CFL-HC: A Coded Federated Learning Framework for Heterogeneous Computing Scenarios

Dong Wang*, Baoqian Wang†, Jinran Zhang‡, Kejie Lu*, Junfei Xie†, Yan Wan§ and Shengli Fu‡
*Department of Computer Science and Engineering, University of Puerto Rico at Mayagüez
†Department of Electrical and Computer Engineering, San Diego State University
‡Department of Electrical Engineering, University of North Texas
§Department of Electrical Engineering, University of Texas at Arlington

***Abstract*—Federated learning (FL) is a promising machine learning paradigm because it allows distributed edge devices to collaboratively train a model without sharing their raw data. In practice, a major challenge to FL is that edge devices are heterogeneous, so slow devices may compromise the convergence of model training. To address such a challenge, several recent studies have suggested different solutions, in which a promising scheme is to utilize coded computing to facilitate the training of linear models. Nevertheless, the existing coded FL (CFL) scheme is limited by a fixed coding redundancy parameter, and a weight matrix used in the existing design may introduce unnecessary errors. In this paper, we tackle these issues and propose a novel framework, namely CFL-HC, to facilitate CFL in heterogeneous computing scenarios. In our framework, we consider a computing system consisting of a central server and multiple computing devices with original or coded datasets. Then we specify an expected number of input-output pairs that are used in one round. Within such a framework, we formulate an optimization problem to find the best deadline of each training round and the optimal size of the computing task allocated to each computing device. We then design a two-step optimization scheme to obtain the optimal solution. To evaluate the proposed framework, we develop a real CFL system using the message passing interface platform. Based on this system, we conduct numerical experiments, which demonstrate the advantages of the proposed framework, in terms of both accuracy and convergence speed.**

***Index Terms*—Federated learning, edge computing, heterogeneous, coded computing, task allocation and scheduling, message passing interface**

I. Introduction

In recent years, we have witnessed a tremendous increase in data produced by smart edge devices along with a rising need to train learning models based on these data while preserving data privacy [1]. To address these needs, a promising learning paradigm, namely, federated learning (FL) [2], has attracted significant attention because FL allows edge devices to collaboratively train a global model without sharing the local raw data, and thus can protect data privacy [3], [4].

Despite such salient features, FL also faces many design challenges in the real-world networks, especially how to cope with the heterogeneous computing environments that are common in edge computing [3], [4]. In practice, the heterogeneity can have multiple forms, in terms of computing capability, communication capacity, energy capacity, etc. In general, all these factors can affect the performance of model training in FL.

To address the aforementioned challenge, many existing studies design various scheduling schemes for different FL systems. For instance, the authors in [5] proposed a scheduling policy for FL in a heterogeneous edge computing scenario, in which edge devices share the same wireless channel with limited capacity. For this FL system, they designed an age-based scheduling (ABS) scheme that schedules slow devices less frequently while guaranteeing data in these slow devices can contribute to the training of the learning model before a threshold. The authors then demonstrated that the proposed scheduling scheme can lead to faster convergence of a classification model that is trained by using the classical MNIST database with approximately 70% converged accuracy.

In addition to the scheduling schemes, coded computing schemes have also been introduced recently [6], [7]. While most of these coded computing schemes aim to accelerate matrix multiplication, which is a building block in machine learning, a novel coded computing scheme, namely, coded federated learning (CFL), was proposed recently in [8] that can accelerate the training of linear regression models in a heterogeneous computing environment. Compared to coded matrix multiplication, the key novelty of CFL is that only the encoding process is required in the initialization stage of FL. Consequently, the privacy of raw data can still be preserved because the confidentiality of raw data can be protected by a random encoding matrix. Moreover, despite the errors introduced by the encoding process, using coded data in the training stage seems to speed up the convergence of the learning model. More recently, the authors in [9] further extended the CFL model to solve more general non-linear problems. Specifically, they demonstrated that the inputs in the MNIST database can first be converted by using random Fourier features (RFFs) to approximate the radial basis function (RBF) kernel, and then the converted input data and output labels can be used to train a linear regression model, which can quickly converge to about 95% accuracy.

