Predictive GAN-Powered Multi-Objective Optimization for Hybrid Federated Split Learning

Benshun Yin, Zhiyong Chen, Senior Member, IEEE, and Meixia Tao, Fellow, IEEE

Abstract—As an edge intelligence algorithm for multi-device collaborative training, federated learning (FL) can protect data privacy but increase the computing load of wireless devices. In contrast, split learning (SL) can reduce the computing load of devices by model splitting and assignment. To take advantage of FL and SL, we propose a hybrid federated split learning (HFSL) framework for wireless networks in this paper, which combines the multi-worker collaborative training of FL and the flexible splitting of SL. To reduce the computational indelness in model splitting, we design a parallel computing scheme for model splitting without label sharing and conduct a theoretical analysis of the impact of the delayed gradient on the convergence. Aiming to obtain the trade-off between the training time and energy consumption, we model the joint optimization problem of splitting decisions, the bandwidth, and computing resources as a multi-objective problem. As such, we propose a predictive generative adversarial network (GAN)-powered multi-objective optimization algorithm to obtain the Pareto front of the problem, which utilizes the discriminator to guide the training of the generator to predict promising solutions. Experimental results demonstrate that the proposed algorithm outperforms the considered baselines in finding Pareto optimal solutions, and the solutions obtained from the proposed HFSL framework can dominate the solution of FL.

Index Terms—Federated learning, split learning, parallel computing, generative adversarial network, multi-objective optimization.

I. INTRODUCTION

With the rapid growth of Internet of things (IoT), a large amount of data is generated by IoT devices every day [1]. To take advantage of the distributed data, edge machine learning algorithms are being developed to realize intelligent applications in wireless networks [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Federated learning (FL) [12] is proposed as a method to collaboratively train a machine learning algorithm with the local data of many wireless devices. Compared with traditional centralized learning that transmits large amounts of raw data to the cloud server for training, FL can effectively protect data privacy without exchanging the local data. However, the IoT devices, also called workers, need to perform the local update of the training model with their computing power in FL. It can greatly increase the computation burden of the workers, especially for training deep neural networks with high computational complexity. When workers have low computing power, the FL training time can be significantly prolonged, which can impede the practical application of FL. In addition, local updating entirely by the own computing power of workers increases their energy consumption.

Split learning (SL), also known as model splitting, can split a deep neural network into multiple parts and deliver them to different computational nodes for processing [13], [14], [15], [16], such as workers and edge servers. It can significantly reduce the computing burden and energy consumption of a single node. SL can jointly update the parameters of the training model with the computing resources of many workers by splitting the model and transmitting the intermediate results (output feature and gradient) of the model across workers. In SL, the parameters locally updated by a worker need to be aggregated with the parameters of other workers to obtain global information. Therefore, training frameworks combining FL and SL are proposed [16], [17], [18], [19], [20], [21] to integrate the multi-worker collaborative training of FL and the low computational load on workers of SL.

In the learning system that combines FL and SL, the splitting decisions and resource allocation result in many trade-offs between the training time and energy consumption of workers required to achieve the desired performance. The hybrid federated split learning (HFSL) framework considered in this paper is shown in Fig. 1, which combines SL without label sharing [14]. In SL without label sharing, the model is split into three parts (part a, part b and part c) to protect label privacy, as shown by worker k in Fig. 1. Specifically, part b is offloaded to the server for processing, while the other two parts are executed on the worker. In SL with label sharing, the model is split into two parts, where the worker side executes the first part, and the server executes the other one.

Manuscript received 30 August 2022; revised 13 January 2023, 23 March 2023, and 8 May 2023; accepted 10 May 2023. Date of publication 19 May 2023; date of current version 16 August 2023. This work was supported by the National Natural Science Foundation of China under Grant 62222111 and 62125108, and by the Fundamental Research Funds for the Central Universities. The associate editor coordinating the review of this article and approving it for publication was N. Shlezinger. (Corresponding authors: Zhiyong Chen; Meixia Tao.)

Benshun Yin and Zhiyong Chen are with the Cooperative Medianet Innovation Center and the Shanghai Key Laboratory of Digital Media Processing and Transmission, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yinbsh@sjtu.edu.cn; zhiyongchen@sjtu.edu.cn).

Meixia Tao is with the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: mtxiao@sjtu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2023.3277878.

Digital Object Identifier 10.1109/TCOMM.2023.3277878
Because the output size and computational load of different layers in a neural network can vary, different splitting decisions (whether to split the model and which two layers to split the model in) can affect the computational load offloaded to the server as well as the communication load of transmitting intermediate results between the server and the worker. If the local execution time of part b exceeds the combined time of transmitting intermediate results and executing part b by the server, offloading the execution of part b to the server can reduce the time consumption of parameter updating. If the energy consumption for a worker to execute part b locally exceeds that of uploading the intermediate results to the server, utilizing model splitting and offloading part b to the server can save the energy of the worker. The specific values of these time and energy consumption are determined not only by system parameters including the training model size (output size and computational load of each layer of the neural network) and wireless channel status, but also by the splitting decisions, allocated bandwidth and server computing resources for workers. Therefore, it is necessary to adaptively optimize the splitting decisions and resource allocation based on the training model (e.g., MobileNetV3 [22]) and other system parameters using the multi-objective algorithm to achieve a balance between the time and energy consumption.

A. Related Work

For utilizing the multi-worker parallel update of FL and the low computational requirement for workers of SL, the combination of FL and SL has been considered in many works [16], [17], [18], [19], [20], [21]. In [16], all the workers split the neural network into the worker-side and the server-side part. The server updates the parameters of the server-side part in parallel after receiving the intermediate output of the worker-side part from the workers, while the parameters of the worker-side part need to be transmitted to the server for global averaging. For the learning system with large-scale workers, the workers are divided into multiple groups in [17]. To reduce the frequent communication between the workers and the server, an auxiliary light-weight neural network is applied to update the parameters of the worker-side part without the participation of the server-side part [18], [19]. FL and SL are combined to train a large natural language processing model in [20], where the model can be split and distributed to multiple workers and then updated one by one for sequentially partitioned data. The combination of FL and SL is extended to multi-task learning in [21] with the shared server-side part and task-specific worker-side part. However, existing works [16], [17], [18], [19], [20], [21] split the training model of all workers in the same manner, instead of flexibly splitting the models of different workers based on heterogeneous computing and communication resources as shown in Fig. 1.

As shown in Fig. 2, due to the inherent execution sequence of the layer-by-layer computation as well as the forward and backward propagation for neural networks, the computing resource of the worker is idle while waiting for the intermediate results from the server. When updating the model parameter, the gradient of the current parameter is calculated using a minibatch of data through the forward and backward propagation in each iteration. Different colors in Fig. 2 are used to represent different iterations. To reduce the computational idleness, an effective way is to move part of the computation in the next iteration to the idleness of the previous iteration [23], [24]. For example, when the worker finishes the computing of part a in the first iteration (blue solid-line box marked with a in Fig. 2), it can start the computing of part a (green solid-line box marked with a) in the second iteration while uploading the output feature. By this method, several iterations can be performed in parallel rather than serially as shown in Fig. 2. By subtly arranging the computation of the split parts in different iterations for parallel computing, some related works have effectively reduced the computational idleness [23], [24]. However, these parallel mechanisms [23], [24] are designed for SL with label sharing [14], which means data labels can be shared with others. In the SL system comprising a worker and a server, the model is split into two parts if data labels are shared, where the first part is executed by the worker and the other part is executed by the server, as depicted in Fig. 2(b). As shown in Fig. 2, the execution flow of SL without label sharing is more complex than that of SL with label sharing, because the number of parts executed by the worker changes from one to two and the two parts...
follow the inherent execution sequence. Therefore, we need to design a parallel training mechanism for SL without label sharing.

To obtain a trade-off solution for a multi-objective problem, a traditional method is to transform the multi-objective problem into a single-objective problem by optimizing the weighted sum of the objectives, which requires prior information about the objective preference to determine the weights. Besides, many classic multi-objective evolutionary algorithms (MOEAs) are proposed, such as multi-objective evolutionary algorithm based on decomposition (MOEA/D) [25] and non-dominated sorting genetic algorithm III (NSGA-III) [26]. Moreover, numerous model-based MOEAs are proposed using the decision variable clustering [27], Gaussian process-based inverse modeling [28], dominance relationship classification [29], Pareto rank learning [30], etc. In general, the performance of these model-based MOEAs degrades as decision variables increase due to the increase of computational complexity and data requirement [31]. Because the generative adversarial network (GAN) can approximate high-dimensional data distribution through adversarial learning [32], GAN is applied to multi-objective optimization by approximating the distribution of dominating solutions [31], [33]. In the GAN-powered algorithm [31], [33], the training and solution generation of GAN are performed alternately with the solution selection to improve the quality of generated solutions. Furthermore, multi-objective reinforcement learning algorithms [34] are proposed to deal with multi-objective optimization problems with sequential decision-making. In this paper, the GAN-powered algorithm can be performed by the base station (BS) in the considered HFLS system. The workers only need to upload the information including their computing power and channel state to the BS. The BS can use the collected information to calculate the optimization objectives, and use its computing resource to perform the algorithm. The training of the GAN-powered algorithm hardly consumes the energy of workers, because the energy consumed by uploading the above information to the BS can be ignored compared with uploading the model parameters in HFLS. The training of the HFLS starts after obtaining the splitting decisions and resource allocation with the GAN-powered algorithm, so performing the algorithm introduces the delay in solving the optimization problem.

