communication Distance Matrix

Figure 5 illustrates the generated distance matrix $D_{i \times j}$ in the 10-node scenario when applying a β value of 0.1 and a random seed of 0.

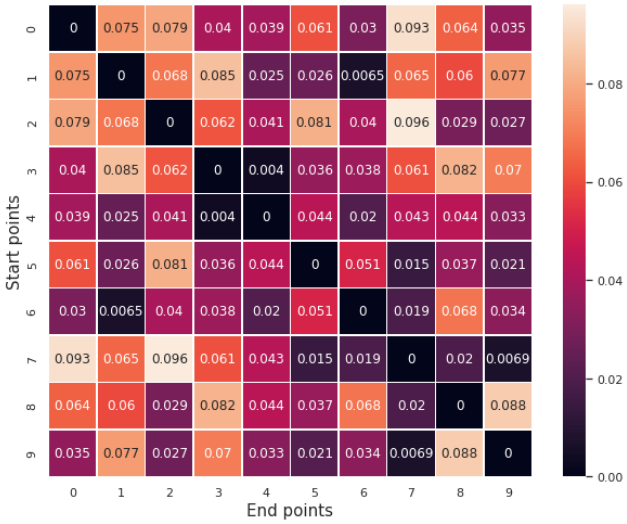

Figure 5: The adopted distance matrix $D_{i\times j}$ in the 10-node scenario.

A.2 Optimization of Model Distribution Representation

Under the assumption of data heterogeneity, to allow a reinforcement learning (RL) agent to efficiently learn a communication policy by observing model states in the systems, a trade-off between the batch size and the epoch of local foundation model training was discussed. Figure 6 illustrates the trained models' weights distribution in the 10-node scenario after applying the principal component analysis (PCA), with different batch sizes and epochs applied to train on the MNIST dataset. Moreover, it shows a 100-node scenario where each color represents nodes with the same main data class. As shown in the graphs, various combinations of these two parameters have different distribution representation capabilities. By comparing the distribution density and scale, we found that when adopting a batch size of 32 and an epoch of one the models distribution was best represented, which could facilitate the policy learning of an agent.

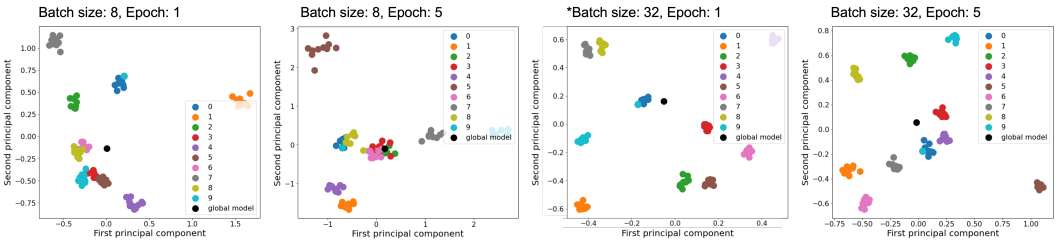

Figure 6: Optimization of model distribution representation in the 10-node scenario.