Although these recent studies push forward the development of FL, we notice that these schemes still have some

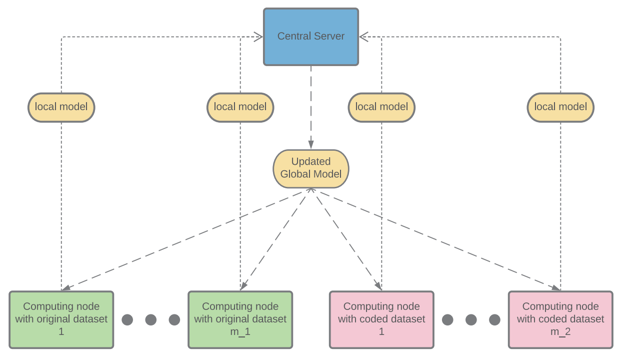

Central Server
local model
local model
local model
local model
Updated Global Model
Computing node with original dataset 1
Computing node with original dataset m_1
Computing node with coded dataset 1
Computing node with coded dataset m_2

Fig. 1: the System Model

limitations. First, there is a lack of a clear formulation that aims to optimize both the deadline to execute a training round and the amount of data that shall be used by each computing device to train the local model in one training round. In addition, the CFL model in [8] and [9] uses only one extra computing device, which may not fully exploit the computing resources in edge systems. Moreover, the encoding process in the CFL model [8], [9] uses a fixed coding redundancy and uses a weight matrix that may introduce unnecessary coding errors, which will be analyzed in Section II-C.

In this paper, we aim to tackle the above issues to advance the state-of-the-art. Specifically, we propose a novel coded FL scheme, namely, coded federated learning framework for the heterogeneous computing environment (CFL-HC). In our framework, we consider an FL system that consists of a central server, multiple computing devices with original data, and multiple computing devices that store coded data. In terms of learning, we apply the stochastic gradient descent approach and specify an expected number of input-output pairs that are used in one round. Within such a framework, we formulate an optimization problem to find the best deadline of each training round and the optimal load allocated to each computing device. We then design a two-step optimization scheme to obtain the optimal solution. To evaluate the proposed framework, we develop a real CFL prototype, based on which we conduct numerical experiments that confirm the advantages of the proposed framework in terms of both accuracy and convergence speed.

The rest of this paper is organized as follows. In Section II, we describe the framework of the proposed CFL-HC, including the system model, the learning model, and the encoding strategy. Next, in Section III, we formulate an optimization problem and then solve it using a two-step method. To evaluate the proposed framework, we implement a prototype for CFL-HC and conduct experiments in Section IV. Finally, we conclude this paper in Section V.

II. The CFL-HC Framework

A. The System Model

In this paper, we consider a generic FL system that consists of one central server and multiple edge computing devices. As illustrated in Fig. 1, we consider that there are two types of computing devices, i.e., Type 1 devices that store original datasets, and Type 2 devices that store coded datasets. To facilitate further discussions, we let m_1 ($m_1 > 1$) be the number of Type 1 devices and let m_2 ($m_2 \geq 0$) be the number of Type 2 devices. Moreover, we let $M = m_1 + m_2$ and we assign an integer i ($1 \leq i \leq M$) to each computing device such that all indices from 1 to m_1 are assigned to Type 1 devices and the rest are assigned to Type 2 devices. Here we note that, if $m_2 = 0$, then the FL system becomes a traditional FL system without using coded computing.

In the CFL-HC framework, the central server is responsible for the following tasks. In particular, during the initialization stage, the central server must (1) initialize and broadcast a global model, (2) determine the size of the coded dataset generated by each Type 1 device, and (3) determine the computing task assigned to each computing device in each training round. During the model training stage, the central server must (4) broadcast the latest global model at the beginning of the training round, (5) set a timer then wait for the training parameters sent by computing devices before timeout, (6) aggregate the training parameters and (7) update the global model using the aggregated training parameters.

During the initialization stage, each computing device must prepare a local model based on the global model broadcasted by the central server. Moreover, each Type 1 computing device must generate a certain amount of coded data and then distribute them to Type 2 devices after it receives the instructions from the central server. On the other hand, each Type 2 device must store coded data from different Type 1 devices separately.

