FedCA: Efficient Federated Learning with Client Autonomy

Na Lyu
Shanghai Jiao Tong University
Shanghai, China

Zhi Shen
Shanghai Jiao Tong University
Shanghai, China

Chen Chen
Shanghai Jiao Tong University
Shanghai, China

Zhifeng Jiang
Hong Kong University of Science and Technology
Hong Kong, China

Jiayi Zhang
Shanghai Jiao Tong University
Shanghai, China

Quan Chen
Shanghai Jiao Tong University
Shanghai, China

Minyi Guo
Shanghai Jiao Tong University
Shanghai, China

ABSTRACT

Federated Learning (FL) enables collaborate model training without privacy violation, where clients periodically report their updates to the server in communication *rounds*. Due to heterogeneous resource and limited bandwidth, FL processes often suffer from low efficiency. Existing works in that regard are oblivious to the *intra-round* execution status on clients; however, such status information has great potential to support flexible efficiency optimizations. In this paper, we propose FedCA, a novel mechanism that allows clients to autonomously exploit intra-round training status for higher efficiency. We first devise a metric to help quantify the statistical contribution of different iterations in a round, which can be efficiently profiled at runtime with the periodical sampling strategy. With the instantaneous system and statistical status, to improve computation efficiency, clients under FedCA can adaptively determine the intra-round workloads based on a utility function. Besides, to mitigate the communication bottleneck, for some parameters attaining fast local convergence, clients under FedCA can eagerly transmit their updates to the FL server prior to round completion. We implemented FedCA atop PyTorch, and large-scale experiments show that it can improve FL efficiency by over 15%.

ACM Reference Format:
Na Lyu, Zhi Shen, Chen Chen, Zhifeng Jiang, Jiayi Zhang, Quan Chen, and Minyi Guo. 2024. FedCA: Efficient Federated Learning with Client Autonomy. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673049

1 INTRODUCTION

Federated Learning (FL) [12, 22] is an effective paradigm that allows massive edge clients to jointly train a neural network model without revealing their private data. Confronting limited network bandwidth, existing FL practices typically adopt the FedAvg [22] algorithm for client collaboration: the FL clients first pull the latest model parameters from the FL server, and then—after multiple local iterations—report the updated model parameters back to the server for synchronization. Such a procedure is called a *round* and will repeat until model convergence.

Low training efficiency is a well-known hurdle for realistic FL deployment, which involves both the computation and communication aspects. Regarding the computation efficiency, the computing capability of typical FL clients (like cellphones or IoT devices) are quite limited and also highly dynamic; in particular, there usually exists strong system heterogeneity among clients [19], and those *stragglers* (i.e., with slower computing speed than others) may slow down the entire training process [3, 15, 33]. Regarding the communication efficiency, due to the poor network condition of edge devices, the model synchronization process remains a remarkable performance bottleneck for FL [5, 21].

In the literature, many research works have been proposed to improve the computation and communication efficiency of FL. For example, to address computation heterogeneity, some works [3, 15, 24] proposed to select a portion of clients with more consistent computing speed to participate in FL, and another work FedBalancer [28] proposed to properly set the aggregation deadline of each round. Meanwhile, to reduce the communication cost, existing works include quantization [4, 12], sparsification [5, 8] and synchronization-frequency tuning [6, 17, 30]. While proven effective, these methods nonetheless neglect significant optimization opportunities by being *server-autocratic*. That is, due to the privacy constraint and monitoring inconvenience, the intra-round training status on clients (at iteration granularity) is treated as a black box; the efficiency optimization decisions are made on the FL server and enforced on all the clients indiscriminately.

In fact, the instantaneous training status during the local iterations of a round, which is accessible only by the client itself, has great potential to enable intra-round optimizations. For example, a cellphone participating in FL may suddenly slow down when the user opens another app competing for computing resources; in that case, that cellphone shall proactively reduce its local iterations to avoid being stragglers. Besides, in the communication perspective, some parameters may converge faster than others during the local iterations on a client; to mitigate the network bottleneck, they can be immediately communicated to the FL server to enable computation-communication overlap. Note that, due to the loosely-coupled nature of FL, the instantaneous training status is highly heterogeneous on different clients in both system and statistical aspects. To fully unleash the intra-round optimization capabilities, we thus need to grant the FL clients with sufficient autonomy to timely react to the runtime training characteristics.

In this paper, we propose Federated Learning with Client Autonomy (FedCA), a novel FL mechanism that enables flexible intra-round optimizations on clients. We note that in FL, the server is insensitive to the intra-round training strategies on clients as long as they can yield identical updates as the default setup. Therefore, clients under FedCA need to improve the computation and communication efficiency with minimum change of the accumulated per-round update. For clarity, we introduce a metric, *statistical progress*, to quantify the similarity between the cumulative update so far and the expected update after a standard round. We find that each client has distinct curves on how statistical progress increases with the number of local iterations, and such curves are inconsistent across different layers even for the same client. With the knowledge of such statistical progress curves and instantaneous system status, to attain higher training efficiency, clients can autonomously decide whether to terminate the local training iterations or whether to eagerly transmit the early-converged layers. Our FedCA design further addresses three immediate challenges.

First, statistical progress curves—which are crucial for intra-round optimizations—shall be acquired efficiently and also apriori on each FL client. This renders it infeasible to naively record the model snapshot after each iteration. To address that challenge, we leverage the observations that the statistical-progress curves are similar across consecutive rounds and also across parameters within the same layer, and propose the *periodical sampling* strategy. That is, each client gets the statistical progress curves by analyzing only a few sampled parameters in each layer, once for multiple rounds.

Second, when reducing the local iterations for better computation efficiency, clients also need to consider the statistical value of the iterations-to-skip to ensure accuracy validity. To this end, we design a utility function that combines the system and statistical impacts of training for an additional iteration, which can guide clients to determine whether to terminate the local computations.

Third, eagerly transmitting the early-converged layers can mitigate the communication bottleneck, but due to the approximation in periodical sampling, the early-convergence diagnosis of some layers may turn out inaccurate for the current round. For such layers, which can be identified by the error between their eagerly-transmitted values and the actual ones in the end, we can re-transmit them after round completion.

We have implemented FedCA in PyTorch, and further evaluated its performance with a 128-node EC2 cluster emulating realistic FL setups. In our experiments, when confronting heterogeneous and dynamic resources, FedCA can attain a convergence speedup of over 15%. In particular, for the WideResNet-28 model, FedCA surpasses the second best method by 71.5% in per-round time and 45.3% in the overall time to reach a near-optimal accuracy. Meanwhile, we confirm that FedCA is in general robust to hyperparameter variation and the overhead it incurs is also negligible.

