CMFL: Communication-Mitigated Federated Learning

Luping Wang, Wei Wang, Bo li
Hong Kong University of Science and Technology
{lwangbm, weiwa, bli}@cse.ust.hk

***Abstract*—Federated learning is recently proposed as a new machine learning setting where a global model is maintained by and shared with a large volume of edge devices (clients). By aggregating only the training *updates* to a central server while leaving the privacy-sensitive training data distributed local, federated learning provides effective protection to the clients' privacy. Considering the expensive and relatively slow network connections at client-side devices (e.g., mobile phones), improving the efficiency of the client-server communication is of great importance. However, existing works are limited to compress the number of transmitted bits in each update, while directly reducing the rounds of client-side updates has so far received little attention in the literature. We argue that by precluding some *insignificant* client-side updates from being uploaded, those unnecessary client-server communication cost could be effectively mitigated. Unfortunately, a recent design for geo-distributed learning, Gaia which follows the similar intuition, fails in effectively revealing updates' significance when there are hundreds of thousands of participating clients, making it unsuitable in the context of federated learning. In this paper, we propose a new communication strategy, Communication-Mitigated Federated Learning (CMFL), which is tailored to identify the significance of client-side updates in federated learning. CMFL provides *feedback* information regarding the global training tendency to clients, and each client uses this information as a reference to judge whether its update follows the *collaborative tendency*. By eliminating those updates whose optimization direction is far from the global trend, CMFL dramatically reduces the accumulated communication overhead while guaranteeing the learning convergence. We show that CMFL is a *generic* enough to improve the communication efficiency for all the existing federated learning designs. Both simulation and EC2 deployment confirm that CMFL improves the communication efficiency of basic federated learning by $13.97\times$ in terms of accumulated communication cost, outperforming the state-of-the-art Gaia by $11\times$. Moreover, when built under the follow-up Federated Multi-Task Learning, CMFL further enhances its communication efficiency by $5.7\times$ along with a $1.04\times$ improvement of the learning accuracy.**

I. Introduction

The past a few years have seen the tendency that the training process of machine learning is moving towards the mobile devices at the edge of the Internet [1], [2]. Unfortunately, conventional distributed machine learning systems [3]–[5] which are designed for highly controlled environments (e.g., data centers) do not suit in this context, due to the highly unbalanced and non-IID data distribution [6].

To this end, federated learning [6]–[11] is proposed as an alternative machine learning setting towards the large volume of edge devices (clients) that participate in the model training. In federated learning, a global model is maintained in a trusted central server. By aggregating the client-side *training updates* while leaving the sample data distributed local, the global model is optimized without exposing the privacy-sensitive data [12]. The resulting model is then distributed back to all clients, eventually converging to a jointly representative model.

A successful demo implementation of federated learning in the industry is Gboard [13], the Google Keyboard on Android, which provides a query suggestion according to user's input context, while each user's click history is, in turn, served as the training data to improve the suggestion model [14]. The cloud-side global suggestion model is collaboratively trained by hundreds of thousands of Gboard users (clients) without uploading their local click information.

In order to reap the benefits from practical federated learning, the scarce network connection poses a particular challenge. On the one hand, in federated learning, the targeted clients are edge devices such as mobile phones and tablets, each with unreliable, expensive, and asymmetric network connections. On the other hand, advanced machine learning services like Gboard's query suggestion employ increasingly complicated neural networks, dramatically adding the amount of transferred data within each local update. Therefore, specific strategies towards improving the client-server communication efficiency are of utmost importance.

There are two fundamental ways to ease the communication burden during the model updating (1) compressing the size of transferred data within each client-side update and (2) reducing the frequency of model updating for each client. Unfortunately, the majority of existing works are restricted to the first direction, i.e., data compression. For example, structured updates [6] is proposed to employ specific data structures to compress the number of bits uploaded in each client-side update. However, these compressions may hurt the learning accuracy and thus come without convergence guarantees.

In this paper, we challenge the status quo with a bold question: *is it possible to enhance the communication efficiency by reducing the communication rounds — the number of client-side updates — without hurting the learning accuracy?* We provide an affirmative answer to this question. In specific, we propose to mitigate the underlying client-server communication by dynamically identifying and eliminating those *insignificant* communications from being uploaded. Intuitively, this simple approach is both *efficient* and *general*. First, as federated learning is proposed towards the edge devices, personalized training data is non-negligible. Given the non-IID nature of the client-side training data, some local updates are just *outliers* which modify the global model in a *tangential direction* to the collaborative convergence. Eliminating these local optimizations from uploading hurts little of the learning accuracy, while potentially reducing the unnecessary communication. Second, this insight can *generally* support all the follow-up federated learning designs and further enhance their communication efficiency, as long as their model training is based on aggregating the client-side optimizations.

However, directly implementing existing approach Gaia [15] which has a similar intuition is problematic. Specifically, Gaia is proposed as a geo-distributed machine learning framework that identifies the significance of the local optimization by judging whether the update's *absolute value* is large enough. Those *slight* updates with small absolute value will be precluded from uploading. We argue that this simple method ignores the *federation* of a large volume of participating clients, losing the opportunity to *correctly* identify the updates' significance under federated learning. Specifically, by comparing the update's absolute value with a fixed threshold, Gaia is unaware of whether this update follows the collaborative optimization trend among all the clients. This is even fatal when there are hundreds of thousands of edge devices. As we will show in Sec. V, directly implementing Gaia provides an unsatisfying improvement of communication efficiency in federated learning.

We address this challenge through a novel algorithm called Communication-Mitigated Federated Learning (CMFL). Our key insight is to identify the significance of client-side updates by adding the *feedback* information of the *global training tendency* to each client as a reference to judge whether its update follows the joint optimization. Particularly, in each learning iteration, a client first gets the feedback information about the *global update* from the central server. Next, the client proceeds its local training and calculated its *local update*. The client then compares its local update with the global update, checking whether its local optimization follows a collaborative convergence tendency. To do this, for each local update, CMFL calculates the percentage of parameters in the local update whose signs (positive/negative) are *opposite to* the corresponding value in the global update. Intuitively, the *higher* this percentage is, the *more tangential* direction to the collaborative convergence this client-side update will have, and thus the more *insignificant* this update will be. By eliminating those local updates with the percentage higher than a threshold, CMFL can effectively mitigate those unnecessary communication load.

We evaluated CMFL against basic federated learning [7], Gaia [15] and the follow-up advances [16] through both simulations and EC2 deployment on a 30-machine cluster. Evaluation results show that CMFL outperforms the basic federated learning by $13.97\times$ in terms of communication cost, whereas the alternative Gaia only improves it by $1.26\times$. Moreover, for the advanced federated multi-task learning [16], CMFL further enhances the communication efficiency by $5.7\times$, while also increasing the learning accuracy by $1.04\times$.

II. Model and Background

In this section, we describe our models for federated learning along with our objectives, i.e., minimizing the communication rounds while still guaranteeing the learning convergence. We also survey the background information of existing works and motivate the need for a new communication mechanism to eliminate the uploading of insignificant local updates.

A. System Model

Synchronous update scheme. Following the previous works [6], [7], [17], We assume a synchronous update scheme that proceeds in each training iteration:

1) Clients independently train the updated models based on their local training data (e.g., from the sensor information on device).
2) All the clients upload their local optimization to the central server for aggregation.
3) The central server aggregates received local models (typically by averaging) to construct an improved global model and then distributes the resulting model back to the clients for the next iteration.

Protection of privacy. Instead of uploading the raw training data directly to a centralized server, the participating clients in federated learning transfer only the model updates which could be ephemeral [1], [7], [18]–[20]. This anonymous update reveals little information about the source client. Besides, as the global optimization in the central server requires no information of the meta data about the update's source, the communication can be executed without identifying the personal information or through a third party.