During the training stage, each computing device first updates its local model using the new model parameters broadcasted by the central server. It will then randomly choose a subset of local data, depending on the number of tasks specified by the central server, to train the local model. Finally, it will send the local training parameters (e.g., gradients) to the central server.

B. The Learning Model

Following recent studies in [8] and [9], we consider the training of a linear regression model. Usually, the input of a linear model is a data point $\boldsymbol{x}$ ($\boldsymbol{x} \in \mathbb{R}^{1\times d}$ and $\boldsymbol{x} = [x_1, x_2, \cdots, x_d]$), the output of the model is y ($y \in \mathbb{R}$), and the model to be trained is $y = \boldsymbol{x}\boldsymbol{\beta} + \beta_0$, where β_0 ($\beta_0 \in \mathbb{R}$) is the bias and $\boldsymbol{\beta}$ ($\boldsymbol{\beta} \in \mathbb{R}^{d\times 1}$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_d]^T$) is a d-dimensional vector of linear coefficients.

In this paper, to simplify the discussions, we let input data point $\boldsymbol{x}$ be a $(d+1)$-dimensional data point, i.e.,

$\boldsymbol{x} \in \mathbb{R}^{1\times(d+1)}$, such that the first element in $\boldsymbol{x}$, denoted as x_0, is constant 1. Similarly, we let vector $\boldsymbol{\beta}$ be a $(d+1)$-dimensional vector, i.e., $\boldsymbol{\beta} \in \mathbb{R}^{(d+1)\times 1}$, such that the first element in $\boldsymbol{\beta}$ is the bias β_0. In this manner, the model can be simplified to $y = \boldsymbol{x}\boldsymbol{\beta}$, for which the objective of model training is to find the best $\boldsymbol{\beta}$ that minimizes an error function.

We assume that each computing device i stores a total of n_i input-output pairs. In Type 1 device ($1 \leq i \leq m_1$), the input-output pairs are not coded and are defined by set $S^{(i)} = \{(\boldsymbol{x}_j^{(i)}, y_j^{(i)}) \mid \boldsymbol{x}_j^{(i)} \in \mathbb{R}^{1\times(d+1)}$ and $y_j^{(i)} \in \mathbb{R}$, where $x_{j0}^{(i)} = 1$ and $1 \leq j \leq n_i\}$. In Type 2 device ($m_1 < i \leq M$), the input-output pairs are coded (to be explained in the next subsection) and are defined by set $S^{(i)} = \{(\tilde{\boldsymbol{x}}_j^{(i)}, \tilde{y}_j^{(i)}) \mid \tilde{\boldsymbol{x}}_j^{(i)} \in \mathbb{R}^{1\times(d+1)}$ and $\tilde{y}_j^{(i)} \in \mathbb{R}$, where $1 \leq j \leq n_i\}$.

To train the model, we apply a *stochastic gradient decent* (SGD) approach. Specifically, we consider that in each training round, a total of r input-output pairs will be randomly chosen from existing datasets in the training process. We further assume that computing device i will randomly choose ℓ_i input-output pairs in its dataset (original or coded) for training such that $\sum_{i=1}^{M} \ell_i = r$. In Section III, we will discuss how to find the optimal task allocation for each computing device.

Next, suppose the j-th input-output pair on the i-th computing device $(\boldsymbol{x}_j^{(i)}, y_j^{(i)})$ is chosen to train the model, then the estimated output $\hat{y}_j^{(i)}$ is

$$\hat{y}_j^{(i)} = \boldsymbol{x}_j^{(i)}\boldsymbol{\beta} \tag{1}$$

and the corresponding error $e_j^{(i)}$ is

$$e_j^{(i)} = y_j^{(i)} - \hat{y}_j^{(i)}. \tag{2}$$

Without the loss of generality, we consider that the first ℓ_i pairs are chosen in a training round, then the total square error $E^{(i)}$ on device i can be defined by

$$E^{(i)} = \sum_{j=1}^{\ell_i}(e_j^{(i)})^2 = \sum_{j=1}^{\ell_i}(y_j^{(i)} - \hat{y}_j^{(i)})^2. \tag{3}$$