B. Contributions and Outline

Motivated by the above, we propose a hybrid federated split learning (HFLS) framework in wireless networks, which uses SL to reduce the computational load of workers and implements multi-worker collaborative learning through the multi-worker parallel training and parameter aggregation of FL. Then we jointly optimize the model splitting decisions and communication-computing resource allocation for the HFLS system to reduce the training time and energy consumption of workers. The main contributions are summarized as follows:

- We propose a HFLS framework to combine the advantage of FL and SL. Different from existing works [16], [17], [18], [19], [20], [21] that take the same splitting decision for all workers, the proposed framework allows workers to make different splitting decisions based on their heterogeneous communication and computing capabilities to adjust the communication and local computing burden, resulting in better trade-offs between training time and energy consumption.
- We design a parallel computing scheme for model splitting without label sharing to reduce the computational idleness of workers. The core idea is to move part of the computation in the n-th iteration to the idle time in the \((n-1)\)-th iteration. In the n-th iteration, the gradient is calculated based on the parameter \(w^{n-2}\) that has been obtained by the \((n-2)\)-th iteration, because the parameter \(w^{n-1}\) is not obtained by the start of the n-th iteration. When the gradient of \(w^{n-2}\) is obtained in the n-th iteration, the \((n-1)\)-th iteration is completed and \(w^{n-1}\) is obtained, so the gradient of \(w^{n-2}\) is used to update \(w^{n-1}\) to obtain the parameter \(w^n\). Unlike the standard algorithm, \(w^{n-1}\) is updated by the delayed gradient (the gradient of \(w^{n-2}\)) in the n-th iteration. We theoretically analyze the impact of the delayed gradient on the convergence and derive the number of global rounds required for HFLS to achieve the desired performance.
- We propose a predictive GAN-powered multi-objective optimization algorithm to obtain the set of Pareto-dominating splitting decisions and resource allocation. Specifically, we design a method to find dominance pairs from the current solutions, which is a pair of solutions and one of them dominates the other. With the dominance pairs, the discriminator is trained to learn features from the difference between the dominating solution and the dominated solution. Then the generator can be trained with the discriminator to predict solutions that dominate the current dominating solutions. Experimental results show that the Pareto front found by the proposed algorithm outperforms NSGA-III [26] and GMOA [31].

The rest of this paper is organized as follows. The system model and the problem formulation are introduced in Section II. The convergence analysis of the hybrid federated split learning is presented in Section III. The detail of the proposed algorithm for multi-objective optimization is introduced in Sections IV. Finally, extensive experimental results are presented in Section V, and conclusions are drawn in Section VI.

II. SYSTEM MODEL

We consider a hybrid federated split learning system as shown in Fig. 1, which consists of a base station (BS) and a set of workers \(K = \{1, 2, \ldots, K\}\). Each worker \(k \in K\) has its local dataset \(\{(x_{k,1}, y_{k,1}), \ldots, (x_{k,D_k}, y_{k,D_k})\}\) with the size \(D_k\), where \(x_{k,1}, \ldots, x_{k,D_k}\) are the raw data and \(y_{k,1}, \ldots, y_{k,D_k}\) refer to the corresponding label. The total amount of data for all workers is \(D = \sum_{k \in K} D_k\). By multi-worker collaborative learning, this system aims to find the optimal vector \(W^* \in \mathbb{R}^{1 \times d}\) that minimizes the global
loss function

\[ F(W) = \frac{\sum_{k \in K} D_k F_k(W)}{D} = \frac{\sum_{k \in K} \sum_{d=1}^{D_k} f(x_{k,d}; y_{k,d}; W)}{D}, \]

(1)

where \( F_k(\cdot) \) is the local loss function of worker \( k \). \( f(\cdot) \) is the task-specific loss function, such as cross-entropy. To deal with the non independent and identically distributed (non-IID) data of workers, the proximal term proposed in [35] is added to the local loss function \( F_k(\cdot) \) for limiting the difference between the locally updated parameter \( w \) and the initial global model \( w^0 \). The surrogate local loss function is

\[ h(w; w^0) = F_k(w) + \frac{\mu}{2} || w - w^0 ||^2, \]

(2)

where \( \mu \geq 0 \). So the gradient of the local update is \( \nabla h(w; w^0) = \nabla F_k(w) + \mu (w - w^0) \).

A. Hybrid Federated Split Learning

In the hybrid federated split learning system, if a worker has poor computing power or heavy computing load, the deep neural network trained on the worker can be split to offload part of the computation to the edge server. As shown in Fig. 1, the DNN can be split into three parts, namely the input layer to the \( S_k \)-th layer, the \( (S_k + 1) \)-th layer to the \( H_k \)-th layer, and the \( (H_k + 1) \)-th layer to the output layer, which are denoted as part \( a \), part \( b \) and part \( c \), respectively. User privacy is protected by keeping the raw data and the corresponding label on the worker. Therefore, part \( a \) and part \( c \) are executed on the worker, while part \( b \) can be offloaded to the server. Let \( S_k \) and \( H_k \) be the split decision of the worker \( k \). In particular, we have \( S_k = H_k \) when the DNN is not split, such as the worker 1 in Fig. 1. Let \( I_k \) be the indicator for model splitting, which is defined as

\[ I_k = \begin{cases} 1, & \text{if } S_k = H_k \\ 0, & \text{if } S_k \neq H_k. \end{cases} \]

(3)

In the hybrid federated split learning mechanism, the global aggregation of FL and the device-edge synergy of SL are combined. The execution flow of this mechanism is shown in Fig. 3. Before the training begins, the splitting decisions of all workers are made based on the computing power and channel conditions, and they are fixed throughout the global rounds. At the beginning of the \( t \)-th round, the workers need to download the global aggregated model parameters \( W_{t-1} \) obtained in the \((t - 1)\)-th round from the server. The initial parameter of the worker \( k \) in the \( t \)-th global round is \( w^0_{k,t} = W_{t-1} \). Specifically, the workers without model splitting download the whole \( w^0_{k,t} \). For workers with splitting, \( w^0_{k,t} \) can be rewritten as \( \{w^0_{k,t,a}, w^0_{k,t,b}, w^0_{k,t,c}\} \), and only the parameters of part \( a \) and part \( c \) are downloaded to workers, i.e., \( \{w^0_{k,t,a}, w^0_{k,t,c}\} \). Secondly, the worker \( k \) performs \( N_k \) iterations to update the received parameters. The local model obtained by worker \( k \) after the \( n \)-th iteration of the \( t \)-th global round is denoted as \( w^n_{k,t} \). As shown by worker 1 in Fig. 3, workers without splitting use their own computing resources to compute the whole gradient \( \nabla F_k(w^{n-1}_{k,t}) + \mu(w^{n-1}_{k,t} - w^0_{k,t}) \) by performing the forward and backward propagation in each iteration. As shown by worker \( k \) in Fig. 3, the parameter update for workers with splitting is performed as the vanilla SL without label sharing. \( w^{n-1}_{k,t} \) can be rewritten as \( \{w^{n-1}_{k,t,a}, w^{n-1}_{k,t,b}, w^{n-1}_{k,t,c}\} \), where \( w^{n-1}_{k,t,b} \) is stored and updated by the server. In the forward propagation of model splitting, part \( a \) is executed locally and the output feature of part \( a \) is uploaded to the server for the calculation of part \( b \), then the output of part \( b \) is downloaded for the execution of part \( c \). In the backward propagation, the gradient of part \( c \), part \( b \) and part \( a \) are calculated in order, i.e., \( \nabla F_k(w^{n-1}_{k,t,c}) + \mu(w^{n-1}_{k,t,c} - w^0_{k,t,c}), \nabla F_k(w^{n-1}_{k,t,b}) + \mu(w^{n-1}_{k,t,b} - w^0_{k,t,b}) \) and \( \nabla F_k(w^{n-1}_{k,t,a}) + \mu(w^{n-1}_{k,t,a} - w^0_{k,t,a}) \). The gradient calculation of \( w^{n-1}_{k,t,b} \) is performed separately for different workers in the server, because \( w^{n-1}_{k,t,b} \) is different across workers with different splitting decisions. As the complete gradient of \( w^{n-1}_{k,t} \) is obtained, the update equation for all workers is

\[ w^{n}_{k,t} = w^{n-1}_{k,t} - \eta_n \nabla h(w^{n-1}_{k,t}; w^0_{k,t}) = w^{n-1}_{k,t} - \eta_n (\nabla F_k(w^{n-1}_{k,t}) + \mu(w^{n-1}_{k,t} - w^0_{k,t})), \]

(4)

for \( n = 1, 2, \ldots, N_k \), where \( \eta_n \) is the learning rate in the \( n \)-th iteration of the \( t \)-th global round. Thirdly, the worker \( k \) uploads the locally executed part of \( w^{N_k}_{k,t} \) to the server after completing \( N_k \) iterations. The whole \( w^{N_k}_{k,t} \) is uploaded for workers without model splitting. For workers with splitting, because \( w^{N_k}_{k,t,b} \) is stored in the server, \( w^{N_k}_{k,t,a} \) and \( w^{N_k}_{k,t,c} \) are uploaded. At the end of the \( t \)-th round, the edge server
performs the global aggregation by
\[
W_t = \sum_{k \in K} \frac{D_k}{D} w_{k,t}^N.
\] (5)

Many global rounds are performed to obtain the desired learning performance.