2 BACKGROUND

2.1 FL Basics

In Federated Learning (FL) [12, 22], a multitude of edge clients, such as smartphones and IoT devices, collaboratively train a global model using their local datasets. More specifically, in FL there is a central server maintaining the global model, and client updates are synchronized for global model refinement in communication *rounds*; under the *de facto* FedAvg algorithm, each round includes multiple local iterations to reduce the communication-computation ratio. While FL can protect data privacy, due to the loosely coupled nature, the resource quality of the FL participants is much worse than that in dedicated clusters, which manifests in three aspects:

- *Low capability.* For typical FL clients, both the computing power and communication bandwidth are inadequate.
- *Heterogeneity.* Across different FL clients, the computing capabilities and data distributions are usually heterogeneous.
- *Dynamicity.* On a given FL client, the available resources are dynamic during the local training process.

Given the resource issues above, realistic FL practices have long been plagued by poor efficiency in both computation and communication. In particular, heterogeneous and dynamic resource quality often leads to the *straggler* problem [3, 15], which means that some participants with inferior resources would delay the end-of-round synchronizations. Meanwhile, in the communication aspect, existing study has shown that model update transmission is a severe performance bottleneck for FL [5]: when training a CNN model under a realistic FL setup, over a half of the training time might be spent purely on communication. Therefore, to make FL more efficient, it is of paramount significance to mitigate the straggler effect as well as the communication bottleneck.

2.2 Prior Arts and Their Limitation

Many research works have been proposed to improve FL efficiency by respectively addressing the straggler problem and mitigating the communication bottleneck. We briefly go through those methods and summarize their common limitations.

Handling stragglers. Some works like Oort [15] and REFL [3] proposed to *proactively* evade stragglers by conducting cautious client selection; some other works propose to *reactively* suppress the negative impact of stragglers by not waiting for the late-arriving client updates. For example, the FedAvg algorithm [22] only collects a fixed portion of updates returned the earliest; FedBalancer [28] explicitly specifies a deadline and only collects the updates returned before the deadline. In that regard, FedProx [19] and SAFA [33] further proposed to try exploit the lately-returned updates from the stragglers to attain faster convergence. Recently, FedAda [39] proposes to mitigate stragglers by having the FL server adaptively adjust the intra-round workloads of the straggling clients.

Mitigating communication bottlenecks. To reduce the data transmission amount in each synchronization, quantization [4, 12] and sparsification [5, 8] are two classical methods—quantization means to use fewer bits for each element (originally represented by 32 bits), and sparsification means to reduce the total number of elements to be transmitted for each synchronization. Meanwhile, to reduce the total communication time, some works [6, 17, 30] have proposed to properly adjust the synchronization frequency based on the overall resource budget or instantaneous gradient status.

Limitation of existing works. While effective to some extent, the aforementioned research works however do bear a common limitation: their optimization decisions are *server-autocratic* by relying on server-accessible information—yet being agnostic to the *intra-round* system and statistical status on individual clients. Admittedly, it is privacy-vulnerable and also inconvenient to sense intra-round training status on clients, but such intra-round status is indeed an informative observation angle with the potential to enable flexible efficiency optimizations. For example, a FL client may suddenly slow down in the midst of a training round, and intra-round status perception can immediately trigger straggler-avoiding remediations. Meanwhile, intra-round status perception can help identify the parameters attaining early convergence prior to round completion, whose updates can be immediately transferred to the FL server—a white-box method that can enable computation-communication overlap without incurring model staleness. Nonetheless, such intra-round information—although possessing great potential to further enhance FL efficiency—can only be accessed by the clients themselves; to make use of intra-round information in reality, we must allow clients to autonomously adjust the local training strategy based on the iteration-level execution status.

Our objective. In this paper, we plan to introduce *client autonomy* to unleash the power of intra-round system and statistical information in optimizing FL efficiency, i.e., attaining fast training convergence. In that regard, we further seek to design effective solutions to respectively alleviate the straggler problem and shorten the communication time in the critical path. In the next section, we will empirically study the intra-round training characteristics and explore the potential efficiency optimization opportunities therein.

3 MOTIVATION

3.1 The Need to Monitor Intra-round Status

With client autonomy, each FL client—based on its instantaneous execution status—can independently adjust the local computation and communication strategies for better efficiency. To design effective strategies, we first need to study the typical intra-round execution pattern in both system and statistical aspects.

We first take a look at the intra-round *system* characteristics. It is straightforward to know that the resource quality of an individual client is dynamic during the training process. In that regard, existing studies [3, 14, 36] nonetheless mainly focus on the persistency level (i.e., duration) of client availability, yet our focus here is to understand the level of resource quantity fluctuations of the available clients. While traces including such fine-grained hardware status are still absent in the literature, we find client drop-out can be viewed as an extreme case of shrinking resource quantity, whose frequency can implicitly reveal the intensity of resource fluctuation. As reported by a prior study on client availability [36], which collected device behavior data spanning across 136K users over a week, about 70% of the clients are available for at most 10 minutes (in a similar order of magnitude to the duration of a typical FL training round). Therefore, it can be expected that the resource quantity of a FL client often changes in the midst of a round.

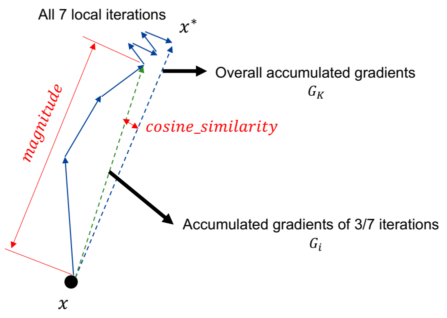

All 7 local iterations
x^*
Overall accumulated gradients
G_K
magnitude
cosine_similarity
Accumulated gradients of 3/7 iterations
G_i
x

Figure 1: In our statistical progress metric, we combine cosine similarity with magnitude similarity to depict the quality of the accumulated gradients. In general, the early iterations of a round usually yield faster statistical progress enhancement.

We then turn to intra-round *statistical* characteristics, which is more intriguing and significant. In studying statistical pattern, we seek to quantify the statistical contributions of different local iterations in a round, which—as we show later—are nonuniform as opposed to the uniformity assumption of FedAda [39]. Note that, each client makes its voice in FL by reporting the accumulated local update after each round; when clients autonomously adjust their local training strategy, we need to try preserving the accumulated local updates of each round. Therefore, the knowledge of *how each individual iteration matters for the accumulated local updates* is crucial for making intra-round optimization decisions: first, for computation optimization, when mitigating stragglers by reducing the local iteration number, clients need to consider its negative impact on statistical performance; in the meantime, to improve communication efficiency, clients also need to rely on intra-round statistical pattern to identify those early-converged layers for eager transmission. Next, we will devise a metric to help analyze intra-round statistical pattern, and then make a series of empirical measurements to depict the typical statistical patterns.