Model and Notations. We formulate the learning problem as an optimization problem where the model $\mathbf{x} \in \mathbb{R}^l$ is trained to minimize the *average value* of the loss function $f(\mathbf{x})$:

$$\min_{\mathbf{x}\in\mathbb{R}^l} f(\mathbf{x}), \quad where \quad f(\mathbf{x}) = \frac{1}{D}\sum_{k=1}^{D} f_k(\mathbf{x}), \tag{1}$$

where $f_k(\mathbf{x})$ is the loss value within the k^{th} client. In particular, there is a fixed set of D clients: $\mathbb{C} = \langle c_1, \ldots, c_D\rangle$, each with a fixed local dataset $\mathscr{P}_k$. A global model $\mathbf{x}$ is shared with and updated by all the D clients.

As a common practice, we build the federated optimization in Eq. (1) through Stochastic Gradient Descent (SGD) [21], where the batch gradient is calculated in each training iteration and the global optimization is approached through iterative steps. Formally, in the t^{th} iteration of the synchronous SGD algorithm, the global model $\mathbf{x}_t$ is obtained by:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \sum_{k=1}^{D} \eta_k \nabla f_k(\mathbf{x}_t) = \mathbf{x}_{t-1} + \sum_{k=1}^{D} \mathbf{u}_{k,t}, \tag{2}$$

where $\nabla f_k(\mathbf{x})$ and η_k represent the gradient function and the learning rate of the k^{th} client, respectively. We also define the local update of the k^{th} client in the t^{th} iteration as $\mathbf{u}_{k,t} = -\eta_k \nabla f_k(\mathbf{x}_t)$. Accordingly, the learning process can be represented as a sequence of optimizations of the global model $\mathbf{x}_T$, i.e.,

$$\mathbf{x}_T = \mathbf{x}_0 + \sum_{t=1}^{T}\sum_{k=1}^{D} \mathbf{u}_{k,t}. \tag{3}$$

B. Objectives

Minimizing the accumulated communication rounds. In practical federated learning where the typical clients are edge devices such as mobile phones and tablets with scarce network connections, the constraint of communication between the central server and clients naturally arises. For example, the client may wish to reduce the amount of uploaded data from his mobile phone to minimize the charging from mobile operators.

Instead of reducing the number of transmitted bits within each update in [6], our first objective is to minimize the number of uploaded client-side updates, i.e., the *accumulated communication rounds*.

Definition 1 (Accumulated Communication Rounds)**.** *In the t^{th} iteration of the synchronous SGD algorithm, let $\mathbb{S}_t$ be set of clients whose local updates are uploaded to the central server. We define the communication round in the t^{th} iteration as $r_t = |\mathbb{S}_t|$, i.e., the number of clients in $\mathbb{S}_t$. Given a targeted training accuracy $\mathscr{A}$ and the corresponding learning iterations T, the accumulated communication rounds is defined as*

$$\sum_{t=1}^{T} r_t = \sum_{t=1}^{T} |\mathbb{S}_t|. \tag{4}$$

Therefore, our first objective is to minimize the accumulated communication rounds shown in Eq. (4).

Guaranteeing the learning convergence. The improved communication efficiency should not sacrifice the learning convergence. In specific, we use $\mathbf{x}^*$ to represent the *optimal* global model which the parameters are trained to approach. Proving the convergence of a machine learning algorithm equals to guaranteeing the following condition:

$$\lim_{T\to\infty} \frac{1}{T} R[\mathbf{x}] = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} |f(\mathbf{x}_t) - f(\mathbf{x}^*)| = 0, \tag{5}$$

where $R[\mathbf{x}]$ is the regret function.

To summarize, our objective is to *minimize the accumulated communication rounds while guaranteeing the convergence of the machine learning algorithm*, i.e.,

$$\begin{aligned} &\text{minimize} \quad \sum_{t=1}^{T} r_t, \\ &\text{s.t.} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} |f(\mathbf{x}_t) - f(\mathbf{x}^*)| = 0. \end{aligned} \tag{6}$$

C. Related Works

Advanced designs of federated learning. Despite the rich literature on federated learning, a large body of works are mainly restricted to designing sophisticated machine learning methods. For example, MOCHA [16] is proposed to employ the multi-task learning (MTL) framework to capture the distributed nature of federated learning, where the goal is to consider fitting separate but related models simultaneously. Formally, MOCHA captures the relationship among all the clients (tasks) through their relationship matrix. Besides, federated meta-learning [22] is another recent advance, where the user's information is shared at the level of algorithms, instead of model or data adopted in previous approaches. Although these learning frameworks can speed up the training process, they achieve *little* in increasing the communication efficiency.

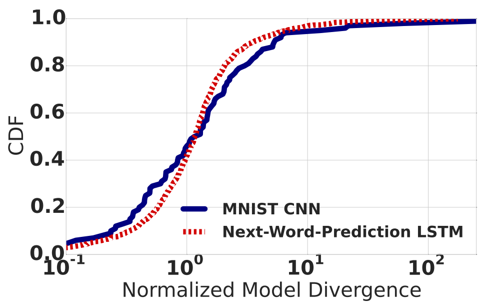


Fig. 1: Distribution of the Normalized Model Divergence (d_j) for all the trained parameters $x_j \in \mathbf{x}$.

There are also other works implementing complex data structures to compress the amount of data communicated, e.g., structured updates and sketched updates [6]. Unfortunately, these works come without convergence guarantees while adding the computation complexity during the communication stages. To our best knowledge, all of these works above ignore the potential benefit from mitigating the unnecessary communication rounds.

Prior art in geo-distributed learning. Gaia [15] is recently proposed as a communication framework that reduces the across-datacenter communication for geo-distributed machine learning. Specifically, Gaia identifies the significance of the local optimization from one datacenter by its update's magnitude relative to the current parameter value (i.e., $||\frac{\text{Update}}{\text{Model}}||$), where $||.||$ is the Euclidean norm of a vector. Any update with $||\frac{\text{Update}}{\text{Model}}|| <$ Threshold will be precluded from uploading. We notice that this identification is made based only on the local update itself, irrelevant to the federation with all the clients. In the context of federated learning where there are hundreds of thousands of clients each with non-IID training data, this *open-loop* method without a *feedback* of the global tendency fails in efficiently identifying the update's significance.

III. Intuition and Challenges

In this section, we present our intuition of reducing the communication overhead by eliminating insignificant local updates. We also discuss the challenges to implement this intuition in federated learning.

A. Intuition

We begin our discussion by analyzing why some local optimizations are *useless* to the global convergence.

The Divergence between global and client-side models. As the global model in federated learning is the aggregation (typically by averaging) of a huge number of client-side local models, the divergence of these two parts of models generally exists, especially when considering the *personalized* local training datasets. Specifically, given the non-IID nature of the clients' local training data, some of the updates may not be representative of the population distribution. For example, when training a query suggestion model for Google's Gboard [14], various clients can show a diversity of clicking choices. While some of these choices follow a common preference, others may also produce *personalized* local updates which are *tangential* to the collaborative trend of the training convergence. To show the model divergence, we measure the difference between the global and local parameters by a metric called *Normalized Model Divergence*. Normalized Model Divergence is defined, for each trained parameter $x_j \in \mathbf{x}$, as the *average* difference between the client-side value and the corresponding global value normalized by the global value, i.e.,

$$d_j = \frac{1}{D}\sum_{k=1}^{D}|\frac{x_{j,k}-\bar{x}_j}{\bar{x}_j}|, \tag{7}$$

where $x_{j,k}$ is the local value of the parameter x_j in client c_k, and $\bar{x}_j$ is the global value of x_j. Intuitively, the larger d_j is, the more difference the global and client-side models with have regarding the trained parameter x_i.

