Flexible Training and Uploading Strategy for Asynchronous Federated Learning in Dynamic Environments

Mengfan Wu, *Student Member, IEEE*, Mate Boban, *Senior Member, IEEE*, and Falko Dressler *Fellow, IEEE*

Abstract—Federated learning is a fast-developing distributed learning scheme with promising applications in vertical domains such as industrial automation and connected automated driving. The heterogeneity of devices in data distribution, communication, and computation, when deployed in dynamic environments typically with wireless communication, poses challenges to traditional federated learning solutions, where successful learning depends on balanced contribution from participants. In this paper, we propose a flexible communication strategy for devices in asynchronous federated learning, which adapts the training and uploading actions based on the condition of the communication link. We propose a novel method of computing aggregation weight based on model distances and number of local optimizations, to control errors introduced in asynchronous aggregation while maximizing learning speed. We prove the convergence of the learning tasks analytically under the new scheme. The improved performance is rooted in the increased number of optimizations during training, which grows by 12% through opportunistically condensing model uploading during good link condition periods. By facilitating timely communication between devices and server, combined with the novel aggregation weight design, our method reduces the communication resources in dynamic environments by at least 5% while even slightly increasing the learning accuracy.

Index Terms—Asynchronous federated learning, flexible communication strategy, flexible training and uploading, dynamic environments, wireless communications

1 Introduction

Given the continuous increase in computing power of end devices, connecting at the edge and utilizing this computing capacity together has been a promising technical direction to enable fast and ubiquitous computing [1], [2]. To this end, federated learning (FL), in which participating devices learn local machine learning models and then exchange them with other devices to achieve a global model applicable for all devices, is developed and widely applied [3]. Specifically, FL is preferred due to its privacy-preserving nature by sharing learned model rather than local data. Moreover, requirements on communication resources are also reduced, due to the smaller size of model compared with the dataset they are trained on [4].

As FL is deployed in a wider range of scenarios, the heterogeneity of devices' computing and communication capabilities becomes prominent. Heterogeneity in communication conditions could originate from different communication capabilities as well as from the propagation environments of wireless communication where the devices find themselves in [5]. In scenarios where devices communicate over wireless channels to enable mobility, the variability of the devices' link capacity thus poses challenges to the traditional synchronization stage in FL [6], [7]. For example, in case high frequency bands (e.g., mmWave or THz) are used, signal blockage is especially prominent in affecting link capacity [8]. Therefore, FL devices within the line-of-sight (LoS) region will be quick to share the learned models while those in the Non-line-of-sight (NLoS) region will be constrained to share and receive learned knowledge. Besides variability in link condition, the participating devices in FL are often designed with multi-tasking purposes which results in the variability of computing capacity as well. Safety-critical tasks are assigned with higher priority and demand computational resources to be allocated timely [9], which later results in the deceleration of learning tasks for FL. Common strategies to deal with heterogeneity in FL include adaptive resource allocation [10], model compression [11], and stratified client clustering as in hierarchical FL [12]. However, most of the works do not include specific modelling of heterogeneous communication and computation resources. Moreover, when considered, the modelling of communication resources is usually performed coarsely (as in [13]–[15] where resource-related variable possible to be changed per round of learning), thus keeping the adapted actions of clients fixed during the corresponding time period [7].

- *M. Wu (Corresponding Author) is with the Munich Research Center, Huawei Technologies Duesseldorf GmbH, Munich, Germany as well as with the School of Electrical Engineering and Computer Science, TU Berlin, Berlin, Germany. E-mail: mengfan.wu@huawei.com.*
- *M. Boban is with the Munich Research Center, Huawei Technologies Duesseldorf GmbH, Munich, Germany. E-mail: mate.boban@huawei.com.*
- *F. Dressler is with the School of Electrical Engineering and Computer Science, TU Berlin, Berlin, Germany. E-mail: dressler@ccs-labs.org.*
- *This work was supported in part by the Federal Ministry of Education and Research (BMBF, Germany) within the 6G Research and Innovation Cluster 6G-RIC under Grant 16KISK020K and within the framework of the HORIZON-JU-SNS-2022 project TIMES, co-funded by the European Union. The views expressed are those of the authors and do not necessarily represent the project.*

To this end, in our previous work [16], a step-wise FL platform is introduced and allows for accurate realizations of heterogeneity on communication and computation resources. In this paper, to address the heterogeneity issues and promote learning efficiency, we propose a flexible learning and uploading strategy for FL devices to capture a good communication time window and upload the learned model efficiently. In case of a prolonged poor communication conditions, the devices are allowed to extend their training process and wait for better uploading opportunities. The learning progress of devices will also be uneven as a result of the flexible learning strategy. To tackle this problem, we analyze the loss evolution of a global model considering heterogeneous training progress of each uploaded model. Based on the analysis, we propose a new way for computing the aggregation weight at the server, aiming to strike a balance between assimilating knowledge fast as well as limiting the error introduced in aggregation heterogeneous model updates. To the best of our knowledge, our method is the first study to i) opportunistically adapt the training and uploading in asynchronous FL based on the communications constraints while also dynamically computing aggregation weight based on model distances and number of local optimizations, whose increased heterogeneity results from the opportunistic approach; and ii) analytically prove the convergence of the proposed dynamic asynchronous FL task.

We conduct extensive experiments to show that the flexible training and uploading strategy combined with the novel weight design method facilitates the FL tasks in terms of reaching high accuracy in dynamic environments after a fixed learning time. Moreover, the communication resource usage is reduced by more than 5% with the flexibility enabled. When learning MNIST with different learning models (multi-layer perceptron (MLP), convolutional neural network (CNN)), our method with flexibility and customized weight design outperforms the benchmarks in 29 out of 48 cases. When learning CIFAR10 with CNN, flexible strategy always helps to improve the performance of various methods by around 3% in dynamic communication environment.

Our work contributes to the deployment of FL in dynamic scenarios as follows:

1) We design and evaluate a flexible training and uploading strategy for FL devices in time-varying communication environments with non-stationary computation resources.
2) We propose a novel method for computing aggregation weight to address the challenges brought by heterogeneous model updates caused by imbalanced data distribution and dynamic computation and communication resources and further aggravated by our flexible communication strategy;
3) We show through experiments that the flexible learning scheme is effective in facilitating asynchronous FL in dynamic environments by shortening the required communication time and improving the utilization of computation resources.

