CMFL: Mitigating Communication Overhead for Federated Learning

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

***Abstract*—Federated Learning enables mobile users to collaboratively learn a global prediction model by aggregating their individual updates without sharing the privacy-sensitive data. As mobile devices usually have limited data plan and slow network connections to the central server where the global model is maintained, mitigating the communication overhead is of paramount importance. While existing works mainly focus on reducing the total bits transferred in each update via data compression, we study an orthogonal approach that identifies *irrelevant* updates made by clients and *precludes* them from being uploaded for reduced network footprint. Following this idea, we propose Communication-Mitigated Federated Learning (CMFL) in this paper. CMFL provides clients with the feedback information regarding the *global tendency* of model updating. Each client checks if its update *aligns* with this global tendency and is *relevant* enough to model improvement. By avoiding uploading those irrelevant updates to the server, CMFL can substantially reduce the communication overhead while still guaranteeing the learning convergence. CMFL is shown to achieve general improvement in communication efficiency for almost all of the existing federated learning schemes. We evaluate CMFL through extensive simulations and EC2 emulations. Compared with vanilla Federated Learning, CMFL yields 13.97x communication efficiency in terms of the reduction of network footprint. When applied to Federated Multi-Task Learning, CMFL improves the communication efficiency by 5.7x with 4% higher prediction accuracy.**

I. Introduction

Due to the widespread adoption of mobile and edge devices [1], [2], the past few years have seen a surging demand of performing Machine Learning (ML) on these devices for personalized, low-latency AI applications [3], [4]. Federated Learning (FL) [4]–[9] has been proposed to enable *collaborative machine learning* over a large number of edge devices (*clients*) without centralized training data. In FL, a *global model*, maintained in a central server, is shared by all participating clients. To train the model, each client uploads its individual *update* to the server while keeping the privacy-sensitive training data on its own device. The server aggregates the updates of all clients and applies it to the model before proceeding to the next training iteration [3], [4].

However, a key challenge posed to FL is the *communication overhead*. On one hand, in FL, participating edge devices such as mobile phones and tablets usually have a limited data plan with unreliable, expensive network connections. On the other hand, advanced machine learning applications deployed on edge devices, such as Gboard [10], increasingly employ complex deep neural networks (DNNs), where the training update uploaded by each client consists of a large gradient vector, making communication a severe bottleneck. Therefore, minimizing the network footprint for FL becomes critically important [4], [5].

In general, there are two approaches to reduce the communication overhead during the model training: (1) reducing the total bits transferred for each client update, and (2) reducing the total number of updates transferred for each client. Existing works mainly focus on the first approach by means of *data compression*. For example, structured updates [4] is proposed to compress the client update using a more compact data structure. However, data compression results in information loss of training updates, which may harm the learning accuracy and usually come with no convergence guarantees [4].

In this paper, we turn to the second approach and ask: is it possible to improve the communication efficiency of FL while still providing convergence guarantee? We give an affirmative answer to this question. We propose to reduce the communication overhead by dynamically identifying *irrelevant* updates made by clients and excluding them from data transfer. This simple approach is *efficient* and *general*. First, as FL is performed on edge devices, the training updates are computed based on the personal data of clients. Given the non-IID distribution of these training data, some local updates can be *biased* and are simply *outliers*. Integrating these biased updates would drive the global model in a *tangential direction* to the collaborative convergence. Therefore, excluding these outliers from model updating has no adverse impact to the learning accuracy, but it avoids unnecessary data transfer. Second, this approach generally applies to a wide range of FL frameworks, provided that their model training is based on aggregating the client-side optimizations.

A simple solution that implements a similar intuition goes to Gaia [11], which is proposed to minimize the communication overhead for geo-distributed machine learning by excluding the *helpless* updates from data transfer. However, instead of checking if a local update is relevant to the global convergence, Gaia concerns the *significance* of a local update by comparing its *absolute value* (magnitude) with a predefined threshold: the update is considered significant if its magnitude is larger than the threshold. The local updates which are insignificant in magnitude are not uploaded to reduce the network footprint. While Gaia is proven effective for geo-distributed learning across a small number of datacenters [11], it does not transfer well to the FL setting due to a number of mismatches in assumptions and target environments. In particular, the magnitude-based significance metric concerns only the *speed* of model training, while being agnostic to the *optimization direction*. By simply comparing the significance of an update with a fixed threshold, Gaia is unable to tell if this update, which is a local gradient, aligns with the *collaborative optimization trend* across all clients, and hence cannot correctly identify its (ir)relevance. This problem can be even more salient given a large number of clients (e.g., hundreds of thousands of edge devices) participating in FL. As we shall show in Sec. V, directly applying Gaia to FL results in marginal savings of communication overhead.

Given the inefficiency of Gaia, we address this challenge with a novel algorithm called Communication-Mitigated Federated Learning (CMFL). Our key insight is to identify the *relevance* of a client update by checking if it aligns with the *global tendency* of model updating that jointly considers all clients. Specifically, in each learning iteration, a client first receives the feedback information about the *global update* from the central server. Next, the client proceeds its local training and produces a *local update*. The client then compares it with the global update, checking how well the two gradients align with each other. To do so, given a local update, CMFL calculates the percentage of its parameters having different signs—positive or negative—compared with their counterparts in the global update. Intuitively, the higher the percentage is, the more *divergent* to the collaborative convergence the local update is, and thus the more *irrelevant* it will be. By excluding those irrelevant local updates from data uploading, CMFL can effectively reduce the communication overhead of FL while still providing *provable convergence guarantees*.