### B. Parallel Computing for Model Splitting

The inherent execution sequence of layer-by-layer computation as well as forward and backward propagation for training neural networks leads to the computational idleness of workers in the vanilla model splitting as shown by worker \( k \) in Fig. 3. This idleness causes the inefficient execution of model splitting. To mitigate it, we design a parallel computing mechanism as shown in Fig. 4(a) to replace the sequential parameter update process of the vanilla model splitting (forward and backward propagations between the parameter download and upload of worker \( k \) in Fig. 3). In Fig. 3 and Fig. 4, we use different colors to represent different iterations, and solid and dashed lines to represent the forward and backward propagation flow respectively. In the vanilla model splitting shown by worker \( k \) in Fig. 3, the \( n \)-th iteration begins when the \((n-1)\)-th iteration ends, and the gradient is calculated based on the parameter \( \hat{w}_{k,t}^{n-1} \) obtained after the \((n-1)\)-th iteration. In the proposed parallel computing scheme shown in Fig. 4(a), part of computation in the \( n \)-th iteration is moved to the idle time in the \((n-1)\)-th iteration. For example, when the worker finishes the computing of part \( a \) in the first iteration (blue solid-line box marked with \( a \) in Fig. 4(a)), it can start the computing of part \( a \) of the second iteration (green solid-line box marked with \( a \)). When the output of part \( b \) of the first iteration is downloaded to the worker, the worker starts to upload the output of part \( a \) of the second iteration while starting to calculate the forward and backward propagation of part \( c \) of the first iteration. Subsequent iterations follow a similar arrangement, resulting in the stage 1 to 4 in Fig. 4(a).

In each stage, a forward and a backward propagation of part \( a \) or part \( c \) is calculated locally. Each stage starts with the uploading to the server and ends with the completion of downloading from the server or local computing.

Applying the proposed parallel computing scheme, the delayed gradient \( \nabla h(\hat{w}_{k,t}^{n-2}; \hat{w}_{k,t}^n) \) is used to update the model parameter at the \( n \)-th iteration. The update equation of workers with splitting changes from the formula (4) to the following equation.

\[
\hat{w}_{k,t}^n = \hat{w}_{k,t}^{n-1} - \eta_t^{n-1} \nabla h(\hat{w}_{k,t}^{n-2}; \hat{w}_{k,t}^0) = \hat{w}_{k,t}^{n-1} - \eta_t^{n-1}(\nabla F_k(\hat{w}_{k,t}^{n-2}) + \mu(\hat{w}_{k,t}^{n-2} - \hat{w}_{k,t}^0)),
\] (6)

for \( n = 1, 2, \ldots, N_k \). If the model of worker \( k \) is split, we use \( \hat{w}_{k,t}^n \) to denote the local model of worker \( k \) obtained after the \( n \)-th iteration of the \( t \)-th global round. This notation can distinguish the parameters obtained by the delayed gradient in the formula (6) from those obtained by the normal gradient in the formula (4). Specifically, \( \hat{w}_{k,t}^{n-1} = \hat{w}_{k,t}^0 = W_{t-1} \). \( \hat{w}_{k,t}^n \) can be written as \([\hat{w}_{k,t}^{n,a}, \hat{w}_{k,t}^{n,b}, \hat{w}_{k,t}^{n,c}]\), where \( \hat{w}_{k,t}^{n,b} \) is stored and updated in the server. In the proposed parallel computing scheme, when starting the \( n \)-th iteration, the \((n-1)\)-th iteration is not completed, so the parameter \( \hat{w}_{k,t}^{n-2} \) obtained by the \((n-2)\)-th iteration is used to calculate gradient in the \( n \)-th iteration. When the complete gradient \( \nabla F_k(\hat{w}_{k,t}^{n-2}) = \nabla F_k(\hat{w}_{k,t}^{n-2,a}) + \mu(\hat{w}_{k,t}^{n-2,a} - \hat{w}_{k,t}^{0,a}) \), \( \nabla F_k(\hat{w}_{k,t}^{n-2,b}) \), \( \nabla F_k(\hat{w}_{k,t}^{n-2,c}) \), \( \nabla F_k(\hat{w}_{k,t}^{n-2,c}) \), \( \nabla F_k(\hat{w}_{k,t}^{n-2,e}) \) is calculated with a minibatch of data in the \( n \)-th iteration, the \((n-1)\)-th iteration is finished and the parameter \( \hat{w}_{k,t}^{n-1} \) is obtained, so the gradient of \( \hat{w}_{k,t}^{n-2} \) is used to update the parameter \( \hat{w}_{k,t}^{n-1} \). For example, the first and the second iteration in Fig. 4(a), i.e., the blue and the green flow, are based on the parameter \( \hat{w}_{k,t}^{n-1} \) and \( \hat{w}_{k,t}^{n-2} \) respectively. When the first iteration is completed, i.e., \( \hat{w}_{k,t}^{n-1} = \hat{w}_{k,t}^0 \), \( \nabla F_k(\hat{w}_{k,t}^{n-2}) \) is obtained, the third iteration (the orange flow) starts with the parameter \( \hat{w}_{k,t}^0 \).

Using the parallel computing mechanism, workers with splitting iterate with the equation (6), while workers without splitting use the equation (4). To integrate the normal gradient and the delayed gradient, we modify the global aggregation equation (5) with the splitting decision indicator \( I_k \) as follows.

\[
W_t = \sum_{k \in K} \frac{D_k}{D} (I_k \hat{w}_{k,t}^N + (1 - I_k) \hat{w}_{k,t}^N).
\] (7)

### C. Adaptive Local Computing Frequency of Parallel Computing for Model Splitting

As shown in Fig. 4(a), the mismatch between the time consumption of worker computing and edge computing in each stage causes the worker to be idle or waiting. For example, in the stage 1 of Fig. 4(a), the time cost of uploading the output of part \( a \), calculating part \( b \) and downloading the output of part \( b \) is larger than that of the forward and backward propagation of part \( c \). In this case, the CPU frequency of the worker is too high, resulting in idleness. We can reduce the CPU frequency of the worker to avoid idleness. Reducing the CPU frequency can also save unnecessary energy consumption, because the computing power of workers is proportional to the cubic of the CPU frequency [36] and the computing delay is inversely proportional to the CPU frequency. In another case, such as stage 3 of Fig. 4(a), the CPU frequency of the worker is too low, causing the worker to wait for the completion of the worker-side computing to enter stage 4 after receiving the gradient from the server. So the CPU frequency of the worker...
can be increased to reduce the waiting time. Considering these
two cases, we adopt the adaptive local computing frequency as shown in Fig. 4 (b).
Mathematically, let \( C_I^F \) and \( C_B^F \) be the number of floating point operations (FLOPs) required by the \( l \)-th layer in the forward and backward propagation of processing each data respectively. \( L \) is the total number of layers of the trained neural network. Denote the batch size of worker \( k \) as \( b_k \). The local computing frequency (in cycle/s) of stage 1 is adaptively set by
\[
T_{k,t}^1 = \max \left\{ T_{k,t}^{UF} + T_{k,t}^{EF} + T_{k,t}^{DF} , \sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) / f_{k,n_k}^{\text{max}} \right\},
\]
where \( n_k \) is the number of FLOPs per cycle. The duration of stage 1 is given by
\[
T_{k,t}^1 = \max \left\{ T_{k,t}^{UF} + T_{k,t}^{EF} + T_{k,t}^{DF} , \sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) / f_{k,n_k}^{\text{max}} \right\}.
\]

D. Time and Energy Consumption

Denote the parameter size (in bit) of the \( l \)-th layer as \( P_l \). The time consumption of downloading and uploading the parameters that need to be updated locally are respectively given by
\[
T_{k,t}^{P_{\text{ParD}}} = \sum_{l=I_1+1}^{L} \frac{P_l}{B_k \log(1 + \frac{\rho_s g_{k,t}}{B_k N_0})},
\]
\[
T_{k,t}^{P_{\text{ParU}}} = \sum_{l=I_1+1}^{L} \frac{P_l}{B_k \log(1 + \frac{\rho_s g_{k,t}}{B_k N_0})},
\]

where \( O_l^F \) denotes the size (in bit) of the intermediate feature output by the \( l \)-th layer in the forward propagation of each data. \( B_k \) is the bandwidth allocated to the worker \( k \). \( p_k \) and \( \rho_s \) are the transmit power of worker \( k \) and the edge server respectively. \( g_{k,t} \) is the channel gain between the worker \( k \) and the server, which is assumed to constant in each global round. \( N_0 \) is the spectral density of the additive white Gaussian noise (AWGN). In (9), \( T_{k,t}^{EF} \) is the time cost of the forward propagation of part \( b \) at the server, which is given by
\[
T_{k,t}^{EF} = \frac{\sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) f_{k,n_k}^{\text{EF}}}{f_{k,n_k}^{\text{EF}}}.
\]

where \( f_{k,n_k}^{\text{EF}} \) is the computing frequency assigned to worker \( k \) by the server, and \( f_{k,n_k}^{\text{EF}} \) is the number of FLOPs per cycle for the server.