The rest of the paper is organized as follows.

In Section 2, we elaborate on previous work related to heterogeneity in FL as well as the proof of learning convergence in FL. In Section 3, the components and algorithms of our FL system are presented. We provide a theoretic analysis of the loss convergence with our method in Section 4. In Section 5, results of experiments are discussed. We conclude the paper in Section 6.

2 Related Work

2.1 Modelling Heterogeneity

Heterogeneity of data distribution and potential communication constraints have been highlighted in one of the first papers proposing FL: FedAvg [17], where the authors conducted experiments of different tasks with independent and identically distributed (IID) data and non-IID data. Although the communication between the server and clients is assumed to be steady, the client-selection step of [17] can be interpreted as working with the varying availability of clients, which is caused by limitations of either computation or communication resources. Qu et al. [18] and Jin et al. [19] and Li et al. [20] also adopted such process of client selection. Specifically in [20], a portion of clients are simulated to be underperforming, but allowed to upload incomplete training results to the server within the round time, so as to mitigate the bias of global model towards fast workers. The previously mentioned work simulated the effect of heterogeneity, e.g. the limited participation of clients and partially trained models. The underlying factor causing the heterogeneity of clients' contribution is not presented in the experiment process. Therefore, whether the heterogeneity effect is close to reality is to be investigated. In some other work, the limitations of clients' resources are realistically modelled. Wu et al. [21] defines a variable, following an exponential distribution, to set the number of batch optimizations processed by a FL client. In FedCS [10], the uplink throughput for clients to upload models is drawn from a simulated LTE environment with time-division resource blocks. Nishio and Yonetani [10] and Wu et al. [21] focus on the computation and communication conditions of FL clients respectively. In our previous work [16], we developed a simulation scheme that models both constraints with possibility of further customization. In this paper, we continue to adopt this simulation scheme and thus model both aspects of the heterogeneity explicitly.

2.2 Approaches to Address Heterogeneity

2.2.1 Data Pruning and Partial Transmission

In [22], the authors experimented with pruning method to accelerate the training of deep neural network. Such experiments confirm the redundancy of information in the neural network, which is possible to be pruned/compressed in FL as well. By pruning models in either training or uploading stage, the corresponding resource consumption can be reduced and thus facilitating the process. Pfeiffer et al. [23] conducted experiments on partial training and transmission of quantized model which is adapted along the resources available to FL clients. Similar approach of compressing models by quantization can be found in [14], [24], where quantization of model parameters into fewer bits is conducted to save communication resources. Moreover, it is also possible for FL clients to transmit an incomplete

part of the model. A common method is to apply matrix sparsification as in [25], [26] so that only the weights with greater numerical importance are transmitted. Data pruning and partial transmission methods can greatly reduce the communication overhead in FL. Nevertheless, for most of the case, the scheme of pruning/compression need to be decided and fixed for the whole training/communication stage of a round, therefore lacking the agility to respond to fast-changing conditions of resources. Besides, the computational overhead and time delay introduced in the data pruning process also need to be weighted with the benefits depending on the specific scenarios.

2.2.2 Flexible Strategies for Client Participation

A multitude of research has been conducted on the flexible participation of clients in FL, e.g. allowing partially trained model updates or stale updates out of the designated synchronization round, and target-driven selection of participants.

Variable Local Progress: To allow incomplete progress of local training as a result of computational limitation, Li et al. [20] designed a parameter of inexactness to set threshold on the number of local optimization, which is then to be customized based on the individual availability of resources of each client in FL. However, the model updates after different number of optimizations are treated homogeneously with identical weights in model aggregation. In [13], an offloading scheme, in which FL clients' computing tasks can be partially offloaded to the associated small base station, is proposed to speed up the training process which otherwise is delayed by slow FL clients. The model heterogeneity resulted from varying training progress is not discussed. In our previous work [16], we also allow model updates of different training progress to be aggregated. The aggregation weights are computed based on clients' data size, staleness of the model, and the number of optimizations performed to train the model. While such method works well in our settings, theoretical analysis on the convergence of learning is to be developed. In this paper, we design a method of computing aggregation weight based on the convergence analysis and thus provide theoretical support for the success of the method.

Easing Synchronization Constraints We often use the term staleness, usually related to the interval between the two consecutive updates of a client, to distinguish fast and slow clients in the FL task. For a FL scenario with clients of heterogeneous computation and communication capacity, synchronized federated learning could suffer from low computation efficiency if it aims to collect stale model updates from slow clients by setting a longer round time. On the other hand, when setting the round time too short, the global model would be biased towards fast workers. The computational resources invested by slow devices are likely to be discarded if not uploaded to server timely. To address the dilemma in setting a suitable round time for synchronization, multiple researchers have worked on relaxing the stringent requirements of round time for synchronization. While still retaining the global clock for round timing, Wu et al. [21], Wang et al. [27], and Ma et al. [28] all adopt a scheme where the stale updates from slow clients are accepted for model aggregation in later rounds, which is often named semi-asynchronous FL or buffered asynchronous FL. They can be further classified by the criteria for the global clock to step forward, including the time-based synchronization [21] and buffer-based synchronization [5], [28], [29]. While it is beneficial to aggregate multiple model updates in the buffer to improve the stability of FL, clients, whose model updates are in the buffer, stay idle and wait for the distribution of global model. The computational resources of these clients are therefore not fully utilized. To maximize the utilization of computational resources, the concept of synchronization round, together with the global clock to time it, can be cancelled. In this case, the corresponding FL scheme becomes fully asynchronous [30]. While Xie et al. [30] provided an effective weight computing method (polynomially-decreasing weight w.r.t. model staleness as), the superiority of it is not explained in the convergence analysis. We elaborate more on this issue in Section 2.2.3.