We evaluated CMFL through both simulations and EC2 emulation, and compared its performance against state-of-the-art solutions, including vanilla FL [5], Gaia [11], and the recently proposed federated multi-task learning [12]. Evaluation results show that CMFL outperforms the vanilla Federated Learning by 13.97x in terms of the reduced communication overhead, whereas Gaia only achieves 1.26x improvement. Moreover, when applied to the advanced Federated Multi-Task Learning [12], CMFL yields 5.7x savings of communication overhead, while at the same time improving the prediction accuracy by 1.04x.

II. Model and Background

In this section, we briefly introduce Federated Learning (FL) and describe our objectives. We then survey related work and motivate the need for a new optimization scheme to reduce the network footprint of Federated Learning.

A. Federated Learning

Synchronous update scheme. Following the previous work [4], [5], [13], we assume a *synchronous* update scheme that proceeds in each training iteration of FL:

1) Clients independently train their local models using the client-side training data, e.g., the click history of Gboard [3], [10].
2) All the clients upload their local optimizations (gradient updates) to the cloud-side central server for aggregation.
3) The central server aggregates the received local updates (typically by averaging) to obtain a global update, and uses it to improve the global model. The system then proceeds to the next training iteration, where each client computes the update based on the newly updated model.

Privacy Protection. Instead of uploading the privacy-sensitive training data directly to the central server, the participating clients in Federated Learning only send the model updates which could be ephemeral [2], [5], [14]–[16]. This anonymous update reveals no information about the source client. Besides, as the global optimization in the central server requires no metadata about the sources of updates, the communication can be performed without exposing the personal information.

Model Training. We now formalize the description of the model training process in FL as follows. Let $\mathbb{C} = \langle c_1, \ldots, c_D \rangle$ be the set of participating clients in FL. Each client has a private training dataset $\mathscr{P}_k$. A global prediction model is shared and collaboratively trained by all D clients. The goal is to find the optimal model parameters $\mathbf{x} \in \mathbb{R}^l$ that minimize the *average* prediction loss $f(\mathbf{x})$:

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

where $f_k(\mathbf{x})$ is the loss value given by client k based on its private training data.

As a common practice, we establish the federated optimization in Eq. (1) through Stochastic Gradient Descent (SGD) [17], where the *batch gradient* is calculated in each training iteration and used to improve the model. 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 $\nabla f_k(\cdot)$ and η_k respectively denote the gradient function and the learning rate of client k, and $\mathbf{u}_{k,t}$ denotes the local update of client k given in the t^{th} iteration: $\mathbf{u}_{k,t} = -\eta_k \nabla f_k(\mathbf{x}_t)$. Assuming the learning converges in iteration T, we describe the learning process as a sequence of optimizations, 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*. As we explained in the introduction, FL clients are usually edge devices with the limited data plan and expensive mobile connections. It is therefore desirable to minimize the total amount of data uploaded from the edge devices to cut the mobile bills. To this end, our first objective is to minimize the *accumulated communication rounds* used for transferring client updates throughout the entire training process.

Formally, in the t^{th} iteration, let $\mathbb{S}_t$ be set of clients who upload their local updates to the 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 prediction accuracy a, suppose this can be achieved in T iterations using an algorithm $\mathscr{A}$. The *accumulated communication rounds* is defined as the total number of local updates made by clients in T iterations:

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

Therefore, our first objective is to design a communication-efficient algorithm $\mathscr{A}$ to minimize the accumulated communication rounds $\Phi^a_{\mathscr{A}}$ given learning accuracy a.

Guaranteeing the learning convergence. Communication efficiency should not be improved at the expense of learning convergence. In particular, let $\mathbf{x}^*$ be the *optimal* model parameters that minimize the prediction loss (1). We require the learning algorithm to meet the following convergence requirement:

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

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 learning algorithm*, i.e.,

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

C. Related Work

Advanced designs of Federated Learning. There is a rich body of work on FL. Most of them are limited to exploring sophisticated model training architectures. For example, MOCHA [12] employs the multi-task learning (MTL) framework to capture the distributed nature of FL, where the goal is to train separated but related models simultaneously. Formally, MOCHA captures the relationship between all clients (tasks) through their relationship matrix. Another recent advance is Federated Meta-Learning [18], where the user's information is shared at the level of algorithms instead of models or data as assumed in the previous approaches. Although these learning frameworks can accelerate the training process, they show little improvement in reducing the communication overhead.

There are other works which implemented the complex data structures to compress the total amount of data communicated in each update, such as structured updates and sketched updates [4]. Unfortunately, these works come without a convergence guarantee while adding the computational complexity during the communication stages.

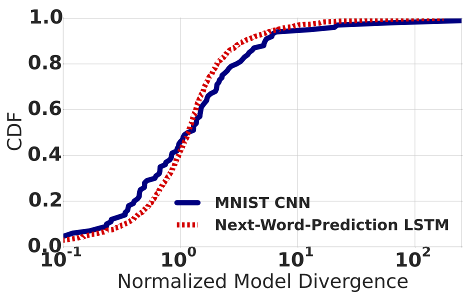


Fig. 1: Distribution of the Normalized Model Divergence (d_j) for all of the trained parameters $x_j \in \mathbf{x}$.