In order to illustrate the difference between the global and client-side models, we follow the previous work [23] to train the following two models: MNIST CNN [24], [25] and Next-Word-Prediction LSTM [26]–[28], each with the dataset distributed to 100 clients. We defer the detailed description of these two training models and datasets to Sec. V.

We extract the Normalized Model Divergences in these two models and plot their distributions in Fig. 1. As we can see, more than 50% parameters in both models produce the model divergence higher than 100%, suggesting that there generally exists an obvious difference between the global and local models. Moreover, the maximum parameter divergences in the two models reach up to 268 and 175, respectively. We attribute this obvious model divergence to the personalized training data.

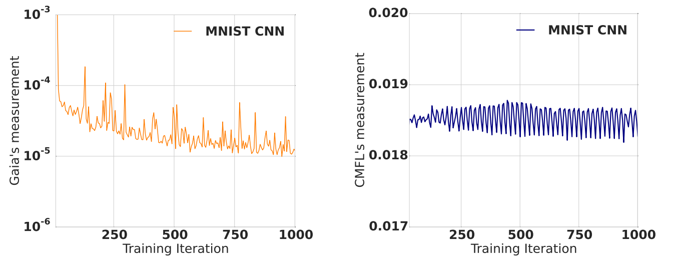
To summarize, there are a non-negligible number of personalized local optimizations which are tangential to the collaborative training convergence. Uploading these outliers to the central server *contributes little* and even *does harm* to the convergence of the global model.

Intuition. Our discussion above reveals the potential opportunity to enhance the communication efficiency in practical federated learning by precluding some outliers from being uploaded from local clients. In light of this observation, our intuition is to *dynamically identify the significance of the client-side updates based on whether these updates follow the collaborative convergence trend or are just outliers with personalized optimization, followed by eliminating those insignificant updates from being communicated.*

B. Challenges

However, directly implementing the intuition above is challenging. We identify two key challenges in designing such a communication mechanism for federated learning.

Guaranteeing the learning convergence. The enhanced underlying communication strategy should guarantee the training process to converge *timely*. Intuitively, reducing the amount of data to be transferred means losing some of the training information, which will inevitably disturb the convergence of



(a) Gaia's identification measurement $||\frac{\text{Update}}{\text{Model}}||$, y-axis is in log scale (b) CMFL's identification measurement $e_t^k = n_t^k/N$

Fig. 2: Illustration of Gaia's failing in identifying updates' significance using MNIST CNN with 168 clients. (a) Gaia's measurement decreases exponentially with the training approaching to convergence, y-axis is in log scale. (b) CMFL's measurement keeps stable.

the global optimization. Therefore, the expected communication strategy should not eliminate arbitrary local optimization from being uploaded and come with convergence guarantee. We note that although some of the follow-up federated learning designs [6] can efficiently reduce the communication overhead, they provide *no* convergence guarantee.

Hardness in capturing the updates' significance. As we have discussed in Sec. II-C, Gaia [15] provides an efficient paradigm in identifying and eliminating the insignificant local update in geo-distributed learning, i.e., and any update with $(||\frac{\text{Update}}{\text{Model}}|| < \text{Threshold})$ will be precluded from uploading.

Directly judging the update's magnitude is an efficient measurement in geo-distributed machine learning, since there are only *a few* datacenters each with heavy communication load. However, due to the massive volume of participating clients and the non-IID and unbalanced nature of the client-side training data, this method fails in identifying the significance of updates in federated learning for the following three reasons:

- First, as there will be hundreds of thousands of clients collaboratively training the global model, some with much heavier local training workload than others, optimizing much more parameters with the larger value of $||\frac{\text{Update}}{\text{Model}}||$. Besides, the update's magnitude also relies on the learning rate, with larger learning rate bringing more obvious optimization. Therefore, it is hard to set a *global threshold* to identify how much of the data size in an update should be regarded as significant.
- Second, as the global model is the federation of a massive amount of clients, the significance of the local updates cannot be represented by their absolute value. Consider a condition where a large volume of clients providing the similar updates but with small absolute values. These updates are *slight but non-negligible*, because their federation forms out a global optimization tendency. In contrast, an outlier with large absolute value will eventually be submerged by other massive updates.
- Third, as the updates' magnitude will *exponentially*

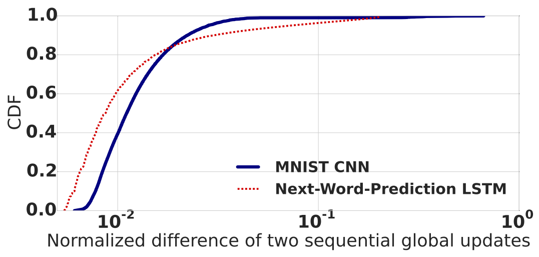


Fig. 3: Distribution of ΔUpdate, x-axis is in log scale.

decrease with the learning approaching convergence, a *fixed* threshold cannot satisfy the identification during the entire learning procedure. In order to illustrate this point, we follow the previous work [7] to train a model of MNIST CNN in federated learning with 168 clients and plot the average value of $||\frac{\text{Update}}{\text{Model}}||$ for all the clients in each learning iteration, as shown in Fig. 2a. As we can see, with the learning iteration raises, this value reduces exponentially. On the one hand, if we set the threshold with a 'large' value, e.g., 5×10^{-5}, almost all the updates later than the 300th learning iteration will be identified as *insignificant*, being eliminated from uploading. Consequently, the global optimization will *stagnate* after the 300th learning iteration, preventing the training from convergence. On the other hand, with a 'small' threshold, e.g., 1×10^{-5}, almost all the updates will be uploaded, losing the opportunity to reduce communication overhead before this watershed. Although we can delicately set this threshold as a time variable which decreases over time, the tuning highly depends on the workload, making it hard to implement in practical federated learning.

Accordingly, this *open-loop* method without a *perception* of the collaborative optimization tendency fails in identifying the update's significance in federated learning. As we will show in Sec. V, directly implementing Gaia only *slightly* improves the communication efficiency in federated learning.

IV. CMFL

In this section, we first illustrate our key insight to perceive the global optimization tendency, which is to obtain a feedback of the global update and use the sign of model parameters to indicate the consistency of the global and client-side updates. Based on this insight, we then present a new communication mechanism, called *Communication-Mitigated Federated Learning (CMFL)*, efficiently eliminating the insignificant local update from being uploaded. We further show that our proposed CMFL provides *provable* convergence guarantee.

A. Key Insight

Feedback of the global update. Our discussion above reveals that the key to eliminating some useless local updates is to be aware of the collaborative optimization tendency in the global aggregation. Ideally, in each learning iteration, clients are expected to obtain the collaboratively-aggregated update *in advance*, based on which they can identify the local updates' significance before uploading. To this end, an efficient *feedback* method is required.

Slight difference between sequential global updates. However, the collaborative optimization cannot be revealed until the finish of local uploading and global aggregation in the current iteration. We argue the global update in the *previous iteration* can be used to *approximately* represent the corresponding value in the current iteration. Intuitively, given the training process approaches the convergence *stably and gradually*, the difference between two sequential global updates should not be arbitrarily large. In order to quantify this difference, we use the metric of ΔUpdate to denote the differential degree of two sequential global updates. Formally, ΔUpdate is defined, for each two sequential iterations, as the difference of the two global updates normalized by the former one, i.e.,

$$\Delta\text{Update}_i = \frac{||\text{Update}_{i+1} - \text{Update}_i||}{||\text{Update}_i||}, \tag{8}$$

where $||.||$ is the Euclidean norm of a vector.