Target-Driven Selection of Participants: In both synchronous and asynchronous FL, it is possible to select a subset of fast or reliable clients to achieve fast convergence or fairness. For example, Nishio and Yonetani [10] aim to maximize the number of participants in one round of aggregation by assigning early time-division resource blocks to fast participants. Imteaj and Amini [31] propose a scoring scheme to evaluate the participants' activity and contribution and select the most reliable ones. Wang et al. [14] use a reinforcement-learning-based scheme to perform client selection as well as bitwidth for compressing models in each round of synchronous FL. In selection algorithm taking clients' status of resource as inputs, clients are often required to exchange their system information, thus creating additional communication overhead. Moreover, such exchange might not even be successful in extreme communication conditions, e.g. when the FL clients are only able to connect to the server at sporadically distributed time windows.

2.2.3 Aggregation Weight Design for Mitigating the Adversarial Effect of Stale Model Updates

Stale model updates in FL has been a prominent effect when synchronization requirements are eased or client selection is performed. Due to the lack of new information of global model, stale model updates are deemed to be risky to be aggregated into global model since the optimization direction might differ from the direction to optimize the latest global model. Numerous work has identified this issue and proposed corresponding solutions. From the beginning of asynchronous FL, Xie et al. [30] experimented with polynomial and Hinge functions to decrease the weight of stale model updates. FedBuf in [5] is a semi-synchronous FL scheme which use a buffer to save the model updates from a certain number of clients. In the basic scheme, the model updates inside the buffer are assigned equal weights. For improvements, Nguyen et al. [5] also adopted the scaling function as in [30]. Polynomially-decreasing function generally works well but concrete analysis on its good performance is yet to be developed. KAFL in [29] is also a FL scheme with buffered model updates for aggregation. The authors modify the classical datasize-based weight design with adaptation dependent on the clients' updating frequency and the similarity between clients' model updates and the global model. Zhou et al. [32] present a weight

adaptation method for buffered synchronized FL where the gradient updates from clients are adjusted based on their similarity to the average gradient. The learning rate is adapted with a reciprocal function of the staleness. Wang et al. [27] designed a weighting scheme when aggregating updates with different staleness, with a goal to match each clients' overall contribution proportionally to the size of their local data.

We note that in these methods, adapting the aggregation weights for stale updates usually entails an attenuation function, which needs to be carefully tuned in trial runs so as to achieve good performance. There exists methods other than adjusting aggregation weight based on a deterministic function. Bäckström et al. [33] introduced a way of adapting the step size in asynchronous stochastic gradient descent (SGD) based on the statistical distribution of staleness observed in the learning process.

2.3 Proof of Convergence in the Analysis of FL

Convexity or smoothness is often assumed to facilitate the derivation of convergence bounds. Nevertheless, there is a multitude of work [34], [35] pointing out that assuming convexity is unfit for the architecture of most of the modern machine learning models, therefore leading to the deterioration of performance of applying such models in FL. Yu et al. [36] analyzed the neural network structure in FL and identified that nonconvexity in the final layers of neural networks caused the performance degrade in federated optimization. They further proposed solution to convexify the models and improved the FL performance. Similar techniques can be adopted to other FL schemes and thus allowing the assumption of convexity in analyzing convergence in FL. Normally, we convert the difference of loss values between two models into the scale of the difference of the two models by assuming the gradient of the loss function is Lipschitz-continuous as in [27]. However, after the conversion, bounding the scale of the difference of two models is not well-investigated yet in asynchronous FL. In [30], [37], the difference of two neighboring global models are directly bounded by the scale of changes obtained from local optimization, ignoring the errors introduced in aggregating stale updates. On the other hand, without converting difference of loss values using Lipschitz-smoothness, there exist other simplified approaches (e.g., [38], [39]) to directly assume an upper limit of the difference of loss between of two models, Since the general target of proof of convergence is to derive a bound of loss difference between the initialized model and the (imaginary) optimal model resulting in the least loss, directly bounding the loss difference between two random models is practically too strong of an assumption that fails to consider the scale of proximity of two models.

In general, despite the challenges posed by application conditions, allowing flexible participation suits the characteristics of FL in dynamic environments. Our work follows this direction and aims to build a system that maximizes flexibility and also address the heterogeneity issue resulting from it.

TABLE 1: Variables, parameters, and acronyms used in algorithms, theoretical analysis, and experiments

Notation	Definition
k	version of global model, also used as subscript for terminology related to the k-th model update/aggregation
$\mathbf{x}_k$	global model after k aggregations
α_k	aggregation weight for the k^{th} aggregation
τ_k	index of global model on which that the model used for the k-th aggregation is trained
h_k	number of optimizations performed to yield the k-th model update
i	index of a FL client, also used as superscript for terminology related to client i
$\mathbf{x}^i$	the latest model uploaded by client i
τ^i	index of global model on which the client i's newly uploaded model is trained
h^i	number of optimizations performed by client i to yield its latest model update
$\mathbf{x}_{k,h}$	the model trained based on $\mathbf{x}_k$ after h optimizations
γ, B	learning rate and batch size for all FL clients
μ	convexity factor of the model, to be tested for each task
E_{min}, E_d, E_{max}	the low, middle and high thresholds for optimization epochs
g	Used as superscript for any terminology related to global model at the server
α_{max}	maximum value that α_k can take

3 System Design

In this section, we elaborate on the learning algorithms, the functions of system components with early stopping and extended training actions, and the model aggregation scheme with specific weight design method. Variables, parameters and acronyms are shown in Table 1.

3.1 Learning Objective

We consider a FL task performed by N devices together with a model aggregator (server). Data used for training and testing is distributed as the vector $\mathbf{D} = \langle D^1, \ldots, D^N \rangle$. The local dataset D^i is only visible to device i. The overall target of the FL task is to find a model that minimizes the evaluation loss on the combined dataset $D = \cup_{i=1}^{N} D^i$. Mathematically, with the loss function f, we define the centralized global task as finding the optimal model $\mathbf{x}^*$ which results in the minimum global loss:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d} [F(\mathbf{x}) \coloneqq \mathbb{E}_{d\sim D} f(\mathbf{x}, d)]$$

With local loss function defined as $f^i(\mathbf{x}) \coloneqq \mathbb{E}_{d\sim D^i} f(\mathbf{x}, d)$, devices perform local training task aiming to minimize the local loss $\mathbf{x}_i^* = \arg\min_{\mathbf{x}} f^i(\mathbf{x})$ via mini-batch gradient-descent method: $\mathbf{x}' \leftarrow \mathbf{x} - \gamma \cdot \frac{1}{B} \sum_{d\in D_B^i} \nabla_{\mathbf{x}} f(\mathbf{x}, d)$, where γ is the local learning rate and D_B^i is the batch of sampled data in each optimization of client i.