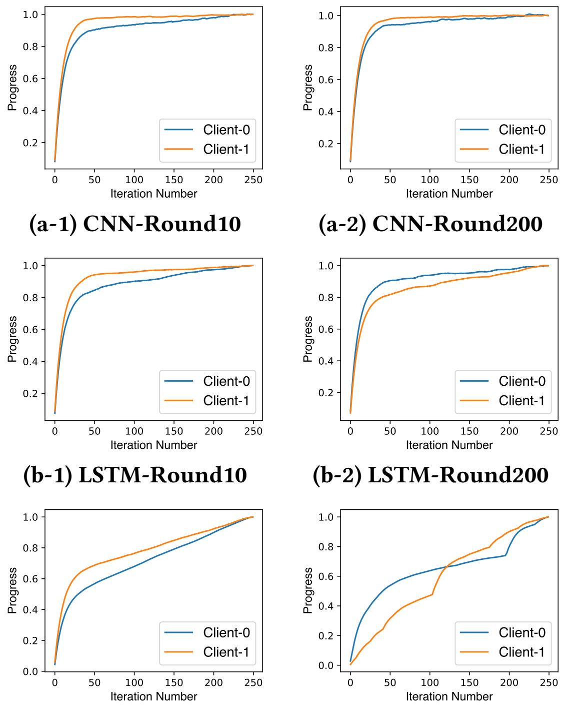
3.2 Intra-round Statistical Pattern Deep Dive

3.2.1 *Statistical Progress: The Metric.*

As aforementioned, to study intra-round statistical pattern on clients, we should design a metric to quantify—after each local iteration—the similarity between the accumulated update so far and the expected update after the standard round. We choose to directly calculate the similarity of the two vectors instead of using implicit metrics like local loss values or test accuracies, because the latter ones do not support flexible analysis at per-layer granularity, and they are meanwhile unstable (if using loss) or compute-intensive (if using test accuracy).

We then describe our proposed metric, *statistical progress*. Regarding similarity quantization of two vectors, the *cosine similarity* metric is often adopted. Yet, cosine similarity itself can not reveal the magnitude gap of two vectors, and our proposed metric chooses to combine both cosine similarity and magnitude similarity. To be concrete, we assume[1] that each client by default needs to train

[1]While for simplicity we assume identical local iteration number across different clients, our solutions can be easily extended to more general cases with heterogeneous

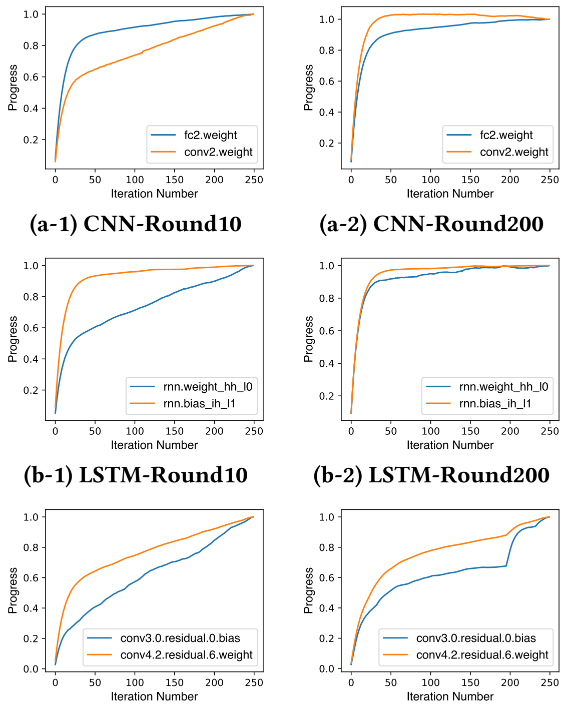


(a-1) CNN-Round10 (a-2) CNN-Round200

(b-1) LSTM-Round10 (b-2) LSTM-Round200

(c-1) WRN-Round10 (c-2) WRN-Round200

Figure 2: The statistical progress curves on two clients at different stages when training CNN, LSTM and WRN.



(a-1) CNN-Round10 (a-2) CNN-Round200

(b-1) LSTM-Round10 (b-2) LSTM-Round200

(c-1) WRN-Round10 (c-2) WRN-Round200

Figure 3: The statistical progress curves for different layers at different stages when training CNN, LSTM and WRN.

for K local iterations in each round, and we let G_i represent the accumulated local update after training for i iterations. In that case, G_K is the overall accumulated update expected to get from that round. Then, our statistical progress metric, P_i, is defined as:

$$P_i = Sim_{cos}(G_i, G_K) \cdot \frac{\min(\|G_i\|, \|G_K\|)}{\max(\|G_i\|, \|G_K\|)}, \quad (1)$$

where the first item is the cosine similarity between G_i and G_K (each flattened to a vector), and the second item is their relative magnitude gap. Note that the value of P_i is always less than 1. Obviously, with more training iterations (i.e., with larger i), G_i would be more closer to G_K, rendering P_i closer to 1. Moreover, with the statistical progress metric, we can further judge the statistical contribution of iteration i as $P_i - P_{i-1}$.

Fig. 1 shows a toy example on how the local gradients are accumulated in a round. During the local training iterations, each client is essentially refining the model parameters towards its local optimum, and the statistical contributions of different iterations are usually not the same: at the early stage of a local round, the model parameters are usually far from the local optimum, yielding large and consistent refinement steps (i.e., updates); as the model parameters get closer to the local optimum, later iterations tend to yield more tiny and conflicting refinement steps. Therefore, although the round shown in Fig. 1 has 7 iterations, the accumulated gradient after 3 iterations is already very close to that of the entire round, indicating that the statistical progress usually increases rapidly in the beginning yet only marginally in later iterations. Next, we verify that with testbed measurements.

3.2.2 *Empirical Observations.*

To empically study the intra-round statistical characteristics, we build a small-scale cluster with 4 client nodes and 1 server node. With that cluster we train three models: CNN [16], LSTM and WideResNet28 (WRN) [37], and their datasets are CIFAR-10 [13], KWS (Keyword Spotting dataset for speech commands recognition [32]), and CIFAR-100, respectively. To emulate non-IID data distribution which is typical in FL, we have the class composition of each client's local dataset follow a distinct Dirichlet distribution, where the concentration hyper-parameter α is set to 0.1. Meanwhile, the number of local iterations in a round is fixed to 250 and we train each model for 200 rounds. For the other hyper-parameters, please refer to Sec. 5.1.

For each case, we measure the instantaneous statistical progress (i.e., the P_i value as defined in Eq. 1) attained after each local iteration. Note that calculating P_i requires the knowledge of G_K—which is only available after round completion; we thus record all the parameter snapshots after each iteration for end-of-round $\{P_i\}_{i=1}^{K}$ calculation. In Fig. 2, we show the statistical progress curves of each model from different clients (Client-0 and Client-1, which are randomly selected) and also at different training stages (round-10 and round-200, the former representing the early stage and the latter representing the late one). Further in Fig. 3, we show the statistical progress curves for each model across different layers (the

local iteration numbers (which is determined by the local dataset size and the batch size). Meanwhile, while it is the model *parameters* that are synchronized under FedAvg, our analysis focuses primarily on the model *update* (i.e., the gap between two model versions), which are essentially equivalent. Besides, in this paper we interchangeably use "gradient" (with the learning rate factor integrated in) and "update".

definition of P_i can be naturally narrowed down for each individual layer). From those figures, we can learn that the statistical progress curves exhibit the following characteristics:

1) Diminishing marginal benefit. From Fig. 2 and Fig. 3, we can see that there is a sharp increase for each curve during the earlier iterations of each round, and the slopes of those curves soon diminish as local training proceeds. For example, for the CNN model, during a round with totally 250 iterations, the statistical progress has already reached 80% after merely 20 iterations. Such a pattern of diminishing marginal benefit is consistent with the illustration in Fig. 1, confirming that the statistical contributions of different iterations within a round are indeed not identical. In particular, as shown in Fig. 3, some layers may converge (i.e., with P_i be nearly 1.0) quite early[2] in a round—like the `conv2.weight` layer of CNN at round 200, or the `rnn.weight_hh_l0` layer of LSTM at round 10. For such layers, by eagerly transmitting their updates to the FL server, it is possible to mitigate the end-of-round communication bottleneck without accuracy degradation.