Intuitively, the larger ΔUpdate_i is, the higher the differential degree of the two sequential global updates will have. We summarize the ΔUpdate during the training of the MNIST CNN and Next-Word-Prediction models in Fig. 1 and plot its distribution, as shown in Fig. 3. We can see that in the MNIST CNN model, more than 99% of the global updates have ΔUpdate lower than 0.05, while the maximum ΔUpdate is less than 0.67. For the Next-Word-Prediction model, in more than 93% of the iterations, the ΔUpdate is lower than 0.05, and the maximum value is only 0.21. Therefore, using the global update in the previous iteration will only bring *slight* difference. We also note that the previous global updates even do not require additional communication, because the clients can maintain the previous global model by themselves.

Parameter's sign as an indication. After getting the feedback of the global update, the next step is to choose an appropriate measurement to identify local updates' significance based on this feedback information. Our previous analysis of Gaia [15] has revealed that directly computing the update's magnitude is ineffective. We argue that *the proportion of the trained parameters whose client-side update have the same sign (positive/negative) with the global update* can be used to efficiently represent the local updates' significance.

Intuitively, the absolute value of an update decides only the speed of the training convergence, while the sign of updates represent the *direction* of the training, deciding the convergence point of the optimization which is much more essential. Therefore, comparing the parameters' sign can effectively measure the consistency of the two training updates. Besides, the disturbance of the additional factors (e.g., the learning rate) which are crucial in calculating the absolute value of updates will be definitely avoided when comparing the sign instead.

Consider an extreme condition where there is a local update with all its parameters having the *opposite* sign to the global update. This local optimization absolutely runs counter to the collaborative direction, and precluding it from being uploaded will not hurt the global convergence while reducing the communication overhead.

Key Insight. Following this observation, our key insight is to *dynamically identify and eliminate the insignificant client-side optimization based on comparing the sign of parameters between the client-side and global updates.*

Formally, in the t_{th} learning iteration, for client c_k, we define the number of its trained parameters whose updates have the *same* sign (positive or negative) with the global update as n_t^k. We also define the total number of parameters in the model as N. We then identify the significance of the update of client c_k in the t_{th} iteration by the following ratio:

$$e_t^k = n_t^k / N. \tag{9}$$

Any update with $e_t^k < \text{Threshold}$ will be identified as *insignificant*, and thus be eliminated from uploading. Compared with prior work Gaia [15], this simple approach can potentially provide the following two benefits:

1) *It effectively reveals whether the client-side updates follow the collaborative direction or are just outliers.* Unlike Gaia, comparing the sign of parameters *directly* shows the *consistency of optimization direction*, regardless of the influence of the learning rate and the size of the local dataset. Therefore, it avoids the misidentification of both the common but slight updates and the outliers with large absolute value. This identification is much more suitable to the nature of federated learning, i.e., the highly unbalanced and non-IID datasets are distributed in a large volume of clients.
2) *This measurement keeps stable during the learning procedure, being easy to set a global threshold.* In contrast to Gaia, the average value of e_t^k during the training process keeps stable all the time, as shown in Fig. 2b. Therefore, we can easily set a global threshold to identify the updates' significance.

B. Communication-Mitigated Federated Learning

Based on the insight above, we next illustrate our proposed new communication mechanism, Communication-Mitigated Federated Learning (CMFL).

Aggregation at the central server. As a starting point, the global aggregation at the central server is similar to the basic federated learning [6], [7]. Specifically, at the beginning of the t_{th} learning iteration, a global model $\mathbf{x}_{t-1}$ is first distributed back to all the clients as a basic model to optimize, along with the feedback information of the global update $\bar{\mathbf{u}}_{t-1}$. After obtaining the local optimization and update's significance from each client in a parallel fashion, the central server gathers those significant updates for aggregation, typically by averaging. We summarize the global aggregation process as a procedure, called `GlobalOptimization`, as shown in Algorithm 1.

Local optimization at the client-side devices. The client-side optimization is executed by the local training based on the local training data. Before uploading the local update $\mathbf{u}_{k,t}$,

Algorithm 1 CMFL

```
 1: procedure GLOBALOPTIMIZATION
 2:     Input: Client set $\mathbb{C} = \langle c_1, \ldots, c_D \rangle$
 3:     Initialize the global model $\mathbf{x}_0$ and the global update $\bar{\mathbf{u}}_0$
 4:     for each iteration $t = 1, 2, \ldots$ do
 5:         for all client $c_k \in \mathbb{C}$ do in parallel
 6:             $(s_{k,t}, u_{k,t}) \leftarrow$ LocalUpdate $(k, \mathbf{x}_{t-1}, \bar{\mathbf{u}}_{t-1})$
 7:         $\mathbb{S}^t \leftarrow \{u_{k,t} | \quad s_{k,t}$ is True$\}$        ▷ Significant updates
 8:         $\bar{\mathbf{u}}_t \leftarrow \frac{1}{|\mathbb{S}^t|} \sum_{u_{k,t} \in \mathbb{S}^t} u_{k,t}$        ▷ global update
 9:         $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \bar{\mathbf{u}}_t$
10: procedure LOCALUPDATE
11:     Input: Client index $k$, Model $\mathbf{x}_{t-1}$ and Update $\bar{\mathbf{u}}_{t-1}$
12:     Do the local training and obtain the local update $\mathbf{u}_{k,t}$
13:     $s_{k,t} \leftarrow$ CheckSignificance $(\bar{\mathbf{u}}_{t-1}, \mathbf{u}_{k,t})$
14:     if $s_k$ is False then
15:         $\mathbf{u}_k \leftarrow$ NULL        ▷ eliminate the insignificant update
16:     return $(s_{k,t}, u_{k,t})$
17: procedure CHECKSIGNIFICANCE
18:     Input: Global update $\bar{\mathbf{u}}_{t-1}$ and Client-side update $\mathbf{u}_{k,t}$
19:     Calculate the parameter ratio $e_t^k$, as shown in Eq. (9)
20:     if $e_t^k < v(t)$ then
21:         $s_{k,t} \leftarrow$ True
22:     else
23:         $s_{k,t} \leftarrow$ False        ▷ identify the insignificant update
24:     return $s_{k,t}$
```

CMFL checks its significance by calculating e_t^k, the ratio of the trained parameters whose updates have the same sign with the global update, as shown in Eq. (9). Any local optimization with e_t^k larger than a tuned threshold $v(t)$ will be identified as *insignificant*, and thus be eliminated from uploading. We also summarize the local optimization and significance identification process in Algorithm 1 as two procedures, called `LocalUpdate` and `CheckSignificance`, respectively.

Generally support existing designs. Although our algorithm above is presented based on the basic federated learning algorithm [6], [7], CMFL can be easily *generalized* to support the follow-up federated learning designs, as long as the global optimization is the aggregation of the local updates. For example, MOCHA [16] employs the multi-task learning (MTL) where distributed clients independently train their own learning models instead of sharing a global model. Our proposed CMFL can also support MOCHA by locally calculating the changing of the global matrix based on the local update and the record of the relationship matrix among clients. As we will show in Sec. V, CMFL can not only reduce the communication cost for MOCHA but also slightly improve its learning accuracy.

C. Convergence Guarantee

Our objective is to minimize the accumulated communication rounds while guaranteeing the learning convergence, as shown in Eq. (6). However, minimizing the required communication rounds is essentially an open problem, given the non-IID distribution of the client-side training data. Therefore, we only do the theoretical analysis bout the convergence guarantee and use our simulation and real-world testbed to justify our reduced communication overhead. We next present that CMFL provides convergence guarantee under offline case with some reasonable assumptions.