Obviously, worker \( k \) calculates with \( f_{k,n_k}^{\text{max}} \) to reduce wait time in stage 1 when
\[
\sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) f_{k,n_k}^{\text{max}} / T_{k,t}^{max} \geq (T_{k,t}^{UF} + T_{k,t}^{EF} + T_{k,t}^{DF}).
\]
Otherwise worker \( k \) calculates in a lower frequency to save energy. Similarly, the local computing frequency in stages 2, 3 and 4 can be obtained by
\[
f_{k,t}^{2} = \sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) / T_{k,t}^{DF} f_{k,n_k}^{DF},
\]
\[
f_{k,t}^{3} = \sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) / T_{k,t}^{EF} f_{k,n_k}^{EF},
\]
\[
f_{k,t}^{4} = \sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) / T_{k,t}^{max} f_{k,n_k}^{max},
\]
respectively. The duration of stages 2, 3 and 4 are
\[
T_{k,t}^{2} = \max \left\{ T_{k,t}^{UB} + T_{k,t}^{EB} + T_{k,t}^{DB} , \frac{\sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) f_{k,n_k}^{DF}}{T_{k,t}^{max}} \right\},
\]
\[
T_{k,t}^{3} = \max \left\{ T_{k,t}^{UB} + T_{k,t}^{EB} + T_{k,t}^{DB} , \frac{\sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) f_{k,n_k}^{EF}}{T_{k,t}^{max}} \right\},
\]
\[
T_{k,t}^{4} = \max \left\{ T_{k,t}^{UB} + T_{k,t}^{EB} + T_{k,t}^{DF} , \frac{\sum_{l=H_k+1}^{L} b_k (C_I^F + C_B^F) f_{k,n_k}^{max}}{T_{k,t}^{max}} \right\},
\]
respectively. The time consumption of uploading the intermediate gradient output by part \( c \) and downloading the gradient output by part \( b \) in backward propagation are
\[
T_{k,t}^{UB} = \frac{b_k O_{b_k}^B}{B_k \log(1 + \frac{\rho_s g_{k,t}^{ UF} }{B_k N_0})},
\]
\[
T_{k,t}^{DB} = \frac{b_k O_{b_k}^B}{B_k \log(1 + \frac{\rho_s g_{k,t}^{ DF} }{B_k N_0})}.
\]
Here, \( O_{b_k}^B \) denotes the size (in bit) of the intermediate gradient output by the \( l \)-th layer in the backward propagation of each data. The time cost of the backward propagation of part \( b \) at the server is
\[
T_{k,t}^{EB} = \frac{\sum_{l=H_k+1}^{L} b_k C_B^F f_{k,n_k}^{EB}}{T_{k,t}^{max}}.
\]

1We can adjust \( e_k \) and \( b_k \) to make \( N_k \) be even.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where $T_i^{\text{max}}$ is the time consumption of the $i$-th global round, that is, the maximum time required for the workers to complete parameter download, local update and parameter upload, i.e.,

$$
T_i^{\text{max}} = \max\{(1 - I_k)T_{k,t}^{\text{sp}} + I_k\left(\frac{E_{k}D_k}{f_k^{\max}}\sum_{l=1}^{L} (C_{l}^P + C_{l}^D) + T_{k,t}^{\text{ParD}} + T_{k,t}^{\text{ParU}}\right)\} k \in K.
$$

(17)

Based on the local computing frequency set by (16), the energy consumption of workers without splitting can be given by

$$
E_{k,t}^{\text{sp}} = \epsilon_k\left(\frac{\tau_{k,t}^{\text{sp}}}{f_k}\right)^3(\tau_k^{\text{max}} - T_{k,t}^{\text{ParD}} - T_{k,t}^{\text{ParU}}) + p_kT_{k,t}^{\text{ParU}}. \quad (18)
$$

The total energy consumption of all workers in $t$-th global round is given by

$$
E_t^{\text{system}} = \sum_{k \in K}\left((1 - I_k)E_{k,t}^{\text{sp}} + I_kE_{k,t}^{\text{nsP}}\right). \quad (19)
$$

E. Problem Formulation

Denote $\varphi \triangleq [S_1, H_1, f_k^P, B_1, \ldots, S_K, H_K, f_K^P, B_K]$ be the optimization variables. By optimizing the model splitting decisions and the allocation of bandwidth and server computing resources, we can minimize the time consumption $V_1(\varphi) = \sum_{t=1}^{T_i^{\text{max}}} T_{k,t}^{\text{max}}$ and the energy consumption $V_2(\varphi) = \sum_{t=1}^{T_i^{\text{max}}} E_t^{\text{system}}$ required for the entire training process. $\tau$ is the number of global rounds required to achieve the desired performance. Therefore, the optimization problem is formulated as

$$
\min_{\varphi} V(\varphi) = [V_1(\varphi), V_2(\varphi)]
$$

s.t. \sum_{k \in K}(1 - I_k)f_k^E \leq f^{E,\text{max}}, \quad (20)

$$
\sum_{k \in K}B_k \leq B^{\text{max}}, \quad (21)

$$
1 \leq S_k \leq H_k < L, \forall k \in K, S_k, H_k \in \mathbb{Z}. \quad (22)

The constraints (20) and (21) denote the feasible regions of allocated server computing frequency and bandwidth, respectively. The constraint (22) indicates that the input and output layer should be kept on the worker to protect privacy. Besides, the decisions $S_k$ and $H_k$ are integers.

We consider the optimization variables $\varphi$ are the same in each global round. This is because optimizing these variables in each round requires full knowledge of the channel conditions in each round, which is difficult in the practical system. In this paper, we consider that the server has the large-scale fading coefficients of all workers and the channel gain changes independently and identically over rounds. In the HFSL system, uploading and downloading cannot be performed at the same time. So we use the same bandwidth for both uplink and downlink, which can simplify the optimization without affecting the splitting decisions.

The optimization problem is multi-objective, which is based on the trade-off between time and energy consumption. Increasing the number of workers with model splitting can reduce the energy consumption of the splitting workers, but it can reduce the server computing power allocated to one worker and increase the training time. Thus, we aim to find a set of Pareto optimal solutions, instead of a single optimal solution. The optimization problem is non-convex, and it is generally hard to obtain the set of Pareto optimal solutions for such a problem. Therefore, a predictive GAN powered multi-objective optimization algorithm is proposed to approximate the set of Pareto optimal solutions.

III. CONVERGENCE ANALYSIS OF THE PROPOSED HFSL

In this section, we analyze the convergence of the proposed HFSL because the delayed gradient may affect the convergence. Although the existing convergence analyses [37], [38] consider the delayed gradient, the cause of their delayed gradient is different from this paper. In [37], workers start the next round of local update concurrently with the global averaging, causing a delay between the arrival of the averaged gradient to workers and the local update. In [38], gradients uploaded by workers to the server can be delayed due to asynchronous parameter aggregation. In this paper, the parameter of workers with splitting is updated by the gradient calculated with the previous parameter. We need to analyze the impact of the delayed gradient on the parameter update compared with the normal gradient. Because the workers with splitting update with the delayed gradient while the others update with the normal gradient, we have to analyze the impact of splitting decisions on the convergence.

A. Assumptions

Assumption 1: (L-smoothness) The local loss function $F_k(\cdot)$ is $L$-smoothness with $L > 0$, i.e., for $\forall w_1, w_2$

$$
F_k(w_2) - F_k(w_1) \leq \langle \nabla F_k(w_1), w_2 - w_1 \rangle + \frac{L}{2}\|w_2 - w_1\|^2. \quad (23)
$$

Assumption 2: (Bounded gradients) There exits a constant bound $G > 0$ on the second moment of the gradient:

$$
\mathbb{E}[\|\nabla F_k(w)\|^2] \leq G^2, \forall w_k. \quad (24)
$$

These assumptions are standard for convergence analysis [37], [39]. L-smoothness is satisfied for most neural networks with smooth activation functions, but not for neural networks with non-smooth components. The convergence analysis can be extended from the smooth case to the non-smooth case based on the subgradient [40]. In this paper, we aim to analyze the effect of the delayed gradient on the convergence of HFSL, and the L-smoothness is assumed for illustrative purposes. Many methods are used to avoid gradient explosion in neural network training, such as gradient clipping [41], which ensures that the second moment of the gradient can be bounded.

B. Analysis of Convergence Bound

To analyze the convergence, we first derive the difference between the parameter $w_{k,t}^o$ and $w_{k,t}^n$ using the equations (4) and (6) in Lemma 1. Then we use the result in Lemma 1 and the smoothness assumption of the loss
function to derive the relationship between $E[||\nabla F(W_{t-1})||^2]$ and $E[F(W_{t-1}) - F(W_t)]$, which is averaged to obtain the convergence result.

**Lemma 1:** The difference between the locally updated parameter $w_{k,t}^n (\hat{w}_{k,t}^n)$ and the initial parameter $w_{k,t}^0$ is bounded by

$$E[||w_{k,t}^n - \hat{w}_{k,t}^0||^2] \leq \alpha_t^n, \quad E[||w_{k,t}^n - w_{k,t}^0||^2] \leq \alpha_t^n,$$

for $n = 1, 2, \ldots, N_{\max}$. Here $\alpha_t^n = n_t^2 \sum_{n=0}^{n_{\max}}(\eta_{t}^{n-1})^2 + \sum_{n=0}^{n_{\max}}(\eta_{t}^{n-1})^2$, and $\alpha_t^n = n_t^2 \sum_{n=0}^{n_{\max}}(\eta_{t}^{n-1})^2 + \sum_{n=0}^{n_{\max}}(\eta_{t}^{n-1})^2$. Thus, $\alpha_t^n$ and $\alpha_t^n$ are the intermediate variables introduced when deriving the expression of $\hat{\alpha}_t^n$. They satisfy $0 \leq \hat{\alpha}_t^n \leq 1$, $0 \leq \tilde{\alpha}_t^n \leq 1$, and $\hat{\eta}_t^n = 1$.

**Theorem 1:** Under Assumption 1 and 2, by setting the learning rate $\eta_t^n = \hat{\eta}_t^n \leq \frac{1}{\tau}$ as a constant, $\eta_t^n \leq \frac{1}{\tau n_{\max}}$ for $n = 1, 2, \ldots, N_{\max} - 1$, we have

$$\frac{1}{\tau} \sum_{t=1}^{T} \left(1 + \sum_{n=2}^{N_{\max}} \eta_t^n \right) E[||\nabla F(W_{t-1})||^2] \leq E[F(W_0)] + \phi(\tau),$$

where $\phi(\tau) = \frac{L^2 + 2L\eta}{2\tau(1-L\eta)} \sum_{t=1}^{T} \sum_{n=2}^{N_{\max}} \eta_t^n \sum_{k=1}^{K} \frac{D_k}{\tau}$.