Then, to train the model locally, computing device i can try to find the gradient of $E^{(i)}$ with respect to $\boldsymbol{\beta}$:

$$\begin{aligned} \triangledown_{\boldsymbol{\beta}} E^{(i)} &= 2\sum_{j=1}^{\ell_i}(\boldsymbol{x}_j^{(i)})^T(\hat{y}_j^{(i)} - y_j^{(i)}) \\ &= 2(\boldsymbol{X}^{(i)})^T(\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}), \end{aligned} \tag{4}$$

where $\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^{\ell_i\times 1}$ is created by grouping all ℓ_i estimated outputs into a vector, $\boldsymbol{y}^{(i)} \in \mathbb{R}^{\ell_i\times 1}$ is created by grouping all ℓ_i outputs into a vector, and $\boldsymbol{X}^{(i)} \in \mathbb{R}^{\ell_i\times(d+1)}$ is generated by grouping all ℓ_i inputs into a matrix.

To update the global model, we assume that I is the set of computing devices that return gradients before timer reaches timeout. If $I = \emptyset$, then the same $\boldsymbol{\beta}$ will be sent to all computing devices in the next round. Otherwise, a new $\boldsymbol{\beta}$ can be calculated by

$$\boldsymbol{\beta} - \frac{\eta}{\sum_{i\in I}\ell_i}\left(\sum_{i\in I}\triangledown_{\boldsymbol{\beta}} E^{(i)}\right), \tag{5}$$

where $\eta \geq 0$ is the learning rate.

Lastly, regarding the time complexity, the computing time of $\hat{\boldsymbol{y}}^{(i)}$ is determined by implementing the matrix-vector multiplication with two matrices in dimensions of $\ell_i \times (d+1)$ and $(d+1)\times 1$. Therefore, the time complexity of computing $\hat{\boldsymbol{y}}^{(i)}$ is $\boldsymbol{O}(\ell_i d)$. Similarly, to compute the gradient for the i-th node $\triangledown_{\boldsymbol{\beta}} E^{(i)}$, the time complexity is also $\boldsymbol{O}(\ell_i d)$.

C. The Coded Computing Model

In [8] and [9], the authors used the same coded computing scheme. Specifically, they introduced a coding redundancy parameter c and assumed that computing device i can return gradient on time with probability p_i, which is known in advance. Then, during the initialization stage, each Type 1 computing device i generates a random matrix $\boldsymbol{G}_i$ of size $c \times \ell_i$, and a diagonal weight matrix $\boldsymbol{W}_i$ of size $\ell_i \times \ell_i$, in which each diagonal element equals to $\sqrt{1-p_i}$. Based on these two matrices, they designed a coding scheme for each computing device i using the following equations:

$$\tilde{\boldsymbol{X}}^{(i)} = \boldsymbol{G}_i\boldsymbol{W}_i\boldsymbol{X}^{(i)}, \tag{6}$$

$$\tilde{\boldsymbol{y}}^{(i)} = \boldsymbol{G}_i\boldsymbol{W}_i\boldsymbol{y}^{(i)}, \tag{7}$$

Next, they considered that all Type 1 computing devices send coded data, i.e., $\tilde{\boldsymbol{X}}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)}$, to the central server, which will aggregate the received coded data by

$$\tilde{\boldsymbol{X}} = \sum_{i=1}^{n}\tilde{\boldsymbol{X}}^{(i)}, \tag{8}$$

$$\tilde{\boldsymbol{y}} = \sum_{i=1}^{n}\tilde{\boldsymbol{y}}^{(i)}. \tag{9}$$

Using our system model, their FL system contains a single Type 2 device, which is the central server itself and this device always has c coded input-output pairs. Consequently, in each training round, they assumed that the central server can always compute a local gradient before the deadline by

$$\begin{aligned} \triangledown_{\boldsymbol{\beta}}\tilde{E} &= \frac{2}{c}(\tilde{\boldsymbol{X}})^T(\tilde{\boldsymbol{X}}\boldsymbol{\beta} - \tilde{\boldsymbol{y}}), \\ &\approx 2\sum_{i=1}^{m_1}(1-p_i)\triangledown_{\boldsymbol{\beta}} E^{(i)}. \end{aligned} \tag{10}$$

Finally, the authors in [8] and [9] considered that the **expected** summation of received gradients is

$$2\sum_{i=1}^{m_1} p_i \triangledown_{\boldsymbol{\beta}} E^{(i)}, \tag{11}$$

which can be added to the locally computed gradient to cancel out p_i.