Assumptions. In order to provide the convergence guarantee for CMFL, we make two assumptions in the following discussion: First, we assume that the loss function $f(\mathbf{x})$ is a *convex function*, which is a common practice: $f(\mathbf{x}) + f(\mathbf{y})/2 \geq f((\mathbf{x}+\mathbf{y})/2)$. Second, we assume that the global update in the *previous iteration* can be used to approximately represent the corresponding value in the current iteration, as discussed in Sec. IV-A.

Convergence Guarantee. Eq. (5) has defined the requirement of the convergence in a machine learning algorithm. In order to represent the identification of updates' significance, we further differentiate two kinds of global models in the i_{th} iteration: $\mathbf{x}_t$ refers to the model optimized by all the local updates regardless their significance, whereas $\tilde{\mathbf{x}}_t$ represents the actual global model in CMFL, eliminating those insignificant updates. Formally, we have the following theorem:

Theorem 1 (Convergence guarantee). *Let η_t and v_t be the learning rate and significance threshold at step t, respectively. For client set $\mathbb{C} = \langle c_1, \ldots, c_D \rangle$, its summarized loss function $R[\tilde{\boldsymbol{x}}]$ in Algorithm 1 can be bounded as follows:*

$$\lim_{T\to\infty} \frac{1}{T} R[\tilde{\boldsymbol{x}}] = \frac{1}{T}[\mathcal{O}(\sum_{t=1}^{T} \eta_t) + \mathcal{O}(\frac{1}{\eta_T}) + \mathcal{O}(\sum_{t=1}^{T} v_t)], \quad (10)$$

where $\mathcal{O}(.)$ is the big O notation that used in asymptotic analysis.

We make two remarks on Theorem 1:

1) *The convergence of Algorithm 1 relies on the setting of the learning rate η_t and significance threshold v_t.* In order to guarantee the learning process finish in a *timely* manner, these two parameters should be set as time-deceased variables that make $\lim_{T\to\infty} \frac{1}{T} R[\tilde{\mathbf{x}}] \to 0$.
2) *There are diverse η_t and v_t we can choose to guarantee the convergence of the algorithm, each with different convergence speed.* For example, if we set $\eta_t = \eta_0/\sqrt{t}$ and $v_t = v_0/\sqrt{t}$, we have $\lim_{T\to\infty} \frac{1}{T} R[\tilde{\mathbf{x}}] \to \lim_{T\to\infty} \mathcal{O}(\sqrt{T}/T) \to 0$.

We next give a proof sketch of Theorem 1. The complete proof is deferred to our technical report [29] due to space constraints.

Proof Sketch. We start by considering the convex property of the loss function $f(\mathbf{x})$. We note that the convex function $f(\mathbf{x})$ has the following property: $f(\mathbf{x}) - f(\mathbf{y}) \leq \langle (x-y), \nabla f(\mathbf{x}) \rangle$, where $\langle , \rangle$ represents the inner product of two vectors. Accordingly, we have $\sum_{t=1}^{T} f(\tilde{\mathbf{x}}_t) - f(\mathbf{x}^*) \leq \sum_{t=1}^{T} \langle (\tilde{\mathbf{x}}_t - \mathbf{x}^*), \nabla f(\tilde{\mathbf{x}}_t) \rangle$. The latter part of this inequality can be further transformed into three parts. The first two parts come together to represent the difference between the optimal model $\mathbf{x}^*$ and the federated model including all the updates regardless of their significance $\mathbf{x}$, while the third part shows the additional loss between $\mathbf{x}$ and $\tilde{\mathbf{x}}_t$, due to the elimination of the insignificant updates. We execute the asymptotic analysis of these three parts independently and obtain the bound of the summarized loss function as shown in Eq. (10).

Proof. As a common practice, we assume that the loss function $f(\mathbf{x})$ is a convex function: $\frac{f(\mathbf{x})+f(\mathbf{y})}{2} \geq f(\frac{\mathbf{x}+\mathbf{y}}{2})$. According to this definition, we could further get the following attribute of function $f(\mathbf{x})$, which will be used in the later part of the proof. Specifically, the delta value of a convex function in to points is equal or larger than the distance between these two points multiplied by the gradient at the left point ($\nabla f(\mathbf{x})$), i.e.,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle (y-x), \nabla f(\mathbf{x}) \rangle, \quad (11)$$

where $\langle , \rangle$ represents the inner product of two vectors. Accordingly, we further transform this inequation and have:

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \langle (x-y), \nabla f(\mathbf{x}) \rangle. \quad (12)$$

Recall that our objective is to prove the limitation of the accumulated loss function can be bounded when the number of iterations approaches to infinite, i.e.,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} f(\mathbf{x}_t) - f(\mathbf{x}^*) = 0. \quad (13)$$

Notice that we use $\tilde{\mathbf{x}}_t$ to represent the real model which precludes those insignificant updates. According to Eq. (12), we deduct the Regret function shown in Eq. (13) as:

$$\sum_{t=1}^{T} f(\tilde{\mathbf{x}}_t) - f(\mathbf{x}^*) \leq \sum_{t=1}^{T} \langle (\tilde{\mathbf{x}}_t - \mathbf{x}^*), \nabla f(\tilde{\mathbf{x}}_t) \rangle. \quad (14)$$

Intuitively, we transfer the Regret function $\sum_{t=1}^{T} f(\tilde{\mathbf{x}}_t) - f(\mathbf{x}^*)$ to the product of the model difference and the gradient at the current global model $\tilde{\mathbf{x}}_t$.

As the optimal model $\mathbf{x}^*$ is unknowable, Eq. (14) could not be further deducted directly. To address this, we separate the sum of $\sum_{t=1}^{T} \langle (\tilde{\mathbf{x}}_t - \mathbf{x}^*), \nabla f(\tilde{\mathbf{x}}_t) \rangle$ into three parts, with the first one and third on irrelevant to the optimal model $\mathbf{x}^*$, i.e.,

$$\begin{aligned} &\sum_{t=1}^{T} \langle (\tilde{\mathbf{x}}_t - \mathbf{x}^*), \nabla f(\tilde{\mathbf{x}}_t) \rangle \\ &\quad = \sum \frac{1}{2} \eta_t ||\nabla f(\tilde{\mathbf{x}}_t)||^2 \\ &\quad + \sum \frac{||\mathbf{x}^* - \tilde{\mathbf{x}}_t||^2 - ||\mathbf{x}^* - \tilde{\mathbf{x}}_{t+1}||^2}{2\eta_t} \\ &\quad + \sum \langle (\tilde{\mathbf{x}}_t - \mathbf{x}_t), \nabla f(\tilde{\mathbf{x}}_t) \rangle. \end{aligned} \quad (15)$$

We next illustrate these three parts one by one, showing their convergence property. The first part will be bounded easily, since $||\nabla f(\tilde{\mathbf{x}}_t)||^2$ is bounded by its maximum value, and thus we have:

$$\sum \frac{1}{2} \eta_t ||\nabla f(\tilde{\mathbf{x}}_t)||^2 \leq \sum \frac{1}{2} \eta_t ||\nabla f(\tilde{\mathbf{x}}_t)||^2_{\max} = \mathcal{O}(\sum_{t=1}^{T} \eta_t). \quad (16)$$