**Proof:** Please refer to Appendix B. \square

IV. PREDICTIVE GAN-POWERED MULTI-OBJECTIVE OPTIMIZATION ALGORITHM

To obtain the Pareto front of the optimization problem, a predictive GAN-powered multi-objective optimization algorithm is proposed in this section. Unlike GMOEA [31] that uses GAN to approximate the distribution of dominating solutions, the proposed algorithm uses GAN to predict solutions that dominate the current dominating solutions, which can improve the speed of searching for promising candidate solutions. We use the discriminator to learn features from the difference between the dominating and the dominated solutions with the defined dominance pairs, and train the generator with the parameter of discriminator frozen to search for solutions that dominate the current dominating solutions.

A. Preliminary

1) Generative Adversarial Network: GAN [32] is a powerful generative model capable of synthesizing realistic data, such as images and text. The basic architecture of GAN consists of a generator network and a discriminator network, which play a zero-sum game during training. With the generator $G$, the input random noise vector $z$ sampled from the distribution $P_z$ is mapped to the generated data distribution $P_{G(z)}$. The optimization of the generator is to make the difference between the generated data distribution $P_{G(z)}$ and the real data distribution $P_{data}$ as small as possible. Commonly used metrics to measure the difference between different distributions include KL divergence [43], JS divergence [44], and Wassertein distance [45]. The discriminator $D$ can predict the probability that the input is real data, and it is used to distinguish the real data and the generated fake data. The optimization objective of the discriminator is to make the output $D(G(z))$ close to 0 when the input is the generated data $G(z)$, and close to 1 when the input is real data. Specifically, the generator and discriminator are trained with the following min-max function

$$\min_{G} \max_{D} E_{x \sim P_{data}}[\log D(x)] + E_{z \sim P_z}[\log(1-D(G(z)))]$$

where $x \sim P_{data}$ represents that $x$ follows the distribution $P_{data}$. The generator tries to fool the discriminator by maximizing $D(G(z))$. After many rounds of training, the discriminator cannot distinguish the real and fake data, which indicates that the data $G(z)$ generated by the generator is close to the real data.
2) Solution Selection of NSGA-III: NSGA-III [26] is a classic multi-objective evolutionary algorithm (MOEA). Similar to many MOEAs, NSGA-III generally has three steps: offspring generation, solution evaluation and solution selection. To generate offspring solutions, genetic operators (i.e., crossover and mutation) are used based on the parent solutions. Then the offspring solutions are evaluated with the optimized function. After evaluation, the parent and offspring solutions are combined to select new parent solutions for the next generation. These steps are repeated for many generations until the termination criterion is satisfied. To select new parent solutions, the definition of Pareto dominance is necessary.

Definition 1: (Pareto dominance) For any two candidate solutions \( \varphi \) and \( \varphi' \) with the optimized two-objective function \( V(\varphi) \), we have

- \( \varphi \prec \varphi' \) indicates that the solution \( \varphi' \) is strictly dominated by \( \varphi \). Specifically, the two function values obtained by \( \varphi \) are both not greater than that of \( \varphi' \), i.e., \( V_1(\varphi) \leq V_1(\varphi') \) and \( V_2(\varphi) \leq V_2(\varphi') \). Meanwhile, at least one of the two function values obtained by \( \varphi \) is strictly smaller than the corresponding value of \( \varphi' \).
- When one element of \( V(\varphi) \) is strictly larger than that of \( V(\varphi') \) and the other one is strictly smaller than that of \( V(\varphi') \), the solutions \( \varphi \) and \( \varphi' \) are incomparable.
- If there is no solution \( \varphi' \) that satisfies \( \varphi' \prec \varphi \), the solution \( \varphi \) is Pareto optimal. The set of function values achieved by the Pareto optimal solutions is called the Pareto front.

Denote the set of parent and offspring solutions at the \( j \)-th generation as \( P_j \) and \( Q_j \), respectively. Let \( |P_j| \) denote the size of the set \( P_j \), and we have \( |P_j| = |Q_j| = R \). Then we need to select \( R \) elites from the set \( P_j \cup Q_j \) as the parent solution of the next generation, i.e., \( P_{j+1} \). Specifically, the non-dominated solutions in \( P_j \cup Q_j \) are selected into the first non-domination level \( X_1 \). In general, the \( r \)-th non-domination level \( X_r \) consists of the non-dominated solutions in \( (P_j \cup Q_j)/\cup_{r=1}^{r-1} X_r \). This selection step is carried out sequentially from \( X_1 \) to \( X_r \), where the size \( |\cup_{r=1}^{r} X_r| \geq R \) is satisfied for the first time. If \( |\cup_{r=1}^{r} X_r| = R \), we have \( P_{j+1} = \cup_{r=1}^{r} X_r \). If \( |\cup_{r=1}^{r} X_r| > R \), the first level to the \((r+1)\)-th level are first selected to \( P_{j+1} \), and then the remaining \( R - |\cup_{r=1}^{r} X_r| \) solutions are chosen from \( X_r \). In the remaining selection process of NSGA-III [26], the distance to the reference line is considered to ensure the diversity of obtained solutions.

B. Overall Framework

The overall framework of the proposed algorithm is shown in Fig. 5, and the proposed algorithm is outlined in Algorithm 1. Firstly, a hybrid solution generation method is applied to generate the set of offspring solutions \( Q_j \) with the parent solutions \( P_j \). Secondly, the solution selection method of NSGA-III is used to choose \( R \) solutions from \( P_j \cup Q_j \) to obtain the parent solutions of the next generation \( P_{j+1} \). Then the dominance pairs between \( P_{j+1} \) and \( Y_j \cup ((P_j \cup Q_j)/P_{j+1}) \) are searched with Algorithm 2. \( Y_j \) is a set to keep the worse solutions that are close to the parent solutions. Next, the discriminator is trained to learn the relationship between the dominated solutions and the dominating solutions using the dominance pairs. Finally, the generator is trained with the parameters of discriminator frozen to produce better solutions based on the dominating solutions. These steps are repeated until the termination criterion is satisfied. The details about the dominance pair and the training of GAN are introduced in the following subsections.

The proposed algorithm can be used to approximate the set of Pareto optimal solutions for the problem formulated in Section II-E. By selecting Pareto-dominating solutions from the set of candidate solutions in each generation, the obtained solutions can gradually approach the Pareto optimal solutions.

In this paper, the proposed algorithm can be performed by the BS in the HFSL system. The BS can collect information including computing power and channel state of workers to calculate the optimization objectives, and use its computing resource to perform the algorithm. After performing
Algorithm 1 Predictive GAN-Powered Multi-Objective Optimization Algorithm
1: Initialize the parameter of GAN
2: Initialize $R$ candidate solutions as the parent set $P_1$
3: Calculate the mean and variance of $P_1$ as $\mu_1$ and $\sigma_1$, respectively
4: Initialize $R$ candidate solutions as the set being compared $Y_1$
5: for $j = 1$ to $J$ do
6: Randomly sample a value $\delta$ from a uniform distribution between 0 and 1
7: if $\delta \geq 0.5$ then
8: Generate $R$ offspring solutions as $Q_j$ with genetic operators based on $P_j$
else
10: Generate $R$ offspring solutions as $Q_j$ using the generator with the mean and variance of input noise set as $\mu_j$ and $\sigma_j$, respectively
11: end if
12: Select $R$ solutions from $P_j \cup Q_j$ as $P_{j+1}$ using the solution selection of NSGA-III
13: Calculate the mean and variance of $P_{j+1}$ as $\mu_{j+1}$ and $\sigma_{j+1}$, respectively
14: Search for dominance pairs between $P_{j+1}$ and $Y_j \cup ((P_j \cup Q_j)/P_{j+1})$, and obtain $Y_{j+1}$ with Algorithm 2
15: for $m = 1$ to $M$ do
16: Train the discriminator with the loss function (28)
17: end for
18: for $m = 1$ to $M$ do
19: Generate $R$ solutions using the generator with the mean and variance of input noise set as $\mu_{j+1}$ and $\sigma_{j+1}$, respectively
20: Train the generator with the loss function (30)
21: end for
22: end for

Output: The set of approximated Pareto optimal solutions $P_J$.

Algorithm 1, we can obtain the set $P_J$, which contains many approximated Pareto optimal solutions. According to the requirements for training time and energy consumption of HFSL, we can choose an appropriate solution $\varphi = [S_1, H_1, f_1^E, B_1, \ldots, S_K, H_K, f_K^E, B_K]$ from $P_J$. Then the splitting decisions and resource allocation of all workers can be obtained based on the components of $\varphi$. For example, $P_j$ contains two solutions, i.e., $P_j = \{\varphi, \varphi'\}$, and $V_1(\varphi) > V_1(\varphi')$, $V_2(\varphi) < V_2(\varphi')$. If we want the energy consumption $V_2$ of workers to be as small as possible, we choose $\varphi'$ as the solution.

C. Dominance Pair and Training of Discriminator

Dominance pair is a pair of solutions $(\varphi, \varphi')$ in the proposed algorithm, which is used to train the discriminator to learn the dominance relationship between solutions. In the dominance pair, we require that the solution $\varphi'$ is strictly dominated by the other. We hope to learn features from the difference between the dominating solution and the dominated solution, thus guiding the generation of offspring solutions. Intuitively, when the distance between the values of the two solutions is very large, the two solutions may be very different, and the guiding effect on the direction may be very small. So we limit the distance $\|V(\varphi) - V(\varphi')\|_2 \leq \gamma$ to $\|Z_{\varphi}\|$. In the proposed algorithm, dominance pairs are searched between the $P_{j+1}$ and $Y_j \cup ((P_j \cup Q_j)/P_{j+1})$ using the above two requirements. As shown in Fig. 6, let the solution corresponding to the brown point be the dominating solution $\varphi$ of a dominance pair, then the value of the other solution is distributed in the fan-shaped area enclosed by the green dotted line. Here, the set of dominated solutions $Y_j \cup ((P_j \cup Q_j)/P_{j+1})$ is composed of half of the solutions in $P_j \cup Q_j$ with poor performance and the set being compared $Y_j$. As the solutions in $(P_j \cup Q_j)/P_{j+1}$ may be far away or very close to the solution in $P_{j+1}$, we add an extra set $Y_j$ to keep some close to solutions, improving the stability of training.