Clearly, the above coding scheme is first limited by a fixed coding redundancy parameter for every computing device. Moreover, we can observe that this scheme introduces unnecessary error in the aggregation stage. Specifically, by the end of a training round, the central server can receive the gradients from a subset of Type 1 computing devices, and as such the actual summation of gradients is never the same as Eq. (11). Therefore, adding the local gradient in Eq. (10) cannot give the desired summation of gradients.

To solve the above issue, in our CFL-HC framework, we design a different coding scheme. Specifically, we consider that each Type 1 computing device calculates a set of c_i coded input-output pairs during the initialization stage, using

$$\tilde{\boldsymbol{X}}^{(i)} = \frac{1}{\sqrt{c_i}}\boldsymbol{G}_i\boldsymbol{X}^{(i)}, \tag{12}$$

$$\tilde{\boldsymbol{y}}^{(i)} = \frac{1}{\sqrt{c_i}}\boldsymbol{G}_i\boldsymbol{y}^{(i)}, \tag{13}$$

where $\boldsymbol{G}_i$ is a $c_i \times n_i$ dimensional random coding matrix, in which every element can be either $+1$ or -1 with 50% probability. Then this Type 1 computing device will send the coded dataset to one or more Type 2 devices. Consequently, when a Type 2 device computes gradients locally, the errors will be only introduced by the coding matrices.

Due to limited space, in this paper, we will skip the analysis about the coding error when we use the coded data to generate gradients. In the next section, we focus on the optimal task allocation in CFL-HC. We will then develop a prototype of the CFL-HC system and evaluate its overall performance in Section IV.

III. An Optimal Task Scheduling Scheme

In this section, we first formulate an optimal task scheduling problem for the CFL-HC framework. We then design a two-step alternative solution that solves the optimization problem.

A. Problem Formulation

In the last section, we have explained the details of the CFL-HC framework, in which the proposed learning model utilizes the SGD approach for training such that each computing device i randomly chooses ℓ_i input-output pairs in a round and $\sum_{i=1}^{M}\ell_i = r$. To formulate an optimal task scheduling problem, we first define a task allocation vector $\boldsymbol{\ell} = (\ell_1, ..., \ell_M)$. We then define T as the random variable representing the time to complete a training round. Based on these definitions, we formulate an optimal task scheduling problem as follows to minimize the expected task completion time $\mathbb{E}[T]$.

$$\begin{aligned} \mathcal{P}_{main} : &\underset{\boldsymbol{\ell}}{minimize} && \mathbb{E}[T] \\ &subject\ to: && \ell_i \le n_i, \forall 1 \le i \le M \\ & && \sum_{i=1}^{M}\ell_i = r, \end{aligned} \tag{14}$$

where the first constraint specifies that the assigned load to every computing device cannot exceed the size of the dataset stored locally, and the second constraint specifies the requirement for SGD.

According to the time complexity analysis at the end of Section II-B, we can observe that the time complexity to calculate gradients in any computing device i is linear in the assigned load, i.e., ℓ_i. We also assume that the communication time is less significant compared to the computing time. Therefore, following the studies on coded matrix multiplication in [6] and [7], we assume that the duration from the time that the central server starts a training round to the time that it receives training parameters from computing device i, denoted as T_i, follows a shifted exponential distribution with parameters a_i and μ_i

$$Pr[T_i \le t] = \begin{cases} 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)} & t \ge a_i l_i \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

B. A Two-Step Solution

Since $\mathcal{P}_{main}$ is hard to solve, we develop a two-step solution that is similar to the solutions proposed in [6] and [7]. Specifically, in Step 1, we assume that a feasible time t is given to complete a training round and we let $X(t)$ be the total number of input-output pairs that have been used by a subset of computing devices to compute local gradients that are received by the central server before t. Consequently, the optimization formulation for Step 1 is defined as follows:

$$\begin{aligned} \mathcal{P}_{alt}^{(1)} : \boldsymbol{\ell}^*(t) = arg\,\underset{\boldsymbol{\ell}}{max} \quad & \mathbb{E}[X(t)] \\ subject\ to \quad & \ell_i \le n_i, \forall 1 \le i \le M, \end{aligned} \tag{16}$$

where $\boldsymbol{\ell}^*(t) = [\ell_1^*(t), ..., \ell_M^*(t)]^T$ is a vector of optimal loads assigned to all computing devices.