Recall that we use the notation of $\mathcal{O}(.)$ to do the asymptotic analysis. As for the second part, we have:

$$\begin{aligned} &\sum \frac{||\mathbf{x}^* - \tilde{\mathbf{x}}_t||^2 - ||\mathbf{x}^* - \tilde{\mathbf{x}}_{t+1}||^2}{2\eta_t} \\ &= \mathcal{O}(1) + \sum ||\mathbf{x}^* - \tilde{\mathbf{x}}_t||^2 (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) \\ &\leq \mathcal{O}(1) + ||\mathbf{x}^* - \tilde{\mathbf{x}}_t||^2_{\max} \sum (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) \\ &= \mathcal{O}(\frac{1}{\eta_T}). \end{aligned} \tag{17}$$

The third part represents the sum of the update loss, due to the elimination of those insignificant update. We use I_t to define the set of insignificant updates at step t, and have:

$$\sum \langle (\tilde{\mathbf{x}}_t - \mathbf{x}_t), \nabla f(\tilde{\mathbf{x}}_t) \rangle = \langle \sum_{u_t \in I_t} u_t, \nabla f(\tilde{\mathbf{x}}_t) \rangle. \tag{18}$$

As there are at most $v_t D$ clients providing insignificant updates, and the update value could be bounded by its maximum value, we further have:

$$\begin{aligned} \sum \langle (\tilde{\mathbf{x}}_t - \mathbf{x}_t), \nabla f(\tilde{\mathbf{x}}_t) \rangle &= \langle \sum_{u_t \in I_t} u_t, \nabla f(\tilde{\mathbf{x}}_t) \rangle \\ &\leq D ||\mathbf{u}_t||^2_{\max} \sum v_t = \mathcal{O}(\sum_{t=1}^{T} v_t). \end{aligned} \tag{19}$$

Combining Eq. (16) and Eq. (19), we draw the conclusion as:

$$\lim_{T \to \infty} \frac{1}{T} R[\tilde{\mathbf{x}}] = \frac{1}{T} [\mathcal{O}(\sum_{t=1}^{T} \eta_t) + \mathcal{O}(\frac{1}{\eta_T}) + \mathcal{O}(\sum_{t=1}^{T} v_t)]. \tag{20}$$

Therefore, there are diverse η_t and v_t we could choose to guarantee the convergence of the algorithm, as shown in Theorem 1.

□

V. Evaluation

We evaluated CMFL through both simulations and real-world implementations deployed with a 30-machine Amazon EC2 cluster. We highlight the following three points here:

- CMFL significantly reduces the accumulated communication rounds of basic federated learning. Specifically, with CMFL, the communication efficiency is enhanced by $13.97\times$ in terms of accumulated communication rounds. CMFL also remarkably outperforms the state-of-the-art Gaia by $11\times$ in terms of the Speedup.
- CMFL *generally* supports existing follow-up designs of federated learning, further reducing their communication cost. In particular, when built under the recently proposed Federate Multi-Task Learning, CMFL can not only increase the communication efficiency by $5.7\times$ but also increase the learning accuracy by $1.04\times$.
- CMFL can be easily implemented in practice, with *slight* additional computation overhead. In specific, checking the updates' significance takes less than 0.13% time of the local training iteration.

A. Simulation of Basic Federated Learning

We start by comparing the communication overhead of CMFL with basic federated learning [7] and Gaia [15].

Workload. In order to provide comparability with existing works, we set up our first simulation using the similar training models and datasets in [7]. In particular, we choose two machine learning models to present the communication improvement provided by CMFL:

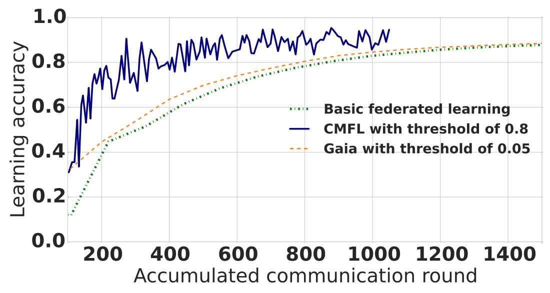
1) *MNIST digit recognition model using CNN.* The training model consists of a CNN with two 5×5 convolution layers, a fully connected layer, and a final output layer [25]. The dataset contains $60,000$ samples of handwritten digits [**?**]. We sort these samples by their digit labels and then divide them into 100 clients each receiving 600 examples, developing a non-IID data distribution.
2) *Next-Word-Prediction model using LSTM.* We train a 2-layer LSTM language model (each with 256 nodes) at the word-level, which after reading each word in a line, predicts the next word [27]. Specifically, we develop the training dataset from *The Complete Works of William Shakespeare* [30]. The input is a 10-word sequence in the dialogue, and the output is the predicted next word. We construct the local dataset of each client with the dialogue of a speaking role in the plays with at least 20 words. This produced a dataset with 100 clients. Totally, there are 1675 vocabularies and 6630 training samples.

Baseline algorithms. We compare CMFL against two baselines: Basic federated learning [7] and Gaia [15].

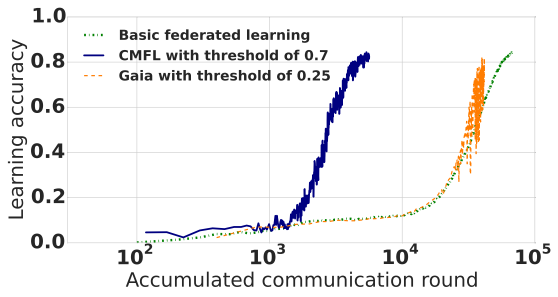
Setup. In our simulations, the model architectures were built upon TensorFlow [31]. We use a 100-client setting to mimic the large volume of participating edge devices in practical federated learning. Similar to previous works [6], [7], we set the parameter E, i.e., the number of training passes each client makes over its local dataset on each round, as 4. We also set the parameter B, the local mini-batch size used for the client updates as 2. For Gaia and CMFL, we set the learning rate η and the significance threshold v as two time variables that decreases over time: $\eta_t = \eta_0/\sqrt{t}$ and $v_t = v_0/\sqrt{t}$.

Communication overhead. We measured the learning accuracy and accumulated communication rounds over time and depict their relationship in Fig. 4. Those communications which are eliminated from being uploaded will not be counted. In order to do a thorough analysis for Gaia and CMFL despite of the influence of the threshold, we tested various threshold values to identify the significance of local updates and chose the threshold with the *best* performance for plotting.

We can observe that in both training models, CMFL significantly reduces communication rounds without losing the learning correctness. In contrast, Gaia shows similar behavior



(a) MNIST CNN



(b) Next-Word-Prediction LSTM

Fig. 4: The performance of CMFL compared to the basic federated learning and Gaia. (a) Learning accuracy vs. communication rounds for the MNIST CNN. (b) Learning accuracy vs. communication rounds for the Next-Word-Prediction LSTM, x-axis is in log scale.

TABLE I: Summary of Speedup for different learning accuracy in MNIST CNN and Next-Word-Prediction LSTM.

	Gaia	**CMFL**
MNIST CNN 60% Accuracy	1.25	3.45
MNIST CNN 80% Accuracy	1.13	3.47
NWP LSTM 60% Accuracy	1.42	13.35
NWP LSTM 80% Accuracy	1.26	**13.97**

to the basic federated learning, decreasing only a *slight* number of communication rounds.

Motivated by the visualized results above, we are curious to know to what extent our CMFL outperforms Gaia in terms of improving the communication efficiency for federated learning. To this end, we compared CMFL against Gaia based on a metric called *Speedup*. Speedup is defined, for a given learning accuracy, as the number of required communication rounds under the compared algorithm normalized by that under the basic federated learning, i.e.,

$$\text{Speedup} = \frac{\text{Compared Commun. rounds}}{\text{Commun. rounds under basic federated learning}}.$$

Intuitively, the greater the Speedup is, the more significantly the compared algorithm improves the communication efficiency. We measured the Speedup of both Gaia and CMFL in MNIST CNN and Next-Word-Prediction LSTM under different leaning accuracies. Table I gives a statistical summary the Speedup for both algorithms when reaching the target accuracy of 60% and 80%, respectively.