We can observe that there are many points in the fan-shaped area of Fig. 6, which can form dominance pairs with the brown point. To ease the training burden by reducing training data, we use the reference point to avoid too many dominance pairs formed by one dominating point. Specifically, the coordinates of the reference point are the minimum values of $V_1$ and $V_2$ in all current solutions as shown in Fig. 6, which is denoted by $\psi$. The reference point is connected to the brown
function is defined as discriminator is trained in a supervised method, and the loss strictly dominates the other. Specifically, the output of the discriminator represents the probability that one input \( D(\phi) \) is expected to be 1, while \( \phi' \) of two inputs in the proposed algorithm, i.e., the dominating the discriminator. As shown in Fig. 5, the discriminator has obtained a set of dominance pairs \( Z \). The details of searching and selecting dominance pairs are dominanted solutions with the smallest distance are retained. The dominated solutions in the dominance pairs is calculated. The dominance pairs is calculated. The

\[
F_D = -\frac{1}{|Z|} \sum_{(\phi, \phi') \in Z} \left( \ln(D(\phi)||\phi')) + \ln(1-D(\phi'||\phi)) \right).
\]

(28)

D. Training of Generator and Solution Generation

With the trained discriminator, we can train the generator to predict solutions with better performance than the current dominating solutions as shown by the red arrow in Fig. 6. As shown in Fig. 5, the input of the generator is a random noise vector \( z \), which is sampled from a multivariate normal Gaussian distribution \( P_{z,j} \). The mean vector and covariance matrix of the Gaussian distribution are obtained from the current set of dominating solutions

\[
\mu_j = \frac{1}{|P_j|} \sum_{\phi \in P_j} \phi, \quad \sigma_j = \frac{1}{|P_j|} \sum_{\phi \in P_j} (\phi - \mu_j)(\phi - \mu_j)^T.
\]

(29)

This setting is helpful to generate solutions that approximate the given dominating solutions and reduce the training difficulty [31]. The output of the generator is the generated solution \( \mathcal{G}(z) \), which is expected to be better than the solutions in \( P_j \). The loss function of the generator is

\[
F_G = -\frac{1}{|P_j|} \sum_{\phi \in P_j} \ln \mathcal{D}(\mathcal{G}(z)||\phi), \phi \sim P_{z,j}.
\]

(30)

which is minimized by gradient descent methods. We can see that the probability \( \mathcal{D}(\mathcal{G}(z)||\phi) \) is maximized to make the generated solution \( \mathcal{G}(z) \) strictly dominate the solutions in \( P_j \).

For the solution generation, we adopt the hybrid generation method [31]. To avoid the mode collapse [46] in the training of GAN, genetic operations and the generator are used to generate offspring solutions with the same probability.

V. SIMULATION RESULTS

In this section, we present the detail of the simulation and evaluate the performance of the proposed HFSL based on the predictive GAN-powered multi-objective optimization algorithm.

A. Simulation Setup

1) Simulation Environment: A circular network with a BS at the center is considered in the simulation, serving for \( K = 16 \) workers. The distance \( d_k \) between the worker \( k \) and the BS is distributed uniformly within 2 to 50 meters. The channel gain \( g_{k,l} \) follows the Rayleigh distribution with the mean \( -PL(d_k)/20 \) which we consider the path loss \( PL(d_k)(dB) = 32.4 + 20 \log_{10}(f_k^{\text{carrier}}) + 20 \log_{10}(d_k) \) and we use \( f_{\text{carrier}} = 2.6 \) GHz. The simulation parameters of the HFSL system is summarized in Table I. In particular, the maximum CPU frequency of workers is randomly selected from \{0.8, 1, 1.2\} GHz, and the number of training data of workers is randomly selected from \{2400, 3200, 4000\}. The training dataset is CIFAR-10 [47], and the size of each image is \( 32 \times 32 \times 3 \). The trained neural network is MobileNetV3-Large [22], which is a well-performing and lightweight convolutional neural network.

2) Hyper-Parameters of the Proposed Algorithm: In the proposed multi-objective optimization algorithm, the generator is a fully connected neural network (FCNN) with \( L_G = 3 \) layers, and the number of nodes is \( \xi^l_G = 64 \) for layer \( l = 0, 1, 2, 3 \). The discriminator is a FCNN with \( L_D = 2 \) layers, and the number of nodes is \( \xi^l_D = \xi^l_G = \xi^l_S = [128, 128, 128] \). The learning rates of generator and discriminator are both \( \eta = 4 \times 10^{-4} \). The number of iterations is \( M = 10 \). The losses are optimized with the Adam optimizer [48]. The number of generated solutions is \( R = 100 \). The distance limitation of dominance pairs is \( \gamma = 0.8 \). The number of dominance pairs corresponding to one dominating solution cannot exceed \( \kappa = 6 \).
B. Performance Comparison

To evaluate the performance of the proposed multi-objective optimization algorithm, we compare it with the NSGA-III [26] and GMOEA [31]. For a fair comparison, the generator of GMOEA has the same number of nodes as the proposed algorithm. The discriminator of GMOEA is also a two-layer FCNN with the number of nodes set as \([\xi_0, \xi_1, \xi_2] = [64, 64, 1]\). The solution selection method of NSGA-III is applied to classify solutions in GMOEA. For the NSGA-III, the SBX [49] and the polynomial mutation (PM) [50] are used to generate offspring solutions with the distribution index of crossover and mutation both set as 20. These three algorithms are performed for 5000 generations.

Fig. 7 (a) shows the Pareto fronts obtained by different algorithms. The performance of the Pareto front obtained by the proposed algorithm is better than the other two algorithms, which is reflected in the less energy consumption for the same training time, and less training time for the same energy consumption. Moreover, we compare the Pareto dominating solutions of the designed HFSL and the solution with no model splitting, i.e., FL. The point without splitting is dominated by some solutions obtained from the HFSL framework.

In Fig. 7 (b), we calculate the hypervolume [51] of the Pareto fronts obtained in different generations, and thus obtain the convergence curve of the multi-objective algorithm. The hypervolume indicator refers to the area dominated by the point of Pareto front \(\Omega\) and bounded above by a reference point \(\omega^*\)

\[
HV(\Omega, \omega^*) = \Lambda(\{\omega \in \mathbb{R}^2 | \exists \omega' \in \Omega : \omega' \preceq \omega \text{ and } \omega \preceq \omega^*\})
\]

where \(\Lambda(\cdot)\) is the Lebesgue measure. Note that a bigger value of hypervolume indicates better performance. With the reference point \(\omega^* = [36000, 10000]\), we obtain the convergence curves of different multi-objective algorithms in Fig. 7 (b). We can find that the proposed algorithm converges faster than GMOEA and NSGA-III, and finally achieves a bigger hypervolume.

As for the learning performance of HFSL, we compare the training accuracy and train loss of HFSL with no model splitting and HFSL with all splitting, as shown in Fig. 8. The workers in HFSL with no splitting update the model with normal gradient, i.e., equation (4). In contrast, all workers in HFSL with all splitting update with the delayed gradient (6). The training accuracy curves of HFSL with no splitting and HFSL with all splitting almost coincide, which verifies that the effect of splitting decisions on the convergence of HFSL can be ignored.
Fig. 9. (a) Pareto fronts of different CPU frequency \( f^{E, \text{max}} \) obtained by the proposed algorithm. (b) Averaged number of FLOPs and layers offloaded to the server per worker with different CPU frequency \( f^{E, \text{max}} \).

Fig. 10. The sum of FLOPs offloaded by all workers corresponding to different solutions.

C. Pareto Front Versus the CPU Frequency \( f^{E, \text{max}} \) of Edge Server

The impact of computing resource \( f^{E, \text{max}} \) on Pareto fronts is shown in Fig. 9 (a) with \( B^{\text{max}} \) = 3 MHz. Increasing computing frequency of edge server can also reduce training time and energy consumption. In Fig. 9 (b), we calculate the averaged number of FLOPs and layers offloaded to the server per worker for different \( f^{E, \text{max}} \), i.e.,

\[
\frac{1}{|P_j|} \sum_{p \in P_j} \sum_{k=1}^{K} e_k D_k \sum_{l=S_k+1}^{H_k} (C^F_l + C^B_l) \quad \text{and} \quad \frac{1}{|P_j|} \sum_{p \in P_j} \sum_{k=1}^{K} (H_k - S_k),
\]

which are directly determined by the splitting decisions. As shown in Fig. 9 (b), when the bandwidth is sufficient, the number of FLOPs offloaded to the server increases with the increase of \( f^{E, \text{max}} \). Besides, the increase of \( f^{E, \text{max}} \) can reduce the computation delay of the server, thereby decreasing the training time.

To observe different splitting decisions on one Parent front, we sort all the solutions on the blue curve in Fig. 9 (a) from small to large according to the training time, and take the index of the solutions as the horizontal axis of Fig. 10. Then we calculate the sum of FLOPs offloaded by all workers corresponding to each solution as the vertical axis, i.e.,

\[
\sum_{k=1}^{K} e_k D_k \sum_{l=S_k+1}^{H_k} (C^F_l + C^B_l).
\]

As shown in Fig. 10, the sum of offloaded FLOPs increases with the increase of training time, which also indicates that the energy consumption of workers decreases. Therefore, we can increase the number of offloaded FLOPs to reduce the energy consumption of workers, but this can increase the computing load of the server and result in an increase of training time with the limited server computing resource.