Prior art in geo-distributed learning. Gaia [11] is recently proposed as a communication framework that reduces cross-datacenter communications for geo-distributed machine learning. Specifically, Gaia measures the significance of a local update from one datacenter by the update's magnitude relative to the current parameter value, i.e., $||\frac{\text{Update}}{\text{Model}}||$, where $||.||$ is the Euclidean norm. Any update with $||\frac{\text{Update}}{\text{Model}}|| <$ Threshold are considered *insignificant* and will be precluded from uploading. We notice that this identification is made based only on the local update, without accounting for the federation of all clients. In the context of Federated Learning where there are a large number of clients each with non-IID training data, this *open-loop* method without a *feedback* of the global tendency fails to efficiently identify helpless updates.

III. Intuition and Challenges

In this section, we present our intuition of reducing the communication overhead by means of excluding irrelevant local updates. We also discuss the challenges to implement this intuition in FL.

A. Intuition

We begin our discussion by analyzing why some local optimizations are not helpful to the global convergence.

The divergence between global and client models. In FL, the global model is obtained by aggregating (e.g., averaging) a large number of local models on the client side. As local models are trained using the *client-specific* training data on devices, the discrepancy between the local and global models generally exists. Given the non-IID nature of the on-device training data, some local updates may not be representative following the population distribution. For example, when training a query suggestion model of Google's Gboard [3], clients have different clicking choices. While some choices align well with the common preference, others may produce client-specific updates which are *tangential* to the collaborative trend of the training convergence.

In this work, we measure the difference between the global and local parameters by a metric called *Normalized Model Divergence*. Normalized Model Divergence is defined, for each model parameter $x_j \in \mathbf{x}$, as the *average difference* between its values in the client and global models normalized by the global value, i.e.,

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

where $x_{j,k}$ is the local value of 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 divergent the global and client-side models are with respect to the trained parameter x_j.

In order to illustrate the divergence between the global and client-side models, we trained two models following the previous work [19]: MNIST CNN [20], [21] and Next-Word-Prediction LSTM [22]–[24], each with the dataset distributed to 100 clients. The detailed description of these two models and datasets can be found in Sec. V-A. We measured the Normalized Model Divergences in these two models and depict their CDF distributions in Fig. 1. We see more than 50% of parameters in both models produce model divergence higher than 100%, suggesting a drastic 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 significant divergence to the client-specific training data.

To summarize, there are a salient number of client-specific local optimizations which are tangential to the collaborative training convergence. Uploading these outliers to the central server makes little contributions but can even *do harm* to the convergence of the global model.

Intuition. Based on our discussions, there is a potential opportunity to reduce the communication overhead without uploading the outlier updates. Our intuition in this regard is to dynamically identify the relevance of client-side updates based on whether these updates align with the collaborative convergence trend, or they are just outliers with client-specific optimization. Irrelevant updates will not be uploaded.

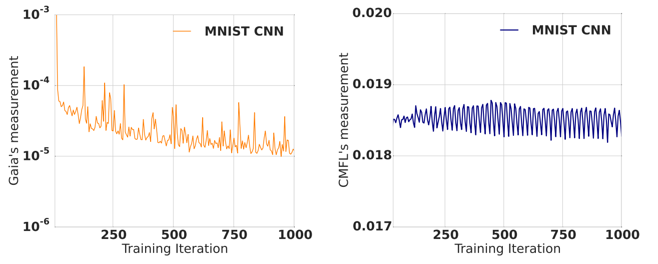
B. Challenges

However, there are two main challenges of implementing the intuition above.

How to guarantee learning convergence? Ideally, our solution should guarantee the training process to converge fast. However, reducing the amount of data transfer means to lose some training information, which would inevitably disturb the convergence of the model training. Therefore, our solution should judiciously determine which updates should be discarded without slowing down convergence. We note that although some FL algorithms [4] can effectively reduce the communication overhead, they provide *no* convergence guarantee.

How to measure the relevance of an update? As we have discussed in Sec. II-C, Gaia [11] provides a simple solution for geo-distributed machine learning that determines which local updates should *not* be transferred based on their *magnitude*, i.e., any updates satisfying $||\frac{\text{Update}}{\text{Model}}|| <$ Threshold are considered insignificant and will not be uploaded. A natural question is: can we apply the same solution to Federated Learning?

Unfortunately, while magnitude serves as a good indicator for geo-distributed machine learning performed in only *a few* datacenters, it is not a good fit for Federated Learning due to the following three reasons.



(a) The *significance* measure in Gaia decreases over iterations (b) The *relevance* measure Eq. (9) in CMFL stabilizes over iterations

Fig. 2: Comparison of two measures over iterations when training MNIST CNN model with 168 clients. (a) Gaia [11] measures the *significance* of a local update by its magnitude, which decreases exponentially over iterations (y-axis in log scale). (b) CMFL measures the *relevance* of a local update based on Eq. (9), which remains stable over iterations.

- First, given a large number of participating clients in FL and their non-IID data distributions, some clients have much heavier training workload and optimize many more parameters than others. These clients usually end up with a larger value of $||\frac{\text{Update}}{\text{Model}}||$. Besides, the update's magnitude also depends on the chosen learning rate. Therefore, it is hard, if not impossible, to set an appropriate *global threshold* of magnitude to identify helpless updates.
- Second, as the global model is the federation of a large number of clients, whether a local update is useful in model optimization cannot be simply measured by its magnitude. Consider a scenario in which a large number of clients making similar updates of small magnitude. These updates, though small in magnitude, are by no means helpless, because their federation shapes the global optimization direction. In contrast, a few outliers, though significant in magnitude, have no impact to the global model training in the presence of a massive number of more relevant updates.
- Third, as the training approaches convergence, the updates' magnitude decreases *exponentially*, making the threshold used in identifying updates' significance hard to tune. To illustrate this problem, we trained a model of MNIST CNN in FL with 168 clients following the same setting of [5]. We depict in Fig. 2a the average value of $||\frac{\text{Update}}{\text{Model}}||$ for all the clients in each learning iteration. As the training proceeds, the value decreases exponentially. If we set a large threshold, say, 5×10^{-5}, almost all the updates beyond the 300th iteration are identified as insignificant and are not uploaded. Consequently, the global optimization will *stagnate* after the 300th iteration, preventing the training from convergence. On the other hand, if we set a small threshold, say, 1×10^{-5}, almost all updates are considered significant, missing the opportunity to reduce the communication overhead before this watershed.