In particular, for the MNIST CNN model, when the learning accuracy raises to 60%, the basic federated learning costs 500 communication rounds, while Gaia *slightly* reduces this overhead to 400. On the other hand, our CMFL significantly reduces the required communication rounds to 145, providing Speedup of 3.45. Furthermore, when the learning accuracy reaches nearly the highest value, i.e., 80%, the basic federated learning and Gaia take 900 and 800 rounds, respectively. Our CMFL uses only 259 rounds, increasing the communication efficiency by 3.47× in terms of Speedup.

For the more complicated Next-Word-Prediction (NWP) LSTM, the communication overhead obviously increases in all of the three algorithms. Intuitively, the neural network in LSTM is more complex and the real-world data from the dialogue dataset is highly non-IID, making the training harder to be converged. As shown in Fig. 4b, the basic federated learning uses 40, 200 rounds to obtain a training model with the accuracy of 60%. Setting the significance threshold as 0.25 provides the best performance under Gaia, costing 31, 900 rounds to reach the same learning accuracy with the Speedup of 1.42. CMFL provides the Speedup of 13.35 with the threshold tuned as 0.7, reducing the required the number of communication rounds to 2, 877. Moreover, our CMFL *dramatically* reduces the communication rounds from 56, 600 to 4, 241 when setting the learning accuracy as 80%, enhancing the communication efficiency by 13.97×, whereas the Speedup under Gaia is only 1.26. In summary, CMFL outperforms Gaia by more than 11× in terms of the Speedup.

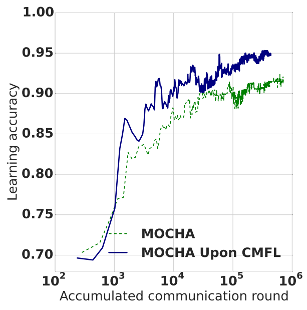
We attribute this to Gaia's fixed threshold could *not* effectively identify the updates' significance during the entire training process. For example, in the Next-Word-Prediction model in Fig. 4b, the threshold with the value of 0.25 will identify almost all the updates before the 20, 000th rounds as significant, losing the potential opportunity to mitigate useless updates. We make two remarks on these results:

1) *CMFL consistently outperforms Gaia in improving the communication efficiency for federated learning in both training models under various learning accuracies.* As we can see in Table I, CMFL keeps outperforms Gaia by more than 2.8× in MNIST CNN and more than 9.4× in Next-Word-Prediction LSTM.

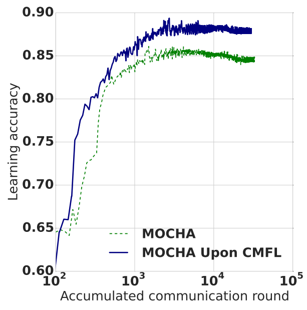
2) *CMFL provides much more significant enhancement in the communication efficiency under more complicated training models.* Intuitively, in the complex training models such as the Next-Word-Prediction LSTM shown in Fig. 4b where there is a huge number of training parameters, employing CMFL to eliminate those useless updates from being uploaded can provide obvious benefit.

B. Simulation of Federated Multi-Task Learning

CMFL can generally support the follow-up federated learning designs and further increase their communication efficiencies. To illustrate this, we built the recently proposed MOCHA [16] upon CMFL and developed the following simulation. Specifically, CMFL identifies local updates' significance in MOCHA's Federated Multi-Task Learning by locally calculating the changing of the global matrix based on the local update and the record of the relationship matrix among clients.



(a) Human Activity Recognition



(b) Semeion Handwritten Digit

Fig. 5: Illustration of CMFL's generality in supporting Federated Multi-Task Learning, x-axis is in log scale. (a) Learning accuracy vs. communication rounds for the Human Activity Recognition dataset. (b) Learning accuracy vs. communication rounds for the Semeion Handwritten Digit dataset.

Workload. Here we use two datasets to develop our simulation.

1) *Human Activity Recognition dataset.* The first is the Human Activity Recognition dataset [32] the same as [16]. The dataset contains 10299 samples each with a 561-length feature vector used to predict between sitting and the other activities. We further randomly separate the data into 142 clients each with 10 to 100 samples.
2) *Semeion Handwritten Digit dataset.* The second is the Semeion Handwritten Digit dataset [33] containing 1593 samples with 256 features. We predict the digit between zero and other numbers. These samples are randomly divided into 15 clients each with 10 to 200 samples.

Setup. In our simulations, we use a 142-client cluster for Human Activity Recognition to mimic a large-scale federated learning setting and a 15-client cluster to behave as a small-scale one in Semeion Handwritten Digit. We also set the number of local training iteration within each communication round as $E = 10$ and the local mini-batch size as $B = 3$. For simplicity, we set a constant learning rate as $\eta = 0.0001$.

Communication overhead. Similarly, we plot the learning accuracy and the accumulated communication rounds relationship over time in Fig. 5. We also choose the threshold with the best performance to plot. Specifically, in the Human Activity Recognition dataset, when setting the learning accuracy as 91%, using MOCHA with the support of CMFL only needs 4892 rounds, whereas directly implementing MOCHA requires 28120 rounds, 5.7× of that with CMFL. For the Semeion Handwritten Digit dataset, training the model to reach the learning accuracy of 84% takes 1500 and 460 rounds in MOCHA and MOCHA with CMFL, respectively. In this dataset, CMFL improves the communication efficiency with the Speedup of 3.3×.

Learning accuracy. More impressively, as we can see in Fig. 5, CMFL can even enhance the learning accuracy from 92.07% to 94.94% (1.03×) and 86.03% to 89.37% (1.04×) in the two datasets, respectively.

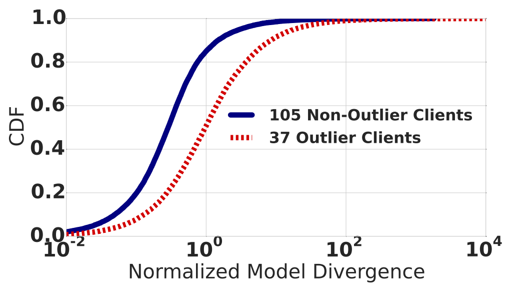


Fig. 6: Distribution of the Normalized Model Divergence (d_j) between outlier/non-outlier local models and the global model in Human Activity Recognition, x-axis is in log scale.

We attribute this to CMFL's elimination of those insignificant updates which are essentially outliers that will *harm* the global training. In order to illustrate this point, we dived into the statistical summary of clients whose local updates are *frequently* eliminated from being uploaded. We found that among all the **142** clients in the Human Activity Recognition dataset, there are **37** clients the numbers of whose eliminated updates are more than 2000. The sum of the eliminated updates in these 37 clients is up to 84.5% of the total number of local elimination. This means *the identified insignificant local updates mainly come from a small subset of the clients.*

Based on this observation, we separated the 142 local optimized models into 37 outliers and 105 non-outliers. We then use the metric of Normalized Model Divergence defined in Eq. (7) and summarize the average global-local parameter divergences in these two parts and plot their distributions in Fig. 6, respectively. As we can see, the 105 non-outlier clients show an obviously smaller model divergence of the global model than that of the 37 outliers. Specifically, in the 37 outliers, there are more than 50% of the parameters whose model divergences are higher than 100%, whereas this value is only 15% in the subset of those 105 non-outliers. Uploading the local updates of these outliers will bring *negative* influence to the model convergence, yet bringing *unnecessary* communication cost. In summary, CMFL is able to efficiently identify and eliminate those outliers, and thus not only reduce the communication overhead but also enhance the global learning accuracy.