2) Heterogeneity across stages, clients and layers. In both Fig. 2 and Fig. 3, by comparing the curves at round-10 with those at round-200, we can learn that the statistical patterns exhibit salient heterogeneity across different temporal stages. Meanwhile, within each case of Fig. 2, the curves from different clients are non-overlapping, indicating the existence of cross-client statistical heterogeneity. Finally, as shown in Fig. 3, the statistical progress pattern also varies across different layers. To summarize, the intra-round statistical progress curve is distinct to each *stage*, to each *client*, and also to each *layer*; it is impossible to create a one-size-fit-all model to analytically depict those curves.

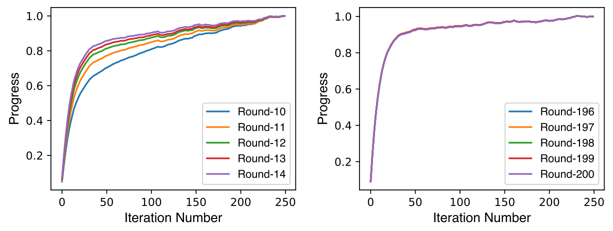
The empirical study above signifies it as a promising way to exploit intra-round *statistical* pattern for efficiency optimizations, which, however, must be conducted by each client independently at runtime. This also holds for exploiting the intra-round *system* status as previously discussed in Sec. 3.1. Therefore, contrary to the current *server-autocratic* nature, we need to grant FL clients sufficient *autonomy* to leverage the instantaneous system and statistical status for better computation and communication efficiency.

4 SOLUTION

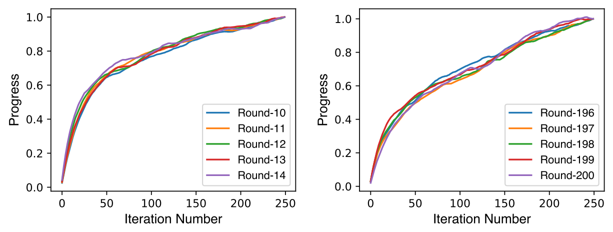
In this paper, we propose *Federated Learning with Client Autonomy* (FedCA), a novel mechanism that enables each client to conduct flexible intra-round optimizations. In principle, based on the instantaneous system and statistical characteristics, clients under FedCA can autonomously adjust the number of local iterations in an ongoing round for better computation efficiency, and can also eagerly transmit the early-converged layers prior to round completion for better communication efficiency.

To be more concrete, our FedCA design is mainly composed of three parts: 1) *profiling part*—To obtain the statistical progress curves of each round in advance and also in a resource-efficient manner, we propose a practical profiling mechanism called *periodical sampling*; 2) *computation-optimization part*—To mitigate stragglers with minimum accuracy degradation, we devise a utility function called *net benefit* to help clients determine whether to terminate the local computations of a round; 3) *communication-optimization part*—To enable computation-communication overlap without compromising accuracy, we propose a mechanism called *eager transmission with error feedback*. Next, we will respectively elaborate the detailed solution for each part.

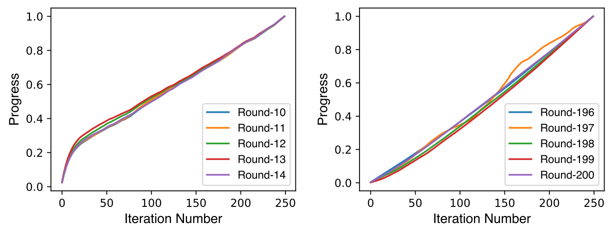
[2]This is indeed reasonable as existing studies [5, 8, 38] have shown that, some input features may be easier to learn than others, and the parameters related to those features would converge in fewer iterations—a property called *non-uniform convergence* [35].



(a) CNN



(b) LSTM

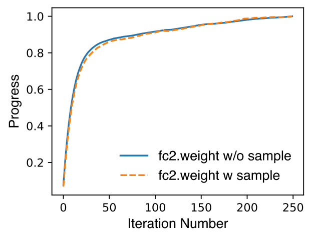


(c) WRN

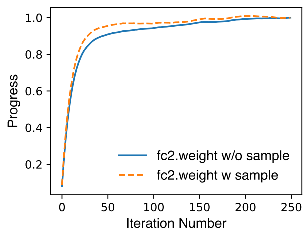
Figure 4: When training each model, the statistical progress curves are very similar across consecutive rounds, for both an early stage (round 10-14) and a late one (round 196-200).

4.1 Efficient Profiling with Periodical Sampling

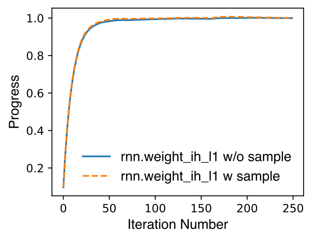
Recall that in Sec. 3.2, to get the statistical progress curves (shown in Fig. 2 and Fig. 3), we need to record the accumulated gradient values after each iteration of a round. Nevertheless, recording those fine-grained snapshots would incur remarkable memory overhead. To illustrate, consider training with WRN-28 (36 million parameters): assuming 4 bytes for each parameter, a round with 100 local iterations would consume a memory space of around 14GB on each client solely for profiling purposes. Meanwhile, as suggested in Eq. 1, calculating statistical progress requires the knowledge of G_K, which is available only after round completion; however, clients under FedCA must have the statistical progress curve readily available before round commencement. In a nutshell, to enable FedCA, the statistical curves must be obtained *efficiently* and also *a priori*. To that end, observing the gradient trajectory similarity across consecutive rounds and also across parameters within a layer, we propose *periodical sampling*, a composite mechanism composed of periodical profiling and intra-layer sampling.



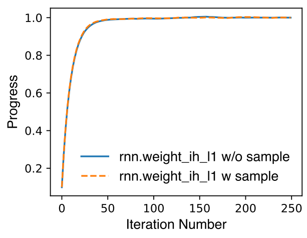
(a-1) CNN-Round10



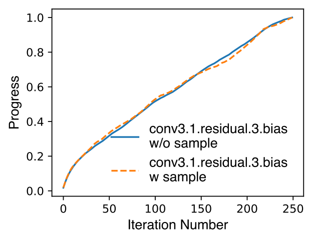
(a-2) CNN-Round200



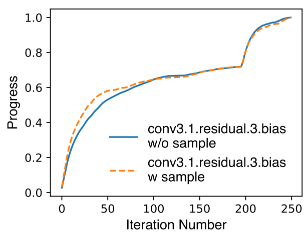
(b-1) LSTM-Round10



(b-2) LSTM-Round200



(c-1) WRN-Round10



(c-2) WRN-Round200

Figure 5: For a randomly selected layer in each model, the statistical progress curves profiled with the sampled parameters and with all the parameters are quite similar.