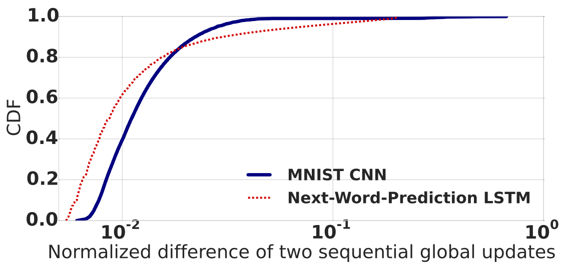


Fig. 3: Distribution of ΔUpdate, x-axis is in log scale.

We will show in Sec. V that directly applying Gaia [11] to FL can only achieve a *marginal improvement* in communication efficiency.

IV. Communication-Mitigated Federated Learning

In this section, we present our solution to minimize the network footprint of FL, which we call *Communication-Mitigated Federated Learning (CMFL)*. We first explain some key insights that guide our design. We then give a detailed description of CMFL. Finally, we show that our solution provides a *provable* convergence guarantee.

A. Key Insight

Estimating the global update. From the previous discussions, we see that in order to avoid transferring irrelevant local updates, each client need to know the collaborative optimization tendency in the global aggregation. Ideally, in each learning iteration, clients should compare their local updates with the global update so as to determine if their updates are relevant. However, a challenge in this regard is that the global update cannot be known *in advance* until all of the local updates have been aggregated in the current iteration.

To address this challenge, we use the global update made in the *previous iteration* to estimate that in the current iteration. Our insight is: given that model training usually converges *steadily* and *smoothly*, the difference between two sequential global updates should be small. Specifically, given two sequential global updates Update_t and Update_{t+1} that are respectively made in iterations i and $i+1$, we measure their *normalized difference* as

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

where $||.||$ is the Euclidean norm of a given vector. Intuitively, the larger the normalized difference is, the more the two sequential updates diverge from each other.

To verify if our insight holds in practice, we trained two models, MNIST CNN [20], [21] and Next-Word-Prediction LSTM [22]–[24], and for each model, we depict in Fig. 3 the CDF distribution of the normalized difference of two sequential global updates. We see that for MNIST CNN, more than 99% of the global updates have normalized difference smaller than 0.05, and the maximum difference is less than 0.67. For the Next-Word-Prediction model, in more than 93% of the training iterations, the difference is smaller than 0.05, and the maximum difference is only 0.21. We therefore confirm our insight that the global update made in the previous iteration can be used as a good estimation for that to be made in the current iteration. Implementing this prediction does not require additional communications, because each client already maintains the global update made in the previous iteration.

Measuring the relevance of an update. Given the (estimated) global update in the current iteration, we need to choose an appropriate metric to measure the relevance of a local update. As a training update is essentially a *gradient vector* of model parameters, we compare the two updates—local and global—*parameter-wise*. We compute how many parameters are of the same sign in the two updates, and normalize the result by the total number of parameters. This gives the *percentage of same-sign parameters* in the two updates. We shall use this percentage to measure the relevance of a local update. Specifically, let $\mathbf{u} = \langle u_1, u_2, \ldots, u_N \rangle$ be a local update over N model parameters. Let $\bar{\mathbf{u}}$ be similarly defined for the (estimated) global update. We measure the *relevance* of local update $\mathbf{u}$ with respect to global update $\bar{\mathbf{u}}$ as

$$e(\mathbf{u}, \bar{\mathbf{u}}) = \tfrac{1}{N} \textstyle\sum_{j=1}^{N} I(\text{sgn}(u_j) = \text{sgn}(\bar{u}_j)), \tag{9}$$

where $I(\text{sgn}(u_j) = \text{sgn}(\bar{u}_j)) = 1$ if u_j and $\bar{u}_j$ are of the same sign, and 0 otherwise.

Intuitively, given a parameter in an update, its sign determines the *direction* (increase or decrease) to which the model should be improved along the dimension of that parameter. Therefore, by comparing the signs of parameters in two updates, we could measure the "alignment" of the two updates.

Following this idea, our solution dynamically identifies relevant local updates and excludes those irrelevant from being updated. An update is considered *irrelevant* if its relevance measure (9) is smaller than a predefined threshold. Compared with prior work Gaia [11], this simple approach provides the following two benefits:

1) *It effectively identifies whether the client-side updates follow the collaborative direction or are just outliers.* Unlike Gaia, comparing the sign of parameters directly measures the alignment with the global update, irrespective of the learning rate and the size of local dataset. Therefore, it avoids the misidentification of the common yet slight updates and the outliers of large magnitude.
2) *The relevance measure remains stable throughout the learning process, making it easy to fine-tune threshold for good performance.* We measured the average relevance of local updates over the training process of MNIST CNN and depict the result in Fig. 2b. Compared with Gaia in Fig. 2a, the relevance measure keeps stable as the training proceeds. This makes threshold tuning a much easier task than Gaia.