To solve $\mathcal{P}_{alt}^{(1)}$, we compute the expected value of $X(t)$ by using the assumption that the task completion time for each computing device follows a shifted exponential distribution:

$$\mathbb{E}[X(t)] = \sum_{i=1}^{M}\mathbb{E}[X_i(t)] = \sum_{i=1}^{M}\ell_i(1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)}). \tag{17}$$

Eq. (17) shows that we can choose ℓ_i independent from ℓ_j $(\forall j, j \neq i)$ to maximize $\mathbb{E}[X_i(t)]$. Since problem $\mathcal{P}_{alt}^{(1)}$ requires constraints $\ell_i \le n_i$ for all i, the optimization approach to find the optimal $\ell_i^*(t)$ is different from the methods used in [6]. Specifically, we first derive partial derivative

$$\frac{\partial}{\partial \ell_i}\mathbb{E}[X_i(t)] = 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)}(\frac{\mu_i t}{\ell_i} + 1) \tag{18}$$

and our analysis shows that the above derivative function approaches to 1 when ℓ_i tends to 0 and it is not increasing with respect to ℓ_i.

Consequently, to find $\ell_i^*(t)$, we first use $\ell_i = n_i$ in Eq. (18). If the derivative is positive, then $\ell_i^*(t) = n_i$.

TABLE I: Configuration and Parameters of Computing Devices

Computing Node	n_i	a_i	μ_i
Node 1 with Original Data	400	15×10^{-4}	1×10^{4}
Node 2 with Original Data	400	20×10^{-4}	1×10^{4}
Node 3 with Original Data	400	25×10^{-4}	1×10^{4}
Node 4 with Original Data	400	30×10^{-4}	1×10^{4}
Node 5 with Coded Data	800	20×10^{-4}	1×10^{4}

Otherwise, if the derivative is negative, we can use numerical method to find $\ell_1^*(t)$ such that the derivative is 0.

After solving $\mathcal{P}_{alt}^{(1)}$, we can obtain the optimal load vector $\boldsymbol{\ell}^*(t)$. We let $X^*(t)$ be the value of $X(t)$ when the optimal load vector $\boldsymbol{\ell}^*(t)$ is used by all computing devices, which is

$$X^*(t) = \sum_{i=1}^{M} X_i^*(t) = \sum_{i=1}^{M} \ell_i^*(t) \times \mathbf{1}\{T_i \le t\}, \tag{19}$$

where function $\mathbf{1}$ equals to 1 when $T_i \le t$ and it is 0 otherwise.

Based on the above definitions, we can formulate the optimization problem in the second step by

$$\begin{aligned} \mathcal{P}_{alt}^{(2)}: \quad & minimize \quad t \\ & subject\ to \quad Pr[X^*(t) < r] = o(\frac{1}{M}). \end{aligned} \tag{20}$$

To solve $\mathcal{P}_{alt}^{(2)}$, we can find an optimal time τ^* such that $\mathbb{E}[X^*(\tau^*)] = r$. Since function $\mathbb{E}[X^*(t)]$ increases monotonically with respect to t, τ^* can be found numerically.

Due to the limited space, we will skip the theoretical analysis that shows the optimality of our solution obtained by the aforementioned two-step approach.

IV. Numerical Results

A. Heterogeneous Computing System for Federated Learning

To implement the proposed CFL-HC, we utilize *Message Passing Interface* (MPI), which is a common interface for distributed computing. Specifically, we implement all the source codes in Python and we use the open-source package mpi4py to access the MPI platform, which is installed separately.