Periodical profiling. We first propose periodically estimating a client's statistical progress curve, such as once for every five rounds, rather than doing so for every round. Our key insight is that a client's intra-round statistical progress pattern usually shows strong similarity across consecutive training rounds. As depicted in Fig. 4 which corresponds to a randomly selected client in the aforementioned testbed (Sec. 3.2.2), the progress curves for five consecutive rounds exhibit high resemblance both in the early stage (rounds 10-14) and the late one (rounds 196-200). Such similarity is expected since the global model and clients' data evolve smoothly over time [21]. With periodical profiling, we can use the progress curve profiled at an anchor round[3] for the subsequent rounds, alleviating the total memory burden and also making the curves available a priori. In practical implementation, the profiling period is a tunable hyperparameter that enables developers to strike an optimal balance between profiling precision and memory usage.

Intra-layer sampling. Aside from temporal optimization, when conducting profiling, we also perform spatial optimization by only recording the updates for a sampled parameter subset in each layer instead of for the entire model. Specifically, for each layer, we randomly choose either 50% or 100 (whichever is smaller) scalar parameters, and use such a compressed subset to represent the original layer in profiling statistical progress. This idea is inspired by the observation that parameters within the same layer tend to evolve at a similar pace (contrary to those across different layers as in Fig. 3). To demonstrate this, in Fig. 5, we respectively show the statistical progress curves profiled with the full layer (in blue solid lines) and then only with the sampled parameters (in orange dashed lines). For each model, across different training stages and layer types, both sets of results consistently align with each other. This allows us to significantly reduce the peak memory usage by sampling. For instance, the total number of parameters considered by our sampling method for WRN will be no more than 10,000, which indicates a peak memory usage of no more than 4MB, in contrast to the earlier mentioned 14GB usage with full profiling.

[3]Note that we do not adopt any computation or communication optimizations in those anchor rounds to ensure the completeness and validity of the profiled curves.

4.2 Early Stopping with Utility Guidance

With statistical progress curves profiled at hand, we can leverage them to optimize the training efficiency. When optimizing the computation efficiency by proactively adjusting the number of local iterations on a straggling client, we must consider its negative impact on the accuracy performance. On one hand, as previously discussed in Sec. 3, a client's progress tends to diminish as the number of iterations increases during a round. On the other hand, training for a larger number of local iterations consumes more time and computing power for the client. These facts imply the presence of a *crossover point* in a client's local training, beyond which the marginal cost exceeds the marginal benefit, rendering subsequent iterations unnecessary. To identify this crossover point to early stop a round, we first quantify the marginal cost and benefit.

Quantifying marginal benefits. In a round R, the marginal benefit of a local iteration τ can be understood as the additional progress a client achieves with that iteration. Thus, we can define the marginal benefit, denoted as $b_{R,\tau}$, as the difference between the statistical progress of iteration τ and $\tau-1$, represented as $P_{R,\tau}$ and $P_{R,\tau-1}$ respectively (Sec. 3.2). Leveraging the similarity exhibited by a client's progress curves across consecutive rounds (Sec. 4.1), we approximate the marginal benefit by referring to the most recently profiled anchor round T. In particular, note that the statistical progress curves may not always be rigidly concave (as shown in Fig. 5), we set a lower bound to tackle the potential curve irregularity. That lower bound is set as the expected per-iteration improvement of statistical progress over all the remaining iterations. Putting it all together, we estimate $b_{R,\tau}$ as:

$$b_{R,\tau} \triangleq max(P_{T,\tau} - P_{T,\tau-1}, \frac{1-P_{T,\tau}}{K-\tau}). \tag{2}$$

Quantifying marginal costs. Intuitively, the marginal cost, denoted as $c_{R,\tau}$, should be defined as a non-decreasing function in the wall clock time a client has spent on local training of a round R up to iteration τ, i.e., $t_{R,\tau}$. However, simply defining $c_{R,\tau}$ as the identity function of $t_{R,\tau}$ overlooks the synchronous nature of standard FL training. In synchronous FL, the server must wait for all selected clients to complete their local training before it proceeds to model aggregation. Therefore, it is not beneficial for a client to prematurely stop its training when the majority of other clients have not finished. In other words, there should be a certain time point determined by the pace of the majority of clients. Below this point, the marginal cost should increase slowly over iterations; while beyond it, the cost should rapidly rise to penalize stragglers. In this regard, we formulate the marginal cost of local training as

$$c_{R,\tau} \triangleq f_{R,\tau} \cdot \frac{t_{R,\tau}}{T_R} \quad \text{with} \quad f_{R,\tau} = \begin{cases} \beta, & t_{R,\tau} \le T_R \\ 1, & \text{otherwise} \end{cases}, \tag{3}$$

where T_R is the desired deadline for round R that allows the majority of selected clients to complete their local training, and $\beta \ll 1$ depicts the marginal cost ratio before T_R. Inspired by the deadline setup strategy in FedBalancer [28], we determine T_R by maximizing the *ratio* of the estimated number of clients that can finish before T_R to T_R itself. This way, T_R will neither be too high to discourage the early stopping of clients, nor too low to collect enough local updates for model aggregation.

Navigating the cost-benefit tradeoff with a utility function. Based on the estimation methodology of marginal benefit and cost, it becomes possible to determine the point at which the latter surpasses the former. Specifically, for each iteration τ of a client's local training in round R, we define a utility function called *net benefit*, represented as $n_{R,\tau}$, to guide the client to adjust the local iteration number. Given that $b_{R,\tau}$ and $c_{R,\tau}$ are dimensionless quantities respectively depicting the relative benefit and cost of a single iteration compared to the entire round, we can define $n_{R,\tau}$ by directly calculating their gap, i.e.,

$$n_{R,\tau} \triangleq b_{R,\tau} - c_{R,\tau}. \tag{4}$$

With this definition, each client can terminate the local training as soon as $n_{R,\tau}$ turns negative. In this way, our FedCA design imposes minimal or negligible impact on the overall quality of the global model, while enhancing the computation efficiency. We validate this with empirical experiments (Sec. 5).

4.3 Eager Transmission with Error Feedback

In Sec. 4.2, to enhance computation efficiency, we have explored the idea of early stopping when the marginal cost of training for an additional local iteration begins to outweigh the marginal benefit; all the analysis and actions in that part is at the granularity of the *entire model*. Recall that upon examination at a finer-grained level (as shown in Fig. 3), we have learned that each layer may exhibit a unique pace of evolution. By eagerly transmitting the early-converged layers, we can overlap their communication latency with the computation time of other layers, thereby enhancing the communication efficiency. In the meantime, we also need to ensure that the model training quality is not compromised by our communication optimization strategy.