B. Communication-Mitigated Federated Learning

We present our solution, Communication-Mitigated Federated Learning (CMFL), in detail.

Aggregation at the central server. As a starting point, the global aggregation at the central server is similar to the vanilla FL [4], [5]. Specifically, at the beginning of the t^{th} learning iteration, a global model $\mathbf{x}_{t-1}$ is distributed to all the clients as an initial model to train, along with the feedback information of the global update $\bar{\mathbf{u}}_{t-1}$ in the previous iteration. After receiving the relevant local updates from all clients, the central server aggregates them as the global update and uses it to refine the model. Algorithm 1 details the global aggregation process in procedure `GlobalOptimization`.

Local optimization on client-side devices. The client-side optimization is iteratively obtained by the local model training using on-device data. Before uploading the local update $\mathbf{u}_{k,t}$, CMFL calculates its relevance $e(\mathbf{u}_{k,t}, \bar{\mathbf{u}}_{t-1})$ using Eq. (9). Any local update with $e(\mathbf{u}_{k,t}, \bar{\mathbf{u}}_{t-1})$ smaller than a tuned threshold $v(t)$ is identified as *irrelevant*, and are not uploaded for aggregation. Algorithm 1 details the local optimization and relevance identification in two procedures, `LocalUpdate` and `CheckRelevance`, respectively.

Extensions. Although our solution is presented based on vanilla FL algorithm [4], [5], CMFL can be easily *generalized* to support a wide range of FL designs, provided that the global model is the aggregation of local updates. For example, MOCHA [12] employs the multi-task learning (MTL) where distributed clients independently train their individual models instead of sharing a global model. CMFL supports MOCHA by locally calculating the update of the global matrix based on the local training iterations and the record of the relationship matrix among client-side models. As we will show in Sec. V, CMFL not only reduces the communication overhead for MOCHA but also slightly improves its prediction accuracy.

Algorithm 1 Communication-Mitigated FL

```
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:          $\mathbb{S}^t \leftarrow \{\mathbf{u}_{k,t} | \quad s_{k,t}$ is True$\}$        ▷ relevant updates
8:          $\bar{\mathbf{u}}_t \leftarrow \frac{1}{|\mathbb{S}^t|} \sum_{\mathbf{u}_{k,t} \in \mathbb{S}^t} \mathbf{u}_{k,t}$        ▷ 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:     Execute the local training and obtain the local update $\mathbf{u}_{k,t}$
13:     $s_{k,t} \leftarrow$ CheckRelevance $(\bar{\mathbf{u}}_{t-1}, \mathbf{u}_{k,t})$
14:     if $s_{k,t}$ is False then
15:         $\mathbf{u}_k \leftarrow$ NULL        ▷ exclude irrelevant update
16:     return $(s_{k,t}, \mathbf{u}_{k,t})$
17: procedure CheckRelevance
18:     Input: Global update $\bar{\mathbf{u}}_{t-1}$ and Client-side update $\mathbf{u}_{k,t}$
19:     Calculate the relevance $e(\mathbf{u}_{k,t}, \bar{\mathbf{u}}_{t-1})$ following Eq. (9)
20:     if $e(\mathbf{u}_{k,t}, \bar{\mathbf{u}}_{t-1}) < v(t)$ then
21:         return True
22:     else
23:         return False        ▷ identify irrelevant updates
```

C. Convergence Guarantee

Our objective is to minimize the accumulated communication rounds while guaranteeing the learning convergence (i.e., Eq. (6)). However, given the non-IID distribution of the client-side training data, minimizing the required communication rounds $\Phi^{a}_{\mathscr{A}}$ remains an open problem [11]. Therefore, in this paper, we only show in theory that Algorithm 1 is guaranteed to converge, while relying on simulations and real-world testbed to illustrate the significant improvement of communication efficiency of our solution (Sec. V).

Assumptions: Our convergence analysis is based on two assumptions. First, we assume *convex* loss function $f(\mathbf{x})$, which is a standard assumption for tractable analysis in the Machine Learning literature [25].

Second, we assume that the global update in the *previous iteration* can be used to estimate the global update to be made in the current iteration. We have justified this assumption through measurement experiments in Sec. IV-A (Fig. 3).

Convergence Guarantee. We next show that CMFL is guaranteed to converge. To this end, we consider two scenarios: 1) The global model is trained in vanilla FL using the updates made by all clients, irrespective of their relevance; 2) The global model is trained using CMFL, without irrelevant updates. Let $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$ respectively denote the global models trained up to the t^{th} iteration in scenarios 1) and 2). Following Eq. (5), to show that Algorithm 1 converges, it is equivalent to proving $\lim_{T\to\infty} \frac{1}{T} R[\tilde{\mathbf{x}}] = 0$. We have the following theorem:

Theorem 1 (Convergence Guarantee). *Let η_t and v_t be the learning rate and relevance threshold in the t^{th} iteration, respectively. We bound the time-average loss function of Algorithm 1 as follows:*

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

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

We make two remarks on Theorem 1:

1) *The convergence of Algorithm 1 depends on the choice of learning rate η_t and relevance threshold v_t.* To ensure fast learning convergence, these two hyper-parameters should be set as time-decreased variables that make $\lim_{T\to\infty} \frac{1}{T} R[\tilde{\mathbf{x}}] \to 0$.
2) *There are a diverse choices of η_t and v_t that can guarantee convergence, though the convergence speed can be different.* For example, if we choose $\eta_t = \eta_0/\sqrt{t}$ and $v_t = v_0/\sqrt{t}$, we have $\lim_{T\to\infty} \frac{1}{T} R[\tilde{\mathbf{x}}] = \lim_{T\to\infty} \mathcal{O}(\sqrt{T}/T) \to 0$.