Algorithm 1 Learning System Simulation

Input: $N, T, \gamma, B, E_{min}, E_d, E_{max}, \mathbf{D}, m$ // Input parameters: number of clients, total steps, local learning rate, batch size, three thresholds of optimization number, dataset distribution, size of model, function fitting threshold
Output: $\mathbf{x}_k$

Initialization
Initialize and distribute $\mathbf{x}_0$ to all clients
Set parameters: $\gamma, B, E_{min}, E_d, E_{max}$ on each client
Assign local dataset D^i to client i, $\forall i \in \{1, 2, \ldots, N\}$
$\texttt{TA} \leftarrow \emptyset$

Optimization
$k \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  for $i \in [N]$ **do**
    $\texttt{client}[i].\texttt{step}()$
    if $\texttt{client}[i].\texttt{state} == \texttt{wait}$ **then**
      $\texttt{TA} \leftarrow \texttt{TA} \cup \{(i, \tau^i, h^i, \mathbf{x}^i, t^i_{cut})\}$
    end if
  end for

  while $\texttt{TA} \neq \emptyset$ **do**
    pick from TA the model $\mathbf{x}^i$ with smallest t_{cut}
    $\alpha_k \leftarrow \texttt{ComputeWeight}(k, h^i, \tau^i, i)$
    $\mathbf{x}^g_{k+1} \leftarrow (1 - \alpha_k) \cdot \mathbf{x}^g_k + \alpha_k \cdot \mathbf{x}_{\tau^i, h^i}$
    $k \leftarrow k + 1$
    distribute $\mathbf{x}^g_k$ to client i
    $\texttt{client}[i].\texttt{state} \leftarrow \texttt{distributed}$
  end while

end for
return $\mathbf{x}_k$

3.2 Learning Systems

We orchestrate a learning task in T time steps as described in Algorithm 1. The system starts by defining learning parameters and initializing models on clients. At each step, the clients and the server take actions sequentially. The computation token s^i_t and the communication token c^i_t are used to control the local learning and uploading speed, respectively. Currently, both tokens are drawn from an offline sampling and set at the beginning of learning; in future work, we will consider extending the work to connect with real-time measurement from a mobile computing system. We simplified the modelling of communication and only consider the transmission from client to server. The reasoning behind focusing on upload as opposed to download is that, in a typical communication system, the available resources for upload (uplink) are lower than those for download (downlink), thus making the uplink the bottleneck. If cases where downlink is expected to have limitations comparable do uplink, the transmission in the reverse direction can be modelled in the same way.

Algorithm 2 `ComputeWeight`: Aggregation Weight Computation

Input: k, h, τ_k, i // current version of global model, number of optimizations, older version of global model on which the received model is trained, client index
Output: α_k

Initialization
α_{max}, a, b, c distributed from server

Computation
if all clients have uploaded once **then**
  $q_k \leftarrow \exp(a \cdot e^{bk} + c)$
  $l_{opt} \leftarrow (1 - \gamma\mu)^h \cdot \|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2$
  $\alpha_k \leftarrow q_k / l_{opt}$
  return $\min\{\alpha_k, \alpha_{max}\}$
else
  return α_{max}
end if

3.3 System Components

3.3.1 Server

The server maintains a global model and receives model updates from clients. It tracks and indexes the uploaded models from clients, and keeps a list of the global models based on which the clients are performing training.

The server keeps an index k to denote the version of the global model. At each time step, if a model update is received from client ($\mathbf{x}_k$), the server logs the model, the corresponding number of batch optimizations performed by the client (h_k), as well as the index of global model on which the uploaded model is trained (τ_k). The model aggregation is performed as weighted average of older version of global model and the received model from a client. Our proposed approach to designing aggregation weights is described in Algorithm 2. The motivation of such design is that $\alpha_k \cdot (1 - \gamma\mu)^{h_k} \cdot \|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2$ is a major indicator of error introduced in the federated learning process as shown in Section 4.2 and Section 4.3. We perform trial experiments in Section 4.2 and confirm that $\alpha_k \cdot (1 - \gamma\mu)^{h_k} \cdot \|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2$ decreases in a asymptotic way to $\exp(a \cdot e^{bk} + c)$ with $a > 0$ and $b < 0$. Note that such weight design is only applied after all clients have uploaded model at least once. For clients uploading for the first time, $\tau_k = 0$, $\|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2$ does not necessarily follow the trend of decreasing as described above.

3.3.2 Client

Clients are modelled as deterministic finite-state machines (cf. Algorithm 3). In each time step, a client performs actions such as training and uploading, or it stays idle. If one time step starts at the states of `distributed` or `training`, the client i performs s^i_t number of batch optimizations assigned by the external controller. Once triggered by function `DetermineUpload` which observes communication conditions and local optimization progress, the client transitions to the `uploading` state, where it transmit data of size c^i_{t+1} in the next time step. Upon finishing transmission, the client

Algorithm 3 `Step`: Client i Actions

Initialization

`ToUpload` $\leftarrow m$, `state` $\leftarrow$ `initialized`
`OptimizedEpochs` $\leftarrow 0$

Step function when `state` $\in \{$`distributed`, `training`$\}$

Input: t, s_t^i

$\mathbf{x}_k$ distributed from server, $\mathbf{x}_{k,0} \leftarrow \mathbf{x}_k$
$\tau^i \leftarrow k$
for $h = \{1, \ldots, s_t^i\}$ **do**
  Take B samples from local dataset and train the model $\mathbf{x}_{k,h} \leftarrow \mathbf{x}_{k,h-1} - \frac{\gamma}{|B|} \sum_{d \in D_B^i} \nabla_{\mathbf{x}} f(\mathbf{x}_{k,h-1}, d)$
  `OptimizedEpochs`$+ = 1$ if dataset iterated
  if `DetermineUpload`$(c_{t,\ldots,t+T_{pred}}) ==$ `True` **then**
    `state` $\leftarrow$ `uploading`
    `ToUpload` $\leftarrow m$
    break
  end if
end for

Step function when `state` $==$ `uploading`

Input: t, c_t^i

if `ToUpload` $- c_t^i \leq 0$ **then**
  $t_{cut} \leftarrow$ `ToUpload`$/c_t^i$
  `state` $\leftarrow$ `wait`
  return $i, \tau^i, h, \mathbf{x}_{k,h}, t_{cut}$
else
  `ToUpload` $\leftarrow$ `ToUpload` $- c_t^i$
end if

enters `wait` mode. The distribution of global model from the server triggers `distributed` mode.