Layerwise eager transmission upon stabilization. Specifically, we assess the progress of each model layer l during local training in round R up to local iteration τ, denoted as $P^{(l)}_{R,\tau}$. The method used for assessment is the same as introduced in Sec. 3.2, with the only difference being that we here focus on the parameters of a specific layer rather than of the entire model. When the progress metric $P^{(l)}_{R,\tau}$ surpasses a predefined threshold $T_e \le 1$, we consider the corresponding layer l as starting to stabilize. For such stabilized

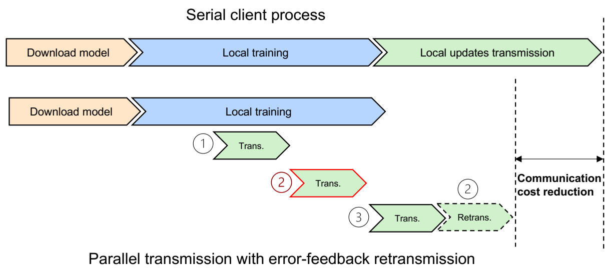

Serial client process
Download model
Local training
Local updates transmission
Download model
Local training
1
Trans.
2
Trans.
2
3
Trans.
Retrans.
Communication cost reduction
Parallel transmission with error-feedback retransmission

Figure 6: In our FedCA design, clients can eagerly transmit the updates of the early-converged layers to the FL server, which can enable computation-communication overlap and reduce the communication time in the critical path. Note that layer-2 is retransmitted to ensure accuracy validity.

layers, as shown in Fig. 6, we can eagerly transmit their updates to the server to hide the transmission latency behind the local computation time of the other unstable layers. In this way, we can reduce the update transmission time in the critical path. Note that, similar to Sec. 4.2, the statistical progress metric of a layer l—$P^{(l)}_{R,\tau}$—can only be computed retrospectively; we therefore propose utilizing the most recently profiled curve from round T for round R. In other words, the eager transmission of layer l in round R occurs in iteration τ when

$$P^{(l)}_{R,\tau} \triangleq P^{(l)}_{T,\tau} > T_e. \tag{5}$$

Retransmission upon significant deviation. Admittedly, while there exists strong similarity among the statistical progress curves of the anchor round and its subsequent rounds, it is still possible that the statistical progress curve of an ongoing round turns out divergent from the previously-profiled anchor round. To address the adverse impact of such curve divergence on model accuracy, we further introduce an error-feedback mechanism. Specifically, once the training of all layers is halted at iteration F (either due to natural completion or early stopping as proposed), we check the ultimate updates of the layers whose updates have been eagerly-transmitted. Given a layer l, supposing its update was eagerly transmitted at iteration $t_{R,l}$, we let $G^{(l)}_{R,F}$ denote the ultimate update and $G^{(l)}_{R,t_{R,l}}$ the reported one. If their cosine similarity falls below a predefined threshold $T_r < 1$, i.e.,

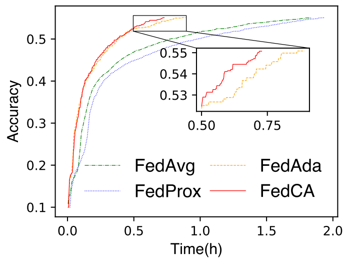
$$Sim_{\cos}(G^{(l)}_{R,F}, G^{(l)}_{R,t_{R,l}}) < T_r, \tag{6}$$

indicating that the layer has significantly deviated from the time it was eagerly transmitted, the client needs to retransmit layer l together with the other regular layers, as also shown in Fig. 6. In a word, we expect to optimize the communication efficiency with no or mild loss in the final model utility.

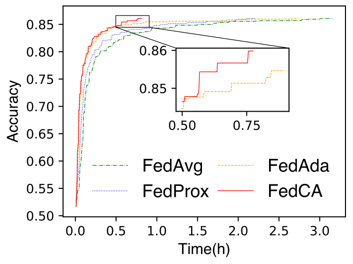
5 EVALUATION

5.1 Experimental Setup

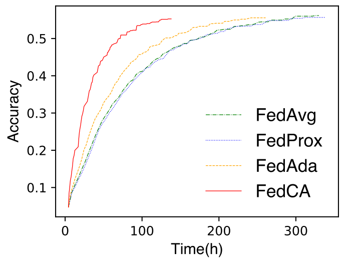
Implementation. We have implemented FedCA atop PyTorch, and the server communicates with clients via RPyC (Remote Python Call) [1]. After each local iteration, the clients call two functions, `TryEarlyStop()` and `TryEagerTransmit()`, to launch the computation and communication optimizations as elaborated in Sec. 4.2 and Sec. 4.3. The eagerly-transmitted updates are communicated in



(a) CNN



(b) LSTM



(c) WRN

Figure 7: Time-to-Accuracy curves when training CNN, LSTM and WRN respectively under the four schemes.

a separated thread with the `threadings.Thread()` API. After each round, each client calls `TryRetransmit()` to rectify the inaccurate updates transmitted before. Besides, at the start of each round, the FL server needs to—together with the latest parameters—also offload the expected deadline (T_R in Eq. 3) to each client. The other procedures are identical with standard FedAvg.

Hardware setup. We build an EC2 cluster with 128 `c6i.large` instances and one `c5a.8xlarge` instance. Each `c6i.large` instance has 2 vCPUs and 4GB RAM (similar to a smart phone), working as a FL client. Following the average network condition of FedScale [14]—a comprehensive FL benchmark including real-world mobile device measurements, we set the link bandwidth of each client to 13.7 Mbps (with the wondershaper [2] tool). Meanwhile, the `c5a.8xlarge` instance—which has 32 vCPUs, 64GB RAM and 10 Gbps link bandwidth—works as the FL server.

We further emulate the heterogeneity and dynamicity properties of FL. To emulate heterogeneity, we randomly map each client to a device in the FedScale trace [14], and have the ratio between any two clients' average speeds resemble that of their FedScale counterparts. To emulate dynamicity, we let each client keep toggling between *fast* and *slow* modes; the duration (in seconds) of each fast and slow period is respectively determined by sampling from two gamma distributions—$\Gamma(2, 40)$ and $\Gamma(2, 6)$. We choose gamma distribution because the device availability duration disclosed in FedScale trace exhibits a similar distribution shape. In the slow mode, the slowdown ratio is randomly sampled from a uniform distribution U(1, 5). To realize the target speed for each client, we inject a sleeping period with proper length after each local iteration.

Workload setup. Models trained in our experiments are LeNet-5 [16], LSTM and WideResNet28-10 [37], and their datasets are CIFAR-10 [13], KWS [32] and CIFAR-100 [13], respectively. The data distribution also follows a Dirichlet distribution identical with that in Sec. 3.2.2. We use the SGD optimizer for all models; the learning rates are set to 0.01, 0.05 and 0.1, and the weight decays are set to 0.01, 0.01, 0.0005, respectively. Besides, the batch size is 50 and the number of local iterations in a round is 125. In each round, the FL server waits for 90% of the updates returned the earliest. We train each model towards a near-optimal accuracy (0.55 for CNN, 0.85 for LSTM, and 0.55 for WRN).