Table I illustrates the configuration of our prototype, which consists of one central server, $m_1 = 4$ computing devices that store original data, and $m_2 = 1$ computing device that stores coded data. To facilitate our experiments, we assume that each computing device stores a limited number of input-output pairs, indicated in the second column. Moreover, Table I shows the configuration for the shifted exponential distribution parameters a_i and μ_i for each computing device i. Here, the unit of a_i is second per input-output pair, and the unit of μ_i is the number of input-output pairs per second.

TABLE II: Experiment Settings for CFL-HC with Different r

r	τ^*	$\boldsymbol{\ell}$	$[c_1, c_2, c_3, c_4]$
400	0.2021s	[113, 87, 71, 60, 87]	[0, 127, 283, 390]
600	0.3027s	[169, 131, 107, 90, 131]	[0, 125, 282, 393]
800	0.4033s	[225, 174, 142, 120, 174]	[0, 127, 283, 390]
1000	0.5039s	[281, 217, 178, 150, 217]	[0, 129, 281, 390]
1200	0.6054s	[337, 261, 213, 181, 261]	[0, 127, 284, 389]

B. Simulation Settings

In our experiments, we follow the method in [8] and first use a vector $\boldsymbol{\beta}^*$ to generate n_i input-output pairs in all Type 1 computing devices,

$$\boldsymbol{y}^{(i)} = \boldsymbol{X}^{(i)}\boldsymbol{\beta}^* + \boldsymbol{n}, \tag{21}$$

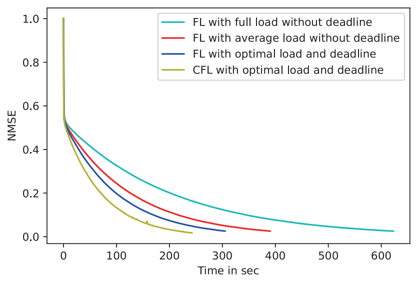
where each row in $\boldsymbol{X}^{(i)}$ is uniformly distributed in a d-dimensional space, and $\boldsymbol{n}$ is an n_i-dimensional vector of random Gaussian noise.

In our experiments, we compare the following four FL schemes.

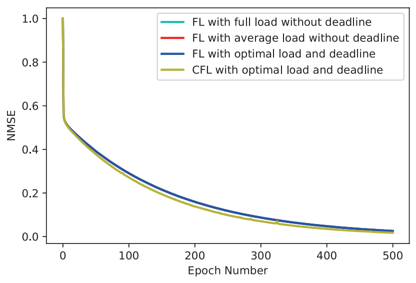
- **FL with full load without deadline**: in this scheme, coded computing is not used, all original data are used to calculate gradients, and there is no deadline.
- **FL with average load without deadline**: in this scheme, coded computing is not used, there is no deadline, and the average load r_i is assigned to each computing device i according to the following equation
$$\ell_i = r_i = \frac{r \times n_i}{\sum_{i=1}^{m_1} n_i}. \tag{22}$$
- **FL with optimal load and deadline**: in this scheme, coded computing is not used, but the optimal load and deadline are obtained by using the two-step optimization approach explained in Section III-B.
- **CFL-HC with optimal load and deadline**: in this scheme, coded computing is used and the optimal load and deadline are calculated by using the two-step optimization approach. Moreover, in the initialization stage, c_i coded input-output pairs are generated, in which c_i is given by
$$c_i = \frac{\max(0, r_i - \ell_i)}{\sum_{i=1}^{m_1} \max(0, r_i - \ell_i)} \times \left(\sum_{i=m_1+1}^{M} n_i \right), \tag{23}$$
where r_i is calculated by using Eq. (22), and the max function helps to generate coded input-output pairs only for slow devices. Table II shows the values of optimal deadline, optimal load, and c_i with respect to different r values.

C. Experimental Results

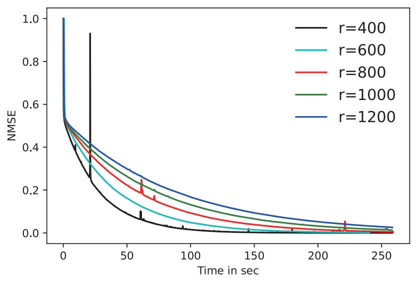
To evaluate and compare the performance of different schemes, we follow [8] and investigate the *normalized mean squared error* (NMSE) of $\boldsymbol{\beta}$ during the training, which is $\frac{||\boldsymbol{\beta}-\boldsymbol{\beta}^*||^2}{||\boldsymbol{\beta}^*||^2}$.