3.4 Triggering Algorithm for Uploading

We propose an algorithm that aims to utilize good link conditions and triggers the devices to stop training and start uploading, as described in Algorithm 4. The action of the clients is dependent on the local training progress E as well as communication conditions. Compared with the fixed progress in classic solutions, denoted as the designated number of epochs E_d, a range of acceptable progress bounded by E_{min} and E_{max} is used. The clients are triggered to upload model only when link condition is good and the local progress is in the acceptable range. Function `TxT` in Algorithm 4 is defined as:

$$\texttt{TxT}(t, m, t_{end}) = \left(\underset{q \in \{t, \ldots t_{end}\}}{\arg\min} \sum_{p=t}^{q} c_p \geq m \right) - t + 1 \quad (1)$$

Given the predicted/detected communication tokens until time step t_{end}, Function `TxT` aims to find uploading duration if the client starts uploading at time t. When E is in the range of $[E_{min}, E_d)$, the client is triggered to upload only when it takes the shortest uploading time if starting at the current time step t. Otherwise, the upload is delayed so that the uploading time is further reduced. The attempt to delay transmission is deactivated when local training progress exceeds E_d. At this stage, clients will transition to upload if the expected upload duration is shorter than a threshold

Algorithm 4 `DetermineUpload`: Determine Uploading Action Based on Communication Conditions

E_{min}, E_d, E_{max} received from server during initialization

Input: $c_t, \ldots, c_{t+T_{pred}}$ // forseeable/predicted communication token of the next T_{pred} steps starting from current time t, current optimization epochs

$E \leftarrow B \cdot h / D^i$
if $E < E_{min}$ **then**
  return False
else if $E_{min} \leq E < E_d$ **then**
  $t_{min} \leftarrow \min_{p \in \{t, \ldots t+T_{pred}\}} \texttt{TxT}(p, m, t + T_{pred})$
  return $\texttt{TxT}(t, m, t + T_{pred}) == t_{min}$ and $\texttt{TxT}(t, m, t + T_{pred}) \leq T_{desired}$
else if $E_d \leq E < E_{max}$ **then**
  return $\texttt{TxT}(t, m, t + T_{pred}) \leq T_{desired}$
else
  return False
end if

$T_{desired}$. If a client does not capture a good uploading time and the optimization progress reaches the highest threshold E_{max}, the client stops the training process to prevent model divergence, and transitions to upload even if the link condition is not ideal. Currently, we assume all devices have knowledge of their local link conditions over T steps starting from the current time stamp t: $c_{t+1}, \ldots, c_{t+T_{pred}}$. Such overview of the link condition can be achieved by the devices' local prediction based on the observed patterns [40] or by receiving information from an entity with predicted quality of service enabled [41] for monitoring link conditions.

4 Aggregation Weight Design and Convergence Analysis

With the flexible uploading and training process enabled by Algorithm 4, the model updates provided by FL clients exhibit greater heterogeneity, as we show later in Section 5 Table 5. The staleness [1] of model updates varies, depending on whether upload action is triggered or delayed in Algorithm 4. Moreover, the number of optimizations performed by FL clients are no longer strictly proportional to the size of their local data. As a result, an aggregation scheme that is carefully tuned by staleness and optimization numbers is needed to address the challenges brought by more heterogeneous model updates. In this section, we start by stating the assumptions of the loss function of our learning tasks in Section 4.1. A brief analysis of the loss evolution is provided in Section 4.2. Furthermore, we introduce the motivation of our method of aggregation weight design based on the analysis. Lastly, in Section 4.3, we provide an asymptotic bound for the loss function based on our method.

4.1 Assumptions of Function Properties

We made the following assumptions commonly used in the analysis of distributed optimization [5], [20], [30] to

1. We define staleness as the time interval between the last model update of a client and its latest model update

facilitate our analysis of the loss evolution of the global model through the learning process.

Assumption 4.1. (L-smoothness) Let function F satisfy L-Lipschitz smooth, so $\exists L > 0$, $\forall \mathbf{x}_1, \mathbf{x}_2 \in$ `model space`

$$F(\mathbf{x}_1) - F(\mathbf{x}_2) - \langle \nabla F(\mathbf{x}_2), \mathbf{x}_1 - \mathbf{x}_2 \rangle \leq \frac{L}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2$$

Assumption 4.2. (Convexity) Let function F be convex. As a result, $\forall \alpha \in [0, 1]$ and $\forall \mathbf{x}_1, \mathbf{x}_2 \in$ `model space`

$$F(\alpha \mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) \leq \alpha F(\mathbf{x}_1) + (1-\alpha)F(\mathbf{x}_2)$$

Assumptions 4.1 and 4.2 are reasonable because the properties of smoothness and convexity of a function are preserved with operations like addition and affine projection (common operations in the machine learning models adopted in our experiments), and can be tested with techniques in Section 5.2.1.

Assumption 4.3. When sampling data points in batch from local dataset D_i, the difference between the averaged gradient and the true gradient of the local loss function is bounded.

$$\Big\| \frac{1}{B} \sum_{d \in D_B^i} \nabla_{\mathbf{x}} f(\mathbf{x}, d) - \nabla_{\mathbf{x}} f^i(\mathbf{x}) \Big\|_2^2 \leq \sigma^2,$$
$$\forall \mathbf{x} \in \texttt{model space}, d \sim D^i \; \forall d \in D_B^i$$

Assumption 4.3 holds in case of reasonably large batch size B of samples taken in one local gradient optimization.

Assumption 4.4. The difference of local target and global target, caused by imbalanced data distribution (feature imbalance), is bounded. Therefore, the difference of their gradients w.r.t. the same model is also bounded.

$$\|\nabla F(\mathbf{x}) - \nabla f^i(\mathbf{x})\| \leq \phi, \; \forall \mathbf{x} \in \texttt{model space}$$

ϕ is directly related to the difference of local dataset w.r.t. the global dataset. Since the local dataset is a subset of the global dataset and, therefore, ϕ is limited, thus, Assumption 4.4 is valid.