Algorithm setup. We compare our FedCA scheme with three baselines: FedAvg [22], FedProx [19] and FedAda [39]. FedProx introduces an additional regularization term in the loss function, and the weight of the regularization item (μ) is set to the recommended value 0.01. FedAda proposes a server-determined workload tuning method assuming homogeneous statistical contribution for each iteration, and the trade-off factor between computation cost and statistical benefit is set to the recommended value 0.5. For FedCA, we set the profiling frequency (Sec. 4.1) to once after every 10 rounds, and set the marginal cost ratio (β in Eq. 3) to 0.01; we also set the triggering thresholds of eager-transmission (T_e in Eq. 5) and retransmission (T_r in Eq. 6) respectively to 0.95 and 0.6.

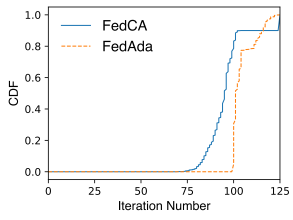
Table 1: Time to reach the target accuracy. The accuracy target for each model is listed below the model name.

Model	Scheme	Per-round Time (s)	# of Rounds	Total Time (h)
CNN (0.55)	FedAvg	16.7	385	1.79
	FedProx	16.7	407	1.89
	FedAda	6.86	453	0.86
	FedCA	5.34	481	0.71
LSTM (0.85)	FedAvg	33.2	177	1.63
	FedProx	33.2	139	1.28
	FedAda	13.1	207	0.75
	FedCA	9.76	228	0.61
WRN (0.55)	FedAvg	15833	65	285.9
	FedProx	15837	67	294.7
	FedAda	11138	74	228.9
	FedCA	4507	100	125.2

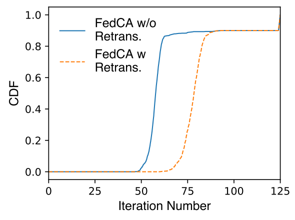
5.2 End-to-End Performance

In Fig. 7, we show the time-to-accuracy curves when training the three models above. In each case, FedCA can always make the most rapid accuracy enhancement. Particularly, for WRN which is the largest one among the three models (the parameter quantities of CNN, LSTM and WRN are respectively 60K, 50K and 36M), the performance benefit of FedCA is the most significant.

In Table 1 we further showcase the total time to reach a near-optimal accuracy target, accompanied with the number of rounds and the average per-round time of the three models. For each case, we can see that FedCA can substantially reduce the per-round time with intra-round computation and communication optimizations, and—-although mildly inflating the number of rounds taken—still reduce the overall convergence time by over 15%. In particular, for the WRN model, FedCA can attain an efficiency enhancement of 45.3% compared to the second best method (FedAda).



(a) Early-stop



(b) Eager-Transmission

Figure 8: The CDF of the iteration number at which early-stop or eager-transmission is triggered for CNN.

5.3 FedCA Behavior Deep Dive

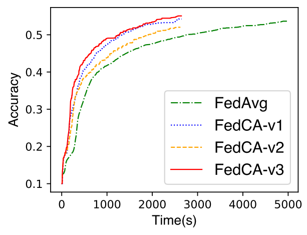
We further take a closer look at the behaviors of FedCA at runtime. In Fig. 8, we show—for the CNN model—the cumulative distribution function (CDF) of the iteration number at which early-stop or eager-transmission is triggered. In Fig. 8a, we first show the CDFs depicting the early-stop moments respectively under FedCA and FedAda. It reveals that some clients stop their local computation as early as at iteration 70 (totally 125 rounds), and the action moments for FedCA are in general earlier than that of FedAda (a natural effect of diminishing marginal benefit as elaborated in Sec. 3.2.2). Regarding communication optimization, in Fig. 8b, we show the CDFs of the eager-transmission moments for all the layers with and without retransmission (for a layer that is retransmitted, we record its action moment as at the last iteration). From Fig. 8b, we can learn that many layers converge early at around iteration 50; meanwhile, while retransmission postpones the effective eager-transmission moments in general, the overall speedup is still very salient.

5.4 Ablation Studies

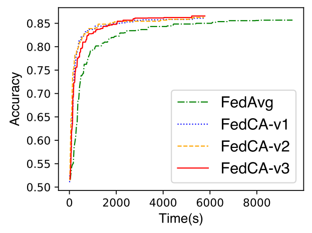
In Fig. 9, we conduct ablation studies on different solution modules of FedCA. We respectively train CNN and LSTM with different versions of FedCA: FedCA-v1—with only the early-stop mechanism, FedCA-v2—with both early-stop and eager-transmission but without retransmission, and FedCA-v3—the standard version with all the functionalities enabled. As shown in Fig. 9, compared with FedAvg, the performance of FedCA-v1 itself is already good enough, demonstrating the effectiveness of early-stopping in tackling resource fluctuation. Meanwhile, regarding eager transmission, we note that its benefit is more salient in the later stage—given an accuracy target of 0.54 (CNN) and 0.86 (LSTM), FedCA-v3 can attain a speedup of 18.8% and 45.5% compared with FedCA-v1. Specifically, given that the curve of FedCA-v2 exhibits a remarkable accuracy loss compared with FedCA-v3, we confirm that the retransmission mechanism is indispensable for eager-transmission.

5.5 Sensitivity Analysis and Overheads

We have introduced several hyper-parameters in FedCA—for the early-stop part, we scale down the time cost before the deadline with a factor β (set to 0.01); for the eager-transmission part, we adopt the early-convergence threshold (T_e) and retransmission threshold (T_r). Here we change those hyperparameters and train CNN for 200 rounds under each setup. In Fig. 10a, the FedCA performance with $\beta = 0.001$ is similar with the default case, whereas setting $\beta = 0.1$—which would excessively discourage pre-deadline computations—does incur a slowdown. In Fig. 10b, the most rigid combination

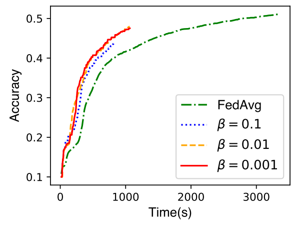


(a) CNN

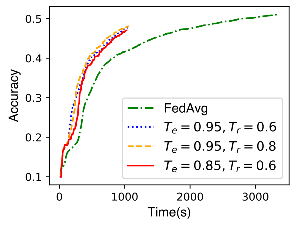


(b) LSTM

Figure 9: Ablation study with FedAvg, FedCA-v1 (with early-stop), FedCA-v2 (with early-stop & eager-transmission but without retransmission) and FedCA-v3 (standard FedCA).



(a) Varying β



(b) Varying T_e and T_r

Figure 10: Sensitivity analysis of the marginal cost ratio (β) and the eager-transmission thresholds (T_e and T_r) for CNN.

($T_e = 0.95$ and $T_r = 0.8$) is slightly better than the others, yet in general, the FedCA performance is stable across different setups.