We next give a proof sketch of Theorem 1. The complete proof is deferred to our technical report [26] 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 $\langle \cdot, \cdot \rangle$ 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 $\mathbf{x}$ including all the updates regardless of their relevance, while the third part shows the additional loss between $\mathbf{x}$ and $\tilde{\mathbf{x}}_t$, due to the elimination of the irrelevant 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). □

V. Evaluation

We evaluated CMFL through both simulations and real-world emulations deployed on a 30-machine Amazon EC2 [27] cluster. The highlights of our evaluation are summarized below:

- CMFL substantially reduces the communication overhead of vanilla FL. Specifically, with CMFL, the communication efficiency is enhanced by 13.97x in terms of accumulated communication rounds. CMFL also outperforms Gaia [11] by 11x in terms of saving.
- CMFL provides *general* improvement in communication efficiency for a wide range of FL algorithms. In particular, when applied to Federate Multi-Task Learning [12], CMFL not only reduces the network footprint by 5.7x but also improves the learning accuracy by 1.04x.
- CMFL can be easily implemented in practice, with negligible computational overhead: it takes $< 0.13\%$ iteration time to identify the relevance of a local update.

A. Simulation of Vanilla FL

We start by evaluating the reduced communication overhead when applying CMFL to vanilla FL [5] via simulations. We also compare the communication efficiency improvement of CMFL and Gaia [11].

Workload. In order to provide comparability with existing works, we set up our first simulation using similar training models and datasets in [5]. In particular, we choose two machine learning models in our simulation:

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 [21]. The dataset contains $60,000$ samples of handwritten digits [28]. 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 (NWP) model using LSTM.* We train a 2-layer LSTM language model (each with 256 nodes) at the word-level, which after reading a fixed number of words in a sentence, predicts the next word [23]. Specifically, we develop the training dataset from *The Complete Works of William Shakespeare* [29]. 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: vanilla FL [5] and Gaia [11].

Setup. In our simulations, the model architectures were built upon TensorFlow [30]. We use a 100-client setting to mimic the large volume of participating edge devices in practical FL. Similar to previous works [4], [5], 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/relevance threshold v as two independent time variables that decrease over time: $\eta_t = \eta_0/\sqrt{t}$ and $v_t = v_0/\sqrt{t}$.

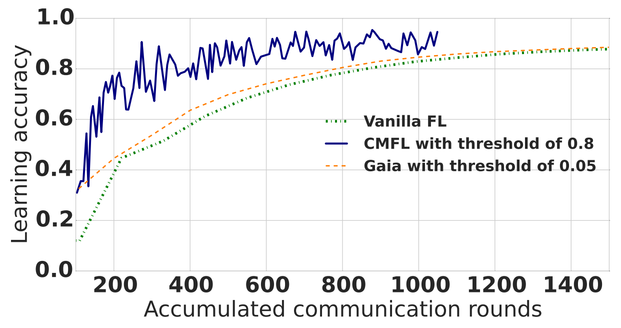
Communication overhead. We measured the learning accuracy and accumulated communication rounds in Eq. (4) over time and depict their relationship in Fig. 4. Those communications which are eliminated from being uploaded will not be counted. In order to do a thorough analysis for Gaia and CMFL despite of the influence of the threshold, we tested various threshold values to identify the significance/relevance of local updates and chose the threshold values with the *best* performance for plotting. In specific, we tested a set of 10 relevance threshold values for CMFL: $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9\}$, and another set of 10 significance threshold values for Gaia: $\{0.02, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.9\}$ in both two models. For the MNIST CNN model, the best performance is obtained when setting the relevance threshold value in CMFL as 0.8 and the significance threshold value in Gaia as 0.05. Similarly, these two values are tuned as 0.7 and 0.25 for the NWP LSTM to get the best performance, respectively.

We can observe that in both training models, CMFL substantially reduces the accumulated communication rounds without losing the learning correctness. In contrast, Gaia shows a similar behavior to the vanilla FL, decreasing only a *slight* number of network footprints. We can also observe that the curves of CMFL in both models are obviously jagged. We attribute this to CMFL's elimination of partial client-side updates, making the intermediate models mismatching part of the local datasets.

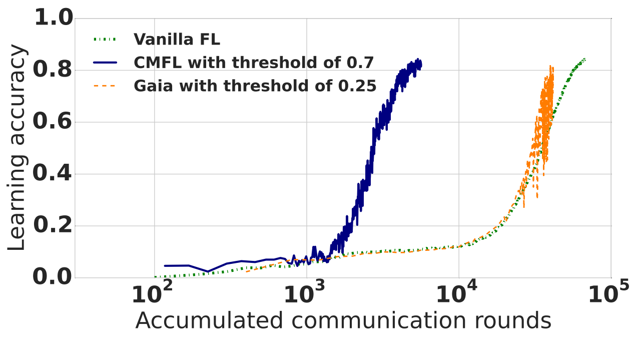
Motivated by the visualized results above, we are curious to know to what extent our CMFL outperforms Gaia in terms of improving the communication efficiency for FL. To this end, we compared CMFL against Gaia based on a metric called *saving*. Saving is defined, for a given learning accuracy a, as the number of required accumulated communication rounds under vanilla FL normalized by that under the compared algorithm $\mathscr{A}$, i.e.,

$$\text{Saving}^a_{\mathscr{A}} = \Phi^a_0 / \Phi^a_{\mathscr{A}},$$

where Φ^a_0 represents the accumulated communication rounds



(a) MNIST CNN



(b) Next-Word-Prediction LSTM

Fig. 4: The performance of CMFL compared to vanilla 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.