(a) the Performance of 4 Different Schemes vs. Time ($r = 1000$).



(b) the Performance of 4 Different Schemes vs. Epoch ($r = 1000$).



(c) the Performance of CFL-HC with Different r vs. time.

Fig. 2: Performance of the Proposed CFL-HC Framework

In Fig. 2, we compare the performance of the four FL schemes over time in 500 rounds when $r = 1000$. Fig. 2a shows that, compared to other FL schemes without using coded computing, the proposed CFL-HC scheme has a much faster convergence speed. In particular, the proposed CFL-HC scheme can finish 500 rounds of training in 240 seconds, while all non-coding schemes need a much longer time to finish the training task. This experiment demonstrates the significant advantage of the CFL-HC framework.

In Fig. 2b, we compare the performance of the four FL schemes versus epoch when $r = 1000$. This figure shows that the proposed CFL-HC scheme achieves a slightly smaller error after the same number of training rounds. On the other hand, all non-coding FL schemes have almost identical NMSE after the same number of training rounds. These results confirm that the error introduced by our coding design is reasonably small.

Finally, in Fig. 2c, we investigate the impact of r in CFL-HC. As we can observe, CHL-HC with $r = 400$ seems to have the best convergence performance, which suggests that choosing a smaller r may accelerate the training process. On the other hand, we can also observe that the curve corresponding to $r = 400$ is less smooth compared to others, which is a common situation when SGD is used.

V. Conclusions

In this paper, we have systematically investigated how to enhance the performance of federated learning (FL) in a heterogeneous computing environment by exploiting coded computing. Specifically, we first proposed CFL-HC, which is a new coded FL (CFL) framework that can accelerate the training of linear models. In this framework, we considered a computing system that consists of a central server and multiple computing devices with original or coded datasets, we developed a comprehensive learning scheme based on stochastic gradient descent, and we designed a novel encoding scheme that can overcome the limitations of existing CFL schemes. Within such a framework, we formulated an optimization problem to obtain the optimal deadline of each training round and the optimal size of the computing task allocated to each computing device, and also designed a two-step method to solve the optimization problem. Finally, to evaluate the proposed CFL-HC framework, we developed a real CFL system, based on which we conducted simulation experiments that demonstrate the outstanding performance of our framework compared to non-coding FL schemes.

Acknowledgement

We would like to thank National Science Foundation under Grants CI-1953048/1730675/1730570/1730325, CNS-1522458, CAREER-2048266, and CAREER-1714519, for the support of this work.

References

[1] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward UAV-Based Airborne Computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, dec 2019.

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," *arXiv*, oct 2016. [Online]. Available: http://arxiv.org/abs/1610.05492

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, jan 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3298981

[4] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, aug 2019. [Online]. Available: http://arxiv.org/abs/1908.07873

[5] H. H. Yang, A. Arafa, T. Q. S. Quek, and H. V. Poor, "Age-Based Scheduling Policy for Federated Learning in Mobile Edge Networks," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2020-May, pp. 8743–8747, oct 2019. [Online]. Available: http://arxiv.org/abs/1910.14648

[6] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," in *IEEE Transactions on Information Theory*, vol. 65, no. 7. Institute of Electrical and Electronics Engineers Inc., jul 2019, pp. 4227–4242.

[7] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "On Batch-Processing Based Coded Computing for Heterogeneous Distributed Computing Systems," *arXiv*, dec 2019. [Online]. Available: http://arxiv.org/abs/1912.12559

[8] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, "Coded federated learning," in *2019 IEEE Globecom Workshops, GC Wkshps 2019 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., dec 2019. [Online]. Available: https://arxiv.org/abs/2002.09574v2

[9] S. Prakash, S. Dhakal, M. Akdeniz, A. S. Avestimehr, and N. Himayat, "Coded Computing for Federated Learning at the Edge," *arXiv*, jul 2020. [Online]. Available: http://arxiv.org/abs/2007.03273