Assumption 4.5. The gradient of global loss function w.r.t. any model is bounded.

$$\|\nabla F(\mathbf{x})\|_2^2 \leq G^2, \; \forall \mathbf{x} \in \texttt{model space}$$

Assumption 4.5 provides a rough estimation of the scale of gradients, which is a relatively loose bound.

Furthermore, we list Polyak-Łojasiewicz (PL) inequality here, which has a weaker quality than strong convexity. We show later that such property is verifiable through testing the convexity of a function. If F is μ-strongly convex, then F also satisfy μ-PL condition.

Assumption 4.6. Let function F satisfy the PL inequality. As a result, $\exists \mu > 0$,

$$\frac{1}{2}\|\nabla F(x)\|_2^2 \geq \mu \left(F(x) - F\left(x_*\right)\right)$$

With Assumptions 4.1, 4.6, and 4.4 and setting $\gamma L \leq 1$, we derive the following corollary about the evolution of loss evaluated during the local optimization process.

Corollary 4.1. The expectation of the difference of loss between the h-time locally-optimized model and the optimal global model can be expressed as

$$\begin{aligned}&\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k}\right) - F\left(\mathbf{x}_*\right)\right] \\ \leq &(1-\gamma\mu)^{h_k} \cdot \mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,0}\right) - F\left(\mathbf{x}_*\right)\right] + \frac{\phi^2 + \sigma^2}{\mu}\end{aligned}$$

4.2 Motivation of Weight Design

We start by analyzing the loss evolution of the global model after an aggregation step at the server.

$$\begin{aligned}&\mathbb{E}[F(\mathbf{x}_{k+1}) - F(\mathbf{x}_*)] \\ =&\mathbb{E}[F((1-\alpha_k)\cdot\mathbf{x}_k + \alpha_k \cdot \mathbf{x}_{\tau_k,h_k}) - F(\mathbf{x}_*)] \\ \leq&(1-\alpha_k)\cdot\mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \mathbb{E}[F(\mathbf{x}_{\tau_k,h_k}) - F(\mathbf{x}_*)] \\ \leq&(1-\alpha_k)\cdot\mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] \\ &+\alpha_k \cdot \left[(1-\gamma\mu)^{h_k}\cdot\mathbb{E}[F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_*)] + \frac{\phi^2+\sigma^2}{\mu}\right] \\ \leq&\underbrace{\left[(1-\alpha_k) + \alpha_k\cdot(1-\gamma\mu)^{h_k}\right]\cdot\mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)]}_{S_1} \\ &\underbrace{+\alpha_k \cdot \left[(1-\gamma\mu)^{h_k}\cdot\mathbb{E}[F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k)] + \frac{\phi^2+\sigma^2}{\mu}\right]}_{S_2}\end{aligned} \tag{2}$$

The difference of loss $F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k)$ is hard to obtain in FL settings since the loss needs to be evaluated on all clients. To this end, Assumption 4.1 (L-smooth) is used to convert it into terms of gradient scales $\|\nabla F(\mathbf{x}_k)\|^2$ and the difference of models $\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2$:

$$F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k) \leq \frac{1}{2}\|\nabla F(\mathbf{x}_k)\|^2 + \frac{1+L}{2}\cdot\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2$$

We then obtain:

$$\begin{aligned}S_2 \leq&\alpha_k\cdot(1-\gamma\mu)^{h_k}\cdot\frac{G^2}{2} + \frac{\phi^2+\sigma^2}{\mu} \\ &+\underbrace{\alpha_k\cdot(1-\gamma\mu)^{h_k}\cdot\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2}_{e_1}\cdot\frac{1+L}{2}\end{aligned}$$

We set the error term apart from gradient scale G and gradient difference ϕ as $e_1 = \alpha_k\cdot(1-\gamma\mu)^{h_k}\cdot\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2$. It corresponds to the common concept in FL weight design ([17], [42]) where models from clients with larger share of data (that also perform more local optimizations) get assigned larger weights, while stale updates get assigned smaller weights. To support this in theory, we state that a bigger aggregation weight α_k is always desired, as shown in S_1 in Inequality 2, where bigger α_k leads to a greater decrease of loss. Since $h_k \propto D^i$ and E, we can assign bigger weights α_k to updates with higher h_k under the same error limitation $e_1 \leq e_{limit}$. Similarly, stale updates tend to differ from the latest version of global model and result in greater distance $\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|$. A smaller weight α_k is then needed to satisfy the error constraints.

In a learning task with a converging trend, the loss of global model will drop and gradually converge to a

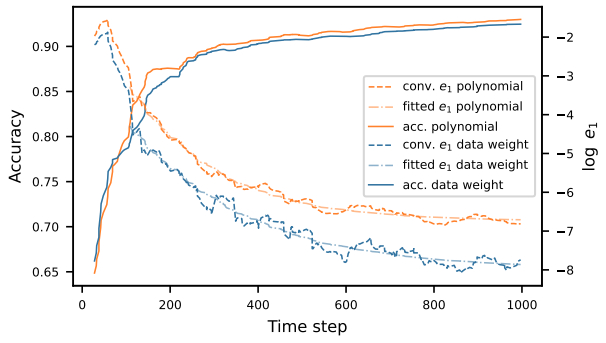


Fig. 1: Value evolution of smoothed `log` e_1 and testing accuracies in trial experiment with MLP learning MNIST

steady value, ideally close to $F(\mathbf{x}_*)$. Moreover, as the learning goes on, the difference of global models $\|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2$ also decreases. As a component of the upper bound for $F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k)$, we suspect e_1 also follows the decrease-flat trend. In a trial run of experiments (in Figure 1) with aggregation weight $\alpha_k = \frac{|D_i|}{\sum_{i\in[N]}|D_i|}$ and $\alpha_k = \beta \cdot (k - \tau_k + 1)^{-\lambda}$, both adopted in our experiments as benchmarks, we obtain values of e_1 with respect to k, which can be approximated by function $\exp(a \cdot e^{bk} + c)$

To enable a smooth learning progress, we set an upper bound for α_k as $\frac{\exp(a \cdot e^{bk} + c)}{(1-\gamma\mu)^{h_k} \cdot \|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2}$. Another upper limit α_{max} is later added to prevent unreasonably large aggregation weight (e.g. greater than 1) resulting from small $\|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2$. Adding α_{max} is close to set a fixed aggregation weight or the simplified case of $\beta \cdot (k - \tau_k + 1)^{-\lambda}$ when $\lambda = 0$ and $\beta = \alpha_{max}$ in [5], [30]. Our method avoids multiple trial runs to find suitable settings of β and λ. Furthermore, We show that this method leads to analytically asymptotic convergence in Section 4.3. In experiments in Section 5.3, it is also verified to work well especially with training and transmission flexibility (in Algorithm 3) enabled.