under vanilla FL, and $\Phi^{a}_{\mathscr{A}}$ represents that under the compared algorithm $\mathscr{A}$. Intuitively, the greater saving$^{a}_{\mathscr{A}}$ is, the more substantially algorithm $\mathscr{A}$ will improve the communication efficiency. We measured the saving of both Gaia and CMFL in MNIST CNN and NWP LSTM under different leaning accuracies. Table I gives a statistical summary of the saving 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 vanilla FL costs 500 communication rounds, while Gaia *slightly* reduces this overhead to 400. On the other hand, our CMFL substantially reduces the required communication rounds to 145, providing a saving of 3.45. Furthermore, when the learning accuracy reaches nearly the highest value, i.e., 80%, the vanilla FL and Gaia take 900 and 800 rounds, respectively. Our CMFL costs only 259 rounds, reducing the network footprints by 3.47x.

For the more complicated NWP LSTM, the communication overhead obviously increases in all of the three algorithms. Intuitively, the DNN 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 vanilla FL 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. CMFL provides the saving of 13.35 with the relevance threshold tuned as 0.7, reducing the required number of communication rounds to $2,877$. Moreover, CMFL *dramatically* reduces the communication rounds from $56,600$ to $4,241$ when requiring the learning accuracy as $80\%$with the saving of 13.97, whereas the saving under Gaia is only 1.26. In summary, CMFL outperforms Gaia by more than 11x in terms of the saving.

TABLE I: Summary of the saving for different learning accuracies 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**

We attribute this to Gaia's fixed significance threshold ignores the updates' relevance 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 $20,000^{\text{th}}$ rounds as *significant*, losing the potential opportunity to mitigate *irrelevant* updates. We make two remarks on these results:

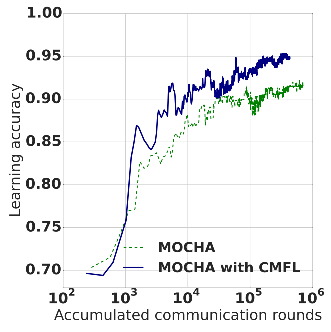
1) *CMFL consistently outperforms Gaia in improving the communication efficiency for FL in both training models under various learning accuracies*. As we can see in Table I, CMFL keeps outperforms Gaia by more than 2.8x in MNIST CNN and more than $9.4x$x in Next-Word-Prediction LSTM.
2) *CMFL provides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

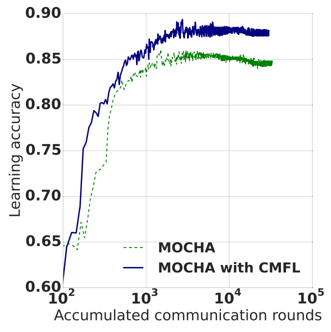
CMFL provides general improvements for almost all the follow-up FL designs in further reducing their network footprint. To illustrate this, we applied CMFL to recently proposed MOCHA [12] and developed the following simulation. Specifically, CMFL identifies local updates' relevance in MOCHA's federated 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.

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

1) *Human Activity Recognition dataset.* The first is the same Human Activity Recognition dataset [31] as used in [12]. 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.



(a) Human Activity Recognition



(b) Semeion Handwritten Digit

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

TABLE II: Summary of the saving when applying CMFL to MOCHA under various learning accuracies.

	MOCHA with CMFL
HAR 85% Accuracy	4.3
HAR 91% Accuracy	**5.7**
SHD 75% Accuracy	1.97
SHD 84% Accuracy	3.3

2) *Semeion Handwritten Digit dataset.* The second is the Semeion Handwritten Digit dataset [32] 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 FL 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 significance/relevance threshold values as 0.75 and 0.2 to get the best performance to plot, receptively. In particular, for 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, 5.7x 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 saving of 3.3x. Similarly, we also summarize the saving of CMFL compared with MOCHA in Table II.

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

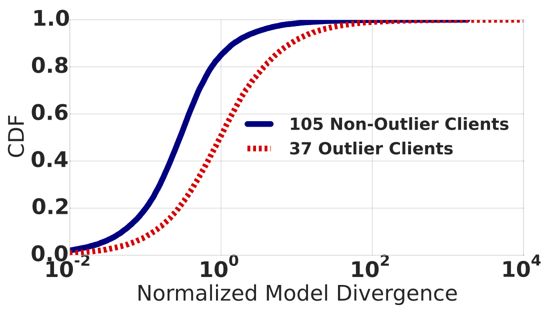


Fig. 6: Distribution of the Normalized Model Divergence (d_j) between outlier/non-outlier local models and the global model in Human Activity Recognition, x-axis is in log scale.

We attribute this to CMFL's elimination of irrelevant updates which are essentially outliers that will *harm* the global training. In order to illustrate this point, we dived into the statistical summary of the clients whose local updates are *frequently* eliminated from being uploaded. We found that among all the **142** clients in the HAR dataset, there are **37** clients whose eliminated updates are more than 2000. The total number of eliminated updates in these 37 outliers is up to 84.5% of the total eliminations, meaning the identified irrelevant local updates mainly come from a small subset of 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 Fig. 6. 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.