D. Pareto Front Versus the Bandwidth \( B^{\text{max}} \)

The impact of bandwidth \( B^{\text{max}} \) on Pareto fronts is shown in Fig. 11 (a) with \( f^{E, \text{max}} = 6 \) GHz. Increasing bandwidth can greatly reduce training time and energy consumption. On the one hand, the increase of \( B^{\text{max}} \) can reduce the transmission delay of parameters and intermediate results, while reducing the energy cost of the upload. On the other hand, with sufficient computing power of the edge server, increasing the bandwidth can increase the offloaded computing load, as shown in Fig. 11 (b), thereby reducing the energy consumption of workers.

E. Comparison of Algorithm Complexity

The major computational cost of the proposed algorithm is contained in the solution selection of NSGA-III, the search of dominance pairs, and the training of GAN. The complexity of solution selection is \( O(R^2) \) [26] in each generation. Because the set \( Y_j \cup (P_j \cup Q_j)/P_j+1 \) contains at most \((\kappa+1)R\) solutions, searching for dominance pairs requires \( O(\kappa R^2) \) computations in each generation. The complexity of training the discriminator and the generator is \( O(M(\sum_{l=1}^{L_g} c_l^{-1} - c_l^{-1}) + \sum_{l=1}^{L_D} c_l^{-1} - c_l^{-1})) \) in each generation. The total computational complexity of the proposed algorithm is \( O(J(R^2 + M(\sum_{l=1}^{L_g} c_l^{-1} - c_l^{-1}) + \sum_{l=1}^{L_D} c_l^{-1} - c_l^{-1})) \), GMOEA [31] does not need to search for dominance pairs, so its complexity is \( O(J(R^2 + M(\sum_{l=1}^{L_g} c_l^{-1} - c_l^{-1}) + \sum_{l=1}^{L_D} c_l^{-1} - c_l^{-1}))) \). The computational complexity of NSGA-III [26] is \( O(JR^2) \).

The complexity of the proposed algorithm is close to that of GMOEA, but it is bigger than that of NSGA-III.

Compared with traditional multi-objective algorithms, such as NSGA-III, the proposed multi-objective algorithm trains the GAN to predict promising solutions, which increases the delay in solving the optimization problem. However, since this paper focuses on the HFSL system, where the workers have low computing capabilities or the training model is large, the training time of the GAN may be significantly shorter than that of the HFSL. The model complexity of the GAN can...
be much smaller than that of the training model in HFSL. Under the simulation settings in Section V-A, the FLOPs of the generator and the discriminator to process one input data are $1.24 \times 10^4$ and $8.4 \times 10^3$ respectively. In contrast, the FLOPs of MobileNetV3-Large for processing one $32 \times 32 \times 3$ image is $7.31 \times 10^6$. The total FLOPs of training the GAN for 5000 generations is at most $9.63 \times 10^{11}$. With the server CPU frequency $f^E = 6$GHz and $n^E = 2$, the time consumption of training the GAN is 80.25s. As shown in Fig. 7, using the splitting decisions and resource allocation obtained by the proposed algorithm, the minimum training time that HFSL can achieve is 6782.46s, which is much greater than 80.25s. Moreover, under the condition that the energy consumption of HFSL is close, the training of HFSL achieved by the proposed algorithm can save 10146.53-7637.61 = 2508.92s compared with that achieved by NSGA-III. The HFSL training delay reduced by the proposed algorithm is greater than the GAN training delay.

VI. CONCLUSION

In this paper, we proposed a hybrid federated split learning framework to utilize the multi-worker parallel update of FL and the low computational requirement for workers of SL to reduce the training time and energy consumption. To reduce the computational idleness of workers with model splitting, we designed a parallel computing scheme for model splitting without label sharing. Convergence analysis shows that the delayed gradient update introduced by the parallel computing scheme does not affect the convergence rate. Then we formulated a multi-objective optimization problem to find the Pareto-optimal splitting decisions and resource allocation for the proposed HFSL. To solve the problem, we proposed a predictive GAN-powered multi-objective optimization algorithm. Experimental results show that the proposed HFSL enables various trade-offs between training time and energy consumption, and the solutions of HFSL can dominate the solution of FL. Moreover, the Pareto front found by the proposed multi-objective algorithm outperforms the considered baselines.

\section*{APPENDIX A}

\textbf{PROOF OF \textit{LEMMA 1}}

\begin{equation}
\begin{aligned}
\mathbf{w}_{k,t}^0 - \mathbf{w}_{k,t}^n &= - \sum_{n' = 0}^{n-1} \eta_{t'}^{n'} (\nabla F_k(\mathbf{w}_{k,t}^{n'}) + \mu(\mathbf{w}_{k,t}^{n'} - \mathbf{w}_{k,t}^0)) \\
&= (1 - \eta_{t}^{n-1}) \mu(\mathbf{w}_{k,t}^{n-1} - \mathbf{w}_{k,t}^0) - \eta_{t}^{n-1} \nabla F_k(\mathbf{w}_{k,t}^{n-1}).
\end{aligned}
\end{equation}

(32)

The following relation can be derived by applying the equation (32) recursively.

\begin{equation}
\begin{aligned}
E[|w_{k,t}^n - w_{k,t}^0|^2] &= E[|\eta_t^{n-1}\nabla F_k(w_{k,t}^{n-1})|] \\
&+ \sum_{n' = 0}^{n-2} \eta_{t}^{n'} (\prod_{n'' = n'+1}^{n-1} (1 - \eta_{t''}^{n''}) \mu) \nabla F_k(w_{k,t}^{n''})|^2]] \\
&\leq n((\eta_t^{n-1})^2 E[|\nabla F_k(w_{k,t}^{n-1})|^2] \\
&+ \sum_{n' = 0}^{n-2} (\eta_{t}^{n'})^2 (\prod_{n'' = n'+1}^{n-1} (1 - \eta_{t''}^{n''}) \mu) E[|\nabla F_k(w_{k,t}^{n''})|^2]]) \\
&\leq nG^2((\eta_t^{n-1})^2 + \sum_{n' = 0}^{n-2} (\eta_{t}^{n'})^2 (\prod_{n'' = n'+1}^{n-1} (1 - \eta_{t''}^{n''}) \mu)^2)),
\end{aligned}
\end{equation}

(33)

for $n = 1, 2, \ldots, N^\text{max}$. (a) is obtained by the inequality $\|\sum_{n'=1}^{n} w'^n\|^2 \leq n \sum_{n'=1}^{n} \|w'^n\|^2$ for any vectors $w'^n$. (b) is obtained with the Assumption 2.

\begin{equation}
\begin{aligned}
\hat{w}_{k,t}^n - \hat{w}_{k,t}^0 &= - \sum_{n'=1}^{n-2} \eta_{t'}^{n'+1} (\nabla F_k(\hat{w}_{k,t}^{n'}) + \mu(\hat{w}_{k,t}^{n'} - \hat{w}_{k,t}^0)) \\
&= (\hat{w}_{k,t}^{n-1} - \hat{w}_{k,t}^0) - \eta_{t}^{n-1} \mu(\hat{w}_{k,t}^{n-2} - \hat{w}_{k,t}^0) \\
&- \eta_{t}^{n-1} \nabla F_k(\mathbf{w}_{k,t}^{n-2}).
\end{aligned}
\end{equation}

(34)

Letting $0 \leq \theta_t^n \leq 1$ and $0 \leq \hat{\theta}_{t}^n \leq 1$ satisfy the conditions $\theta_t^n + \hat{\theta}_{t}^n = 1$ and $\theta_t^{n-1} \hat{\theta}_{t}^n = \eta_{t}^{n-1} \mu$ for $n = 0, 1, \ldots, N^\text{max} - 1$,
the equation (34) can be reformulated as follows.
\[
\langle \hat{\omega}^0_{t,t} - \hat{\omega}^0_{k,t} \rangle - \theta^0_{t,t} - \theta^0_{k,t} = \hat{\theta}^n_{k,t}((\hat{\omega}^n_{t,t} - \hat{\omega}^n_{k,t}) - \theta^0_{k,t} - \hat{\omega}^n_{k,t} - \hat{\omega}^0_{t,t})
\]
(35)
The following relation can be derived by applying the equation (35) recursively.
\[
\mathbb{E}[\|\hat{\omega}^n_{t,t} - \hat{\omega}^n_{k,t}\|^2] = \mathbb{E}[\|\hat{\omega}^{n-1}_{t,t} - \hat{\omega}^{n-1}_{k,t}\|^2 + \sum_{n'=1}^{n} \eta_t^{n'+1} (\sum_{n''=0}^{n'-2} \hat{\theta}^{n''}_{k,t} + \sum_{n''=1}^{n'-2} \hat{\theta}^{n''}_{k,t} + \|\nabla F_k(\hat{\omega}^{n'}_{t,t})\|^2)]
\]
(36)
for \(n = 1, 2, \ldots, N_{\text{max}}\).

**APPENDIX B**

**PROOF OF THEOREM 1**

Based on the \(L\)-smoothness of \(F_k(\cdot)\), it can be proved that \(F(\cdot)\) is also \(L\)-smooth, i.e.,
\[
\mathbb{E}[F(W_t)] \leq \mathbb{E}[F(W_{t-1}) + \langle \nabla F(W_{t-1}), W_t - W_{t-1} \rangle + \frac{L}{2}\|W_t - W_{t-1}\|^2].
\]
(37)
The last term on the right-hand side of the inequality (37) can be rewritten as
\[
\mathbb{E}[\|W_t - W_{t-1}\|^2] = \mathbb{E}[\|\sum_{k=1}^{K} \frac{D_k}{D} (I_k(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t})) + (1 - I_k)(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t})\|^2]
\]
(38)
where the terms with \(\eta_0^k\) are extracted in (a) because
\[
\sum_{k=1}^{K} \frac{D_k}{D} (I_k(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t})) = -\eta_0^k \nabla F(W_{t-1}).
\]
(b) holds because that \(\sum_{n'=0}^{n} w^{n'} \leq n \sum_{n'=1}^{n} \|w^{n'}\|^2\) for any vectors \(w^{n'}\).