4.3 Proof of Convergence

For each set of parameter setting of a learning task, we conduct one trial experiment (with aggregation weight computed as $\alpha_k = \frac{|D_i|}{\sum_{i\in[N]}|D_i|}$) regardless of environment scenario and obtain a fitted curve of e_1: $\exp(a \cdot e^{bk} + c)$; the weight for aggregation α_k upon each received model in subsequent formal experiment is $\min\left\{\frac{\exp(a \cdot e^{bk} + c)}{(1-\gamma\mu)^{h_k} \cdot \|\mathbf{x}_k - \mathbf{x}_{\tau_k}\|^2}, \alpha_{max}\right\}$. Take $Q_1 = \max_{j\in\{0,\ ...,\ k\}} \left\{(1 - \alpha_j) + \alpha_j \cdot (1 - \gamma\mu)^{h_j}\right\}$ and use Assumption 4.5, we further derive Inequality 2 as follows:

$$\mathbb{E}[F(\mathbf{x}_{k+1}) - F(\mathbf{x}_*)] \leq Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \sum_{j=0}^{k} Q_1^{k-j} \left\{\alpha_j \left[\frac{(1-\gamma\mu)^{h_j} G^2}{2} + \frac{\phi^2 + \sigma^2}{\mu}\right] + \frac{(1+L)e^{ae^{bj}+c}}{2}\right\} \tag{3}$$

Set $I(k) = \exp(ae^{bk} + c)$. Since $b < 0$, $I(k)$ decreases when k increases. We can construct a geometric progression $J(k)$:

$$\begin{aligned}
&J(0) = e^{a+c} = I(0) \\
&J(k) = e^{c+\epsilon} \geq I(k), && \text{choose } \epsilon \geq ae^{bk} \\
&J(\kappa) = J(\kappa - 1) \cdot e^{(\epsilon - a)/k}, && \forall \kappa \in \{1, \ldots, k\} \\
&J(\kappa) \geq I(\kappa), && \forall \kappa \in \{0, \ldots, k\} \\
&Q_1^{k-j} \cdot I(j) \leq Q_1^{k-j} \cdot J(j) = J(0) \cdot && Q_1^{k-j} \cdot (e^{(\epsilon-a)/k})^j
\end{aligned}$$

I decrease faster than J. With the factor of geometric progression $r = e^{(\epsilon-a)/k}$, and set $Q_2 = \max\{r, Q_1\}$, $\alpha_{max} = \max_{j\in\{1,\ldots,k\}} \alpha_j$, we can derive Inequality 3 further:

$$\begin{aligned}
&\mathbb{E}[F(\mathbf{x}_{k+1}) - F(\mathbf{x}_*)] \leq Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] \\
&+ \left(\frac{G^2 Q_1}{2} + \frac{(\phi^2 + \sigma^2)\alpha_{max}}{\mu}\right) \cdot \frac{1 - Q_1^{k+1}}{1 - Q_1} \\
&+ \frac{Q_2^k (1+L) e^{a+c}}{2} \cdot (k+1) \\
&\leq O\left(D_0 Q_1^{k+1}\right) + O\left(\frac{Q_1}{1-Q_1}\right) + O\left(\frac{1}{1-Q_1}\right) + O\left((k+1)Q_2^k\right)
\end{aligned} \tag{4}$$

The complete derivation from Inequality 2 to 4 can be found in Appendix A.2.

Q_1 and Q_2 need to be smaller than 1 to ensure finte loss. In our task settings, both γ and μ are ≥ 0 (because of convexity). We also made sure that $\gamma\mu < 1$ during derivation. Therefore, $Q_1 = 1 + \alpha_k \cdot ((1-\gamma\mu)^h - 1) \leq 1$. To ensure $Q_2 \leq 1$, we need to set $r = e^{(\epsilon-a)/k} \leq 1$, which indicates $\epsilon \leq a$.

5 Experiments

We initiate experiments with flexibility dependent on link condition, therefore the varying condition in the experiments is the link profile/behavior.

With functions `SetComputationCapacity(t)` and `SetLinkStatus(t)` in Algorithm 1, we can now experiment with various application scenarios. Moreover, we also experiment with different data distribution on devices. The size of the local dataset, the computational capacity, and the link throughput have intertwined impacts the updating interval of clients. We experiment with three learning tasks: classification tasks with MLP and CNN on MNIST [43], classification task with CNN on CIFAR10 [44], and classification task with SqueezeNet [45] on CIFAR100 [44]. The details of the model structures are shown in Appendix B: Table 11 and Table 12. We perform a centralized training with these models on the corresponding dataset and achieve prediction accuracy of 95% for MLP on MNIST, 98% for CNN on MNIST, 62% for CNN on CIFAR10, 57% for SqueezeNet on CIFAR100.

5.1 Scenarios

5.1.1 Data Distribution

For classification datasets like MNIST and CIFAR with fixed number of data points, we aim to derive scenarios with different data distribution over the clients, in terms of the number of data samples and the number of available classes at one client. We construct two stages of optimization tasks to assign the samples in the dataset to each client.

TABLE 2: Link throughput profiles used in experiments.