C. EC2 Deployment

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

Emulation based on an EC2 cluster. 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 like 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 process. Second, given our measurement of the communication overhead in this paper is the accumulated communication rounds defined in Eq. (4), i.e., the network footprint instead of the time span of data transfer, using EC2 cluster with more stable and larger bandwidth network connection will speed up the experiment without impacting the performance of algorithms.

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/relevance of its local updates before sending them to the `master`. To automate the deployment and management of CMFL in a real cluster, we developed a dedicated plugin based on `Ansible` [33], a popular `DevOps` tool to control the behavior of all clients and the central server.

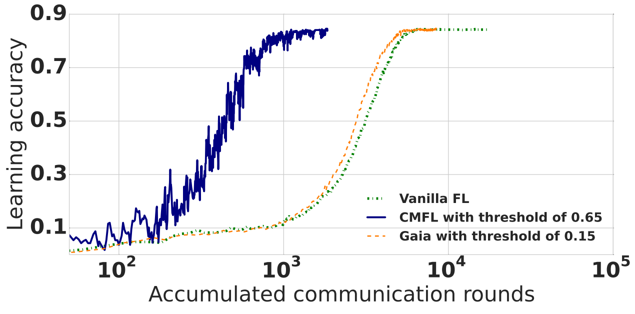
In our implementation, the `master` executes the *synchronous* global optimization through aggregating local updates in each training iteration. In particular, each `slave` periodically sends its update to the `master`. If the local update is identified as insignificant/irrelevant, the `slave` eliminates 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/relevant 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 [27] cluster. For each node, we used a `m4.xlarge` EC2 instance with 4 cores and 16 GB RAM.

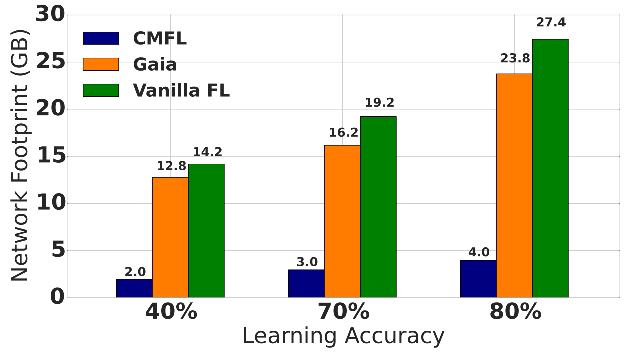
Testbed-benchmark. To benchmark the behavior of CMFL in a more controlled manner, we ran the same Next-Word-Prediction LSTM in Sec. V-A, where we separated the training data into 30 clients each with the dialogue of 3 roles. Similarly, we tested a set of 10 threshold values for CMFL: $\{0.1, 0.2, 0.3, 0.5, 0.55, 0.6, 0.65, 0.7, 0.8, 0.9\}$, and another a set of 10 threshold values for Gaia: $\{0.02, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.9\}$. We plot the training process of Gaia and CMFL with the thresholds 0.65 and 0.15 that produce the *best* performance in Fig. 7, respectively.

Communication overhead. Fig. 7a depicts the communication overhead under various prediction accuracies. We can see a similar trend to Fig. 4b, i.e., CMFL continuously outperforms Gaia, substantially reducing the uploading rounds. To better illustrate CMFL's efficiency, we measured the consumed network footprint during the learning procedure among these schemes, as shown in Fig. 7b. Specifically, CMFL reduces the size of the uploaded data by 7.1x, 6.4x and 6.9x given the three learning accuracy values, respectively.

Computation overhead. In our 30-client benchmark, the computation of checking the relevance of an update takes less than 1.6 microseconds on average, while each client-side learning iteration costing about 1.25 second in our `m4.xlarge` EC2 instances. In other words, checking the updates' relevance takes $< 0.13\%$ time of the local training iteration. In summary,



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



(b) Communication overhead with different accuracies

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

CMFL can be deployed to real-world federate learning with negligible additional computation overhead, and thus can be implemented at a large scale.

VI. Conclusion

In this paper, we have proposed to improve the communication efficiency of Federated Learning while at the same time providing guaranteed learning convergence. Our key idea in this regard is to identify and exclude *irrelevant* client-side updates trained over client-specific and biased data. Following this idea, we have proposed a new algorithm called Communication-Mitigated Federated Learning (CMFL). CMFL measures the *relevance* of a client-side update based on its alignment with the global update collaboratively produced by all participating clients. CMFL sets a tunable threshold and uses it to identify and exclude irrelevant updates whose relevance metric is smaller than that threshold. We have shown in theory that CMFL is guaranteed to converge. Both simulations and EC2 emulation have confirmed that CMFL results in significantly smaller network footprint compared with the state-of-the-art solutions, outperforming vanilla FL by 13.97x, Gaia by 11x, and Federated Multi-Task Learning by 5.7x.

Acknowledgement

The research was support in part by RGC GRF grants under the contracts 16211715, 16206417 and 16207818, a RGC CRF grant under the contract C7036-15G.

References

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

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

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

[4] J. Konečnỳ, 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.

[5] 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.

[6] 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.

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

[8] 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.

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

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

[11] 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." in *NSDI*, 2017, pp. 629–647.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

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

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

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[21] 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.

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

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

[24] 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.

[25] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[26] "Cmfl: Mitigating communication overhead for federated learning," https://home.cse.ust.hk/~lwangbm/TechReport_CMFL.pdf, Tech. Rep., 2019.

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

[28] C. J. B. Yann LeCun, Corinna Cortes, "Mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist/.

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

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

[31] 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.

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

[33] "Ansible," http://www.ansible.com/.