C. EC2 Deployment

We next micro-benchmark the performance of CMFL using our real-world implementation.

Using EC2 clusters to run the testbed. We used an EC2 cluster to prototype the client-side model training as well as the cloud-side coordination. We did not use the real edge devices suck as mobiles or tablets to emulate the scarce network connections, due to the following two reasons: First, the slow network connection along with the limited computation capacity at the real edge-side devices will significantly slow down the training convergence. Second, given our measurement of the communication overhead in this paper is the accumulated communication rounds, i.e., the uploaded data size, using

EC2 cluster with more stable and larger bandwidth network connection will speed up the experiment without impacting the size of the transmitted data.

Implementation. We have prototyped CMFL in `Python` with a master-slave architecture. In particular, the `master` aggregates the local optimizations and sends the new global model back to clients in each learning iteration, as shown in Algorithm 1; a `slave` performs as a client, running local training and identifying the significance of its local updates before sending them to the `master`.

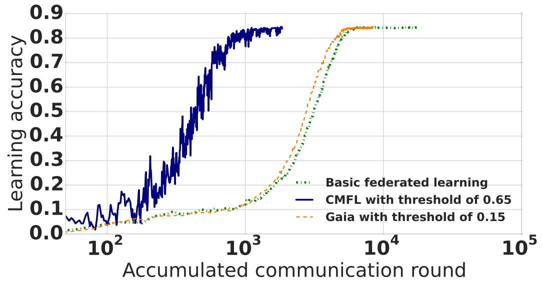
In our implementation, the `master` executes the *synchronous* global optimization based on the aggregation of all the local updates in each training iteration. In particular, each `slave` periodically sends its update to the `master`. If the local update is identified as insignificant, the `slave` eliminates its uploading by sending status information to the `master`, indicating the completion of its local training and the elimination of its update in the current iteration. The transferred data size of this status information is negligible when compared with an entire local update with all the parameters in the weight matrix. Once receives all of the local updates or elimination information from the 30 `slaves`, the `master` executes an aggregation by averaging those significant updates and feeds the new global model to all the clients for the next iteration.

Cluster deployment. We performed experiments in a 30-node Amazon EC2 [34] cluster. For each node, we used a `m4.xlarge` EC2 instance with 4 cores and 16 GB RAM.

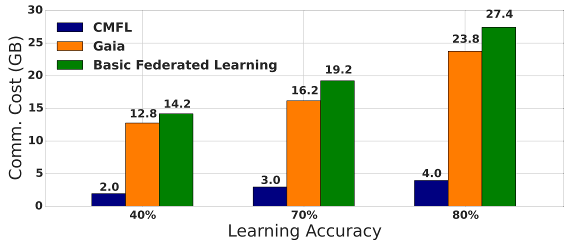
Testbed-benchmark. To benchmark the behavior of CMFL in a more controlled manner, we ran the same Next-Word-Prediction LSTM in Sec. V-A, where we separated the training data into 30 clients each with the dialogue of 3 roles. Similarly, we plot the training process of Gaia and CMFL with the threshold that produces the *best* performance in Fig. 7.

Communication overhead. Fig. 7a depicts the learning accuracy and communication costs over time. We can see a similar trend to Fig. 4b, i.e., CMFL outperforms Gaia, significantly reducing the uploading rounds. To better illustrate CMFL's efficiency, we measured the amount of uploaded data during the learning procedure among these schemes, as shown in Fig. 7b. Specifically, CMFL reduces the size of the uploaded data by 7.1$\times$, 6.4$\times$ and 6.9$\times$ given the three learning accuracy values, respectively.

Computation overhead. In our 30-client benchmark, the computation of checking the significance of an update takes less than 1.6 microseconds on average, while each client-side learning iteration costs about 1.25 second in our `m4.xlarge` EC2 instance with 16 GB RAM, 4 cores. In other words, checking the updates' significance takes less than 0.13% time of the local training iteration. In summary, CMFL can be deployed to real-world federate learning with only slight additional computation overhead.



(a) Training process of the Next-Word-Prediction LSTM



(b) Communication overhead with different accuracy

Fig. 7: Real-world implementations of CMFL with a 30-machine Amazon EC2 cluster. (a) Learning accuracy vs. communication rounds for the Next-Word-Prediction LSTM. (b) Communication cost with different learning accuracy.

VI. Conclusion

In this paper, we have studied improving the efficiency of the underlying communication mechanism for federated learning, which is orthogonal to the existing designs. We for the first time have proposed to feed the information of global updates back to participating clients as a reference to identify and preclude the insignificant local updates from being uploaded. We have also proposed a new communication schema, called Communication-Mitigated Federated Learning (CMFL), to identify the local update's significance by the ratio of its parameters whose value have the same sign with the global update. CMFL outperforms existing Gaia by being aware of whether the local updates follow the collaborative trend among all the clients. We have shown that under some practical assumptions, CMFL is guaranteed to approach the learning convergence. Both simulations and EC2 deployments have confirmed that CMFL outperforms state-of-the-art alternatives and generally supports existing federated learning designs by reducing the communication overhead.

References

[1] J. Poushter *et al.*, "Smartphone ownership and internet usage continues to climb in emerging economies," *Pew Research Center*, vol. 22, pp. 1–44, 2016.

[2] M. Anderson, *Technology device ownership, 2015*. Pew Research Center, 2015.

[3] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," pp. 456–464, 2010.

[4] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of dnns with natural gradient and parameter averaging," *arXiv preprint arXiv:1410.7455*, 2014.

[5] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," pp. 685–693, 2015.

[6] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[8] S. Samarakoon, M. Bennis, W. Saady, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *arXiv preprint arXiv:1807.08127*, 2018.

[9] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," *arXiv preprint arXiv:1804.08333*, 2018.

[10] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "On-device federated learning via blockchain and its latency analysis," *arXiv preprint arXiv:1808.03949*, 2018.

[11] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[12] L. Sweeney, "Simple demographics often identify people uniquely," *Health (San Francisco)*, vol. 671, pp. 1–34, 2000.

[13] G. LLC, "Gboard - the google keyboard," https://www.apkmirror.com/apk/google-inc/gboard/.

[14] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," *Google Research Blog*, 2017.

[15] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds." 2017.

[16] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4427–4437.

[17] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[18] W. House, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," *White House, Washington, DC*, pp. 1–62, 2012.

[19] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," *arXiv preprint arXiv:1611.04482*, 2016.

[20] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[22] F. Chen, Z. Dong, Z. Li, and X. He, "Federated meta-learning for recommendation," *arXiv preprint arXiv:1802.07876*, 2018.

[23] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models." in *AAAI*, 2016, pp. 2741–2749.

[28] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[29] "Cmfl: Communication-mitigated federated learning," https://home.cse.ust.hk/~lwangbm/TechReport_CMFL.pdf, Tech. Rep., 2019.

[30] W. Shakespeare., "The complete works of william shakespeare by william shakespeare," http://www.gutenberg.org/eBooks/100.

[31] T. team, "Tensorflow," http://www.tensorflow.org.

[32] A. G. L. O. X. P. Jorge L. Reyes-Ortiz, Davide Anguita, "Human activity recognition using smartphones data set," http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones.

[33] B. Tactile Srl, "Semeion handwritten digit data set," https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit.

[34] "Amazon ec2," http://aws.amazon.com/ec2.