The second term on the right-hand side of the inequality (37) can be rewritten as
\[
\mathbb{E}[\langle \nabla F(W_{t-1}), \sum_{k=1}^{K} \frac{D_k}{D} (I_k(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t})) + (1 - I_k)(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t})\rangle]
\]
(39)
where the terms with \(\eta_0^k\) are extracted in (a). (b) holds because that \(\sum_{n'=0}^{n} w^{n'} \leq n \sum_{n'=1}^{n} \|w^{n'}\|^2\) for any two vectors \(w_1\) and \(w_2\) of the same size.

The second term in the equation (39) can be rewritten as
\[
\mathbb{E}[\|\nabla F(W_{t-1}) - \frac{1}{\eta_t} \sum_{k=1}^{K} \frac{D_k}{D} (I_k(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t}))
+ (1 - I_k)(\hat{\omega}^n_{k,t} - \hat{\omega}^{n-1}_{k,t}))\|^2]
\]
(40)
where \(\alpha^n = nG^2((\eta_1^n)^2 + \sum_{n'=0}^{n-2} (\eta_1^n)^2 \sum_{n''=n'+1}^{n-1} (1 - \eta_1^n)^2)\), and \(\alpha^n = nG^2((\eta_1^n)^2 + \sum_{n'=0}^{n-2} (\eta_1^n)^2 \sum_{n''=n'+1}^{n-1} (1 - \eta_1^n)^2)\).
The following relation can be obtained by combining the formulas (37), (38), (39) and (40).

\[
\mathbb{E}[F(W_t)] \leq \sum_{n=2}^{N_{\text{max}}} \eta_n^{n-1} \frac{1}{2} \left\| \nabla F(W_{t-1}) \right\|^2 + \sum_{n=2}^{N_{\text{max}}} \left( L \eta_n^{n-1} \right) + \sum_{n=2}^{N_{\text{max}}} \left( \eta_n^{n-1} \right) \left( \sum_{k=1}^{K} \frac{D_k}{D} \right) \left( I_k \alpha_k^{n-1} + (1-I_k) \alpha_k^{n-2} \right)
\]

(41)

By setting the learning rate \( \eta_n^0 = \hat{\eta} \leq \frac{1}{2} \) as a constant, \( \eta_n^0 \leq \frac{1}{2L\eta_n} \) for \( n = 1, 2, \ldots, N_{\text{max}} - 1 \), rearranging (41) and calculating the average, we have

\[
\frac{1}{\tau} \sum_{t=1}^{\tau} \left( \eta(1-L\eta) + \sum_{n=2}^{N_{\text{max}}} \frac{\eta_n^{n-1}}{2} \right) \left\| \nabla F(W_{t-1}) \right\|^2 \leq \mathbb{E}[F(W_0)] + \frac{1}{\tau} \sum_{t=1}^{\tau} \sum_{n=2}^{N_{\text{max}}} \eta_n^{n-1} \left( L^2 + \mu^2 \right) \sum_{k=1}^{K} \frac{D_k}{D} \left( I_k \alpha_k^{n-1} + (1-I_k) \alpha_k^{n-2} \right)
\]

(42)

REFERENCES

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, “Toward an intelligent edge: Wireless communication meets machine learning,” IEEE Commun. Mag., vol. 58, no. 1, pp. 19–25, Jan. 2020.

[2] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, Y. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” IEEE Commun. Surveys Tuts., vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[3] M. Chen et al., “Distributed learning in wireless networks: Recent progress and future challenges,” IEEE J. Sel. Areas Commun., vol. 39, no. 12, pp. 3579–3605, Dec. 2021.

[4] C. Zhang, P. Patras, and H. Haddadi, “Deep learning in mobile and wireless networking: A survey,” IEEE Commun. Surveys Tuts., vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.

[5] A. Zappone, M. Di Renzo, and M. Debbah, “Wireless networks design in the era of deep learning: Model-based, AI-based, or both?” IEEE Trans. Commun., vol. 67, no. 10, pp. 7331–7376, Oct. 2019.

[6] Z. Zhao, C. Feng, H. H. Yang, and X. Luo, “Federated learning-enabled intelligent fog radio access networks: Fundamental theory, key techniques, and future trends,” IEEE Wireless Commun., vol. 27, no. 2, pp. 22–28, Apr. 2020.

[7] E. Li, L. Zeng, Z. Zhou, and X. Chen, “Edge AI: On-demand accelerating deep neural network inference via edge computing,” IEEE Trans. Wireless Commun., vol. 19, no. 1, pp. 447–457, Jan. 2020.

[8] L. Liu, J. Zhang, S. Song, and K. B. Letaief, “Hierarchical federated learning with quantization: Convergence analysis and system design,” IEEE Trans. Wireless Commun., vol. 22, no. 1, pp. 2–18, Jan. 2023.
Z. Wang, H. Hong, K. Ye, G. Zhang, M. Jiang, and K. C. Tan, “Manifold interpolation for large-scale multiobjective optimization via generative adversarial networks,” IEEE Trans. Neural Netw. Learn. Syst., early access, Sep. 29, 2021, doi: 10.1109/TNNLS.2021.313158.

Z. Chen, B. Yin, H. Zhu, Y. Li, M. Tao, and W. Zhang, “Mobile communications, computing, and caching resources allocation for diverse services via multi-objective proximal policy optimization,” IEEE Trans. Commun., vol. 70, no. 7, pp. 4498–4512, Jul. 2022.

T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in Proc. Mach. Learn. Syst., vol. 2, 2020, pp. 429–450.

T. D. Burd and R. W. Brodersen, “Processor design for portable systems,” J. VLSI Signal Process. Syst. Signal Image Video Technol., vol. 13, nos. 2–3, pp. 203–221, Aug. 1996.

L. Zhu, H. Lin, Y. Lu, Y. Lin, and S. Han, “Delayed gradient averaging: Tolerate the communication latency for federated learning,” in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 29995–30007.

H. R. Feyzmahdavian, A. Aytekin, and M. Johansson, “A delayed proximal gradient method with linear convergence rate,” in Proc. IEEE Int. Workshop Mach. Learn. Signal Process. (MLSP), Sep. 2014, pp. 1–6.

Y. Zhang, M. Wainwright, and J. C. Duchi, “Communication-efficient algorithms for statistical optimization,” in Proc. Adv. Neural Inf. Process. Syst., vol. 25, 2012, pp. 1–9.

K. Khamaru and M. Wainwright, “Convergence guarantees for a class of non-convex and non-smooth optimization problems,” in Proc. 35th Int. Conf. Mach. Learn., vol. 80, Jul. 2018, pp. 2601–2610.

Z. Zhang, M. Wainwright, T. He, S. Sra, and A. Jadbabaie, “Why gradient clipping accelerates training: A theoretical justification for adaptivity,” in Proc. Int. Conf. Learn. Represent., May 2020, pp. 1–21.

X. Zhang, M. Hong, S. Dhople, W. Yin, and Y. Liu, “FedPD: A federated learning framework with adaptivity to non-IID data,” IEEE Trans. Signal Process., vol. 69, pp. 6055–6070, 2021.

S. Kullback and R. A. Leibler, “On information and sufficiency,” Ann. Math. Statist., vol. 22, no. 1, pp. 79–86, Mar. 1951.

J. Lin, “Divergence measures based on the Shannon entropy,” IEEE Trans. Inf. Theory, vol. 37, no. 1, pp. 145–151, Jan. 1991.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of Wasserstein GANs,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., vol. 30, Dec. 2017, pp. 5769–5779.

M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in Proc. 34th Int. Conf. Mach. Learn., vol. 70, 2017, pp. 214–223.

A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, pp. 1–60.

K. Deb and B. Agrawal, “Simulated binary crossover for continuous search space,” Complex Syst., vol. 9, no. 2, pp. 115–148, 1995.

K. Deb and M. Goyal, “A combined genetic adaptive search (GeneAS) for engineering design,” Comput. Sci. Inf., vol. 26, no. 4, pp. 30–45, Aug. 1996.

L. White, P. Kingston, L. Barone, and S. Huband, “A faster algorithm for calculating hypervolume,” IEEE Trans. Evol. Comput., vol. 10, no. 1, pp. 29–38, Feb. 2006.

Benshun Yin received the B.E. degree in electronic information science and technology from Lanzhou University, Lanzhou, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Electric Engineering, Shanghai Jiao Tong University, China. His research interests include mobile 3C networks and machine learning.

Meixia Tao (Fellow, IEEE) received the B.S. degree in electronic engineering from Fudan University, Shanghai, China, in 1999, and the Ph.D. degree in electrical and electronic engineering from The Hong Kong University of Science and Technology in 2003. She is currently a Professor with the Department of Electronic Engineering, Shanghai Jiao Tong University, China. Her current research interests include wireless caching, edge computing, physical layer multicasting, and resource allocation.

Dr. Tao was a recipient of the 2019 IEEE Marconi Prize Paper Award, the 2013 IEEE Heinrich Hertz Award for Best Communications Letters, the IEEE/CIC International Conference on Communications in China (ICCC) 2015 Best Paper Award, the International Conference on Wireless Communications and Signal Processing (WCSP) 2012 Best Paper Award, and the 2009 IEEE ComSoc Asia–Pacific Outstanding Young Researcher award. She is serving as a member for the IEEE Open Journal of the Communications Society. She has served as a member of the Editorial Board of several other journals as an Editor or a Guest Editor, including the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS from 2015 to 2019. She was also on the Editorial Board of several other journals.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.