We have also measured the memory overhead of FedCA. The number of sampled parameters when profiling CNN, LSTM, and WRN are 618, 905 and 9974—with the additional memory cost be 0.24MB, 0.34MB, and 3.8MB, respectively. Such overheads are negligible compared to the sizes of the original models (for example, WRN has a model size of 139.4MB).

6 OTHER RELATED WORKS AND DISCUSSION

In this section, we list some other related works (additional to Sec. 2.2) and discuss the future work.

Improving training efficiency. To tackle the straggler problem, apart from client selection [3, 15], partial aggregation [22, 28] and workload adjustment [18, 39] (elaborated in Sec. 2.2), some other works [9, 10, 23] propose to enforce asynchronous training, in which each client can proceed independently without waiting for others. Yet, asynchronous updating may incur stale parameters and compromise the training accuracy. Meanwhile, regarding improving the communication efficiency, aside from the methods elaborated in Sec. 2.2 (quantization [4, 12], sparsification [5, 8, 21] and frequency tuning [6, 17, 30]), Liu et al. [20, 31] proposed a hierarchical structure to mitigate the network contention at the server side, and some others [7, 27] proposed to prioritize the gradient transmission of different layers by their order to be consumed in the future. These methods are orthogonal to FedCA.

Improving FL accuracy. While we focus on improving FL efficiency, given salient data heterogeneity, it is also a hot topic to improve the accuracy performance of FL. In that regard, client selection methods like [3, 15] also consider the data quality as part of the selection criterion. Meanwhile, some works seek to reduce the level of data heterogeneity by maintaining a few common data samples to each client [29, 40]; some other works propose to rectify the optimization process by adding an extra regulation item to the loss function [19] or revising the vanilla gradient based on the global update direction [11].

Discussions on future work. The concept of client autonomy can be extended to other optimization measures. For example, the recent years have witnessed the revival of optimizing traditional hyper-parameters—like learning rate [25], momentum [34] and batch size [26]—for better FL efficiency; clients may also need to *autonomously* adjust these hyper-parameters within a training round. We plan to explore such directions in the future.

7 CONCLUSION

In this work, to improve FL efficiency, we propose FedCA, a novel mechanism that grants clients the autonomy to conduct intra-round training optimizations. We first propose the periodical sampling method to acquire the statistical progress curves efficiently at runtime, and then devise a utility function to help reduce the workloads on potential stragglers with minimum statistical degradation. Finally, for communication acceleration, clients under FedCA can eagerly transmit the early-converged layers to enable computation-communication overlap, with a retransmission mechanism to ensure convergence validity. Extensive experiments show that FedCA can enhance FL efficiency by over 15%.

ACKNOWLEDGEMENT

This work is supported by National Natural Science Fundation of China (NSFC) (62202300) and Shanghai Pujiang Program (22PJ1404600). Na Lyu and Zhi Shen are co-first authors. Chen Chen and Minyi Guo are the corresponding authors.

REFERENCES

[1] [n. d.]. RPyC. https://rpyc.readthedocs.io/en/latest/.
[2] 2020. wondershaper. https://github.com/magnific0/wondershaper.
[3] Ahmed M Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A Fahmy. 2023. Refl: Resource-efficient federated learning. In *ACM Eurosys*.
[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NeurIPS*.
[5] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. 2021. Communication-efficient federated learning with adaptive parameter freezing. In *IEEE ICDCS*.
[6] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. 2023. GIFT: Toward Accurate and Efficient Federated Learning With Gradient-Instructed Frequency Tuning. *IEEE Journal on Selected Areas in Communications* 41, 4 (2023), 902–914.
[7] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. 2019. Tictac: Accelerating distributed deep learning with communication scheduling. In *MLSys*.
[8] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia:Geo-Distributed machine learning approaching LAN speeds. In *USENIX NSDI*.
[9] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *MLSys* (2022).
[10] Zhifeng Jiang, Wei Wang, Baochun Li, and Bo Li. 2022. Pisces: efficient federated learning via guided asynchronous training. In *ACM SoCC*.
[11] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *ICML*.
[12] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492
[13] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[14] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *ICML*.
[15] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *USENIX OSDI*.
[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[17] Sunwoo Lee, Tuo Zhang, and A Salman Avestimehr. 2023. Layer-wise adaptive model aggregation for scalable federated learning. In *AAAI*.
[18] Li Li, Haoyi Xiong, Zhishan Guo, Jun Wang, and Cheng-Zhong Xu. 2019. SmartPC: Hierarchical pace control in real-time federated learning system. In *IEEE RTSS*.
[19] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *MLSys* (2020).
[20] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2020. Client-edge-cloud hierarchical federated learning. In *IEEE ICC*.
[21] WANG Luping, WANG Wei, and LI Bo. 2019. CMFL: Mitigating communication overhead for federated learning. In *IEEE ICDCS*.
[22] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).
[23] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated Learning with Buffered Asynchronous Aggregation. In *AISTATS*.
[24] Takayuki Nishio and Ryo Yonetani. [n. d.]. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE ICC*.
[25] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020).
[26] Dian Shi, Liang Li, Maoqiang Wu, Minglei Shu, Rong Yu, Miao Pan, and Zhu Han. 2022. To talk or to work: Dynamic batch sizes assisted time efficient federated learning over future mobile edge devices. *IEEE Transactions on Wireless Communications* 21, 12 (2022), 11038–11050.
[27] Shaohuai Shi, Xiaowen Chu, and Bo Li. 2019. MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 172–180.
[28] Jaemin Shin, Yuanchun Li, Yunxin Liu, and Sung-Ju Lee. 2022. FedBalancer: data and pace control for efficient federated learning on heterogeneous clients. In *ACM Mobisys*.
[29] Zhenheng Tang, Yonggang Zhang, Shaohuai Shi, Xin He, Bo Han, and Xiaowen Chu. 2022. Virtual Homogeneity Learning: Defending against Data Heterogeneity in Federated Learning. In *ICML*.
[30] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE journal on selected areas in communications* 37, 6 (2019), 1205–1221.
[31] Zhiyuan Wang, Hongli Xu, Jianchun Liu, He Huang, Chunming Qiao, and Yangming Zhao. 2021. Resource-efficient federated learning with hierarchical aggregation in edge computing. In *IEEE INFOCOM*.
[32] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).
[33] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis. 2020. SAFA: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Trans. Comput.* 70, 5 (2020), 655–668.
[34] Xidong Wu, Feihu Huang, Zhengmian Hu, and Heng Huang. 2023. Faster adaptive federated learning. In *AAAI*.
[35] Eric P Xing, Qirong Ho, Pengtao Xie, and Dai Wei. 2016. Strategies and principles of distributed machine learning on big data. *Engineering* 2, 2 (2016), 179–195.
[36] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. 2021. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *WWW*.
[37] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).
[38] Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).
[39] Jinghui Zhang, Xinyu Cheng, Cheng Wang, Yuchen Wang, Zhan Shi, Jiahui Jin, Aibo Song, Wei Zhao, Liangsheng Wen, and Tingting Zhang. 2022. FedAda: Fast-convergent adaptive federated learning in heterogeneous mobile edge computing environment. *World Wide Web* 25, 5 (2022), 1971–1998.
[40] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).