Index	mean of tx. time	std. of tx. time	distribution
1	20.5	4.5	Poisson
2	40.5	6.5	Poisson
3	29.6	14.0	log-normal
4	7.3	0.5	from measurements in Figure 2

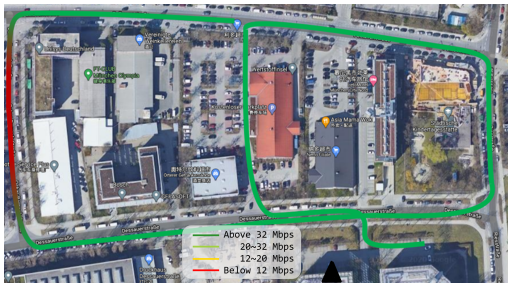


Fig. 2: Driving route and uplink throughput of Link Profile 5. The base station is located at ▲. The height of the base station antenna is 21 meters above ground level, whereas the vehicle antenna is at approximately 1.5 meter height, mounted on the vehicle roof. The test vehicle traversed the double loop shown by the overlay 10 times.

The first stage is to assign specific numbers of samples to each client so that the standard deviation is close to a designed value. To this end, for $n^i, i \in [N]$ and with target standard deviation σ_d and condition $\sum_{i=1}^{N} n^i = n^{total}$, the error to be minimized is $|\frac{1}{N}\sum_{i=1}^{N}(n^i - \mu_D)^2 - \sigma_d^2|$. Here $\mu_D = \frac{|D|}{N}$ since the complete dataset is distributed. The optimization for n_i does not aim for a fix solution, but for finding a data distribution in accordance with the desired heterogeneity on data samples.

The second stage is to assign samples of specific classes to each client. The classes available at a client C_i is randomly chosen in $[C]$. Therefore we have conditions: $n_c^i \geq \theta_c \cdot n^i/|C^i|$ if $c \in C^i$, otherwise $n_c^i = 0$. Then we construct conditions on the number of samples per client and per class: $\sum_{i=1}^{N} n_c^i \leq N_c, \forall c \in [C]$ and $\sum_{c=1}^{C} n_c^i \leq n^i, \forall i \in [N]$. The target to be minimized is then $n^{total} - \sum_{i=1}^{N}\sum_{c=1}^{C} n_c^i$. θ_c controls the class imbalance in a client's local dataset and is gradually decreased if no solution is found. Results from both stages n^i and n_c^i are rounded up to the closest integer.

5.1.2 Link Variability

The variability on link throughput of clients is controlled via c_t^i in Algorithm 3. We list four different scenarios where the link throughput having different characteristics as shown in Table 2, see also Figure 2. The distribution of transmission time under these scenarios are plotted in Figure 3. All the link profiles $c_0^i, \ldots, c_T^i$ are pre-generated individually for each client i.

In experiments assuming perfect predictions on link throughput, $c_t^i, \ldots, c_{t+T_{pred}}^i$ are fed to the client i for determining whether to stop training and upload model to the server. To show the robustness of our algorithm, we also add perturbations to the generated communication tokens at each time step t and use $(\tilde{c}_t^i, \ldots, \tilde{c}_{t+T_{pred}}^i)$ fed to the client as imperfect predictions (Line 6 of Algorithm 3): $\tilde{c}_{t+p}^i = c_{t+p}^i + e_p^i$, where $e_p^i \sim \mathcal{N}\left(0,\ \sigma^i \cdot p/T_{pred}\right)$.

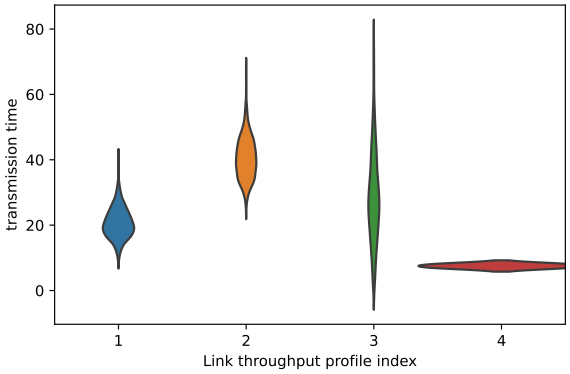


Fig. 3: Distribution plot of transmission time under four different link profile in Table 2. Profile 4 is sampled from measurements shown in Figure 2.

TABLE 3: Variability of computation resources for FL clients in three scenarios

Index	$p_{\min}$	$p_{\max}$	`scale`
a	10	10	1 for MNIST
b	5	15	2 for CIFAR10
c	3	17	
d	5	5	
e	3	7	1
f	1	9	

The motivation of adding perturbation to predictions based on the time difference is that predictions for later time steps are less reliable. When reaching the maximum predicting time step, the predicted communication throughput will exhibit an uncertainty ($e_{T_{pred}}^i$) of the same level as the link throughput of all time (σ^i).

5.1.3 Dynamic Computation Resources

The dynamics of computation resources for each client to perform learning tasks is simulated via $s_t^i \sim \text{uniform}\ \{p_{\min}, \ldots p_{\max}\} \cdot$ `scale` in Algorithm 3. Unlike the communication resource token c_t^i which can change every time step, we keep s_t^i steady for every 32 steps. This is to simulate the periodic change in the total work load for a FL client. We simulated with three different scenarios with different range of variety as shown in Table 3.

5.2 Experiment Settings

5.2.1 Testing Strong Convexity

In our analysis, the loss evolution during local optimization in Corollary 4.1 involves the convexity factor μ which is related to Assumption 4.6. We conduct the following test to get a estimated value of μ so that the aggregation weight design is valid. We test μ in strong-convex setting, which is a stronger requirement than PL inequality. Given the three following strong convexity properties:

$$F(\lambda\mathbf{x}+(1-\lambda)\mathbf{y}) \leq \lambda F(\mathbf{x})+(1-\lambda)F(\mathbf{y}) - \frac{1}{2}\mu t(1-\lambda)\|\mathbf{x}-\mathbf{y}\|_2^2,\ \forall\lambda\in(0,1) \quad (5)$$

$$\mu\|\mathbf{x}-\mathbf{y}\|_2^2 \leq \langle \mathbf{x}-\mathbf{y}, \nabla F(\mathbf{x})-\nabla F(\mathbf{y})\rangle \quad (6)$$

$$F(\mathbf{y})-F(\mathbf{x}) \geq \langle \nabla F(\mathbf{x}), \mathbf{y}-\mathbf{x}\rangle + \frac{\mu}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \quad (7)$$

By sampling $\mathbf{x}$ and $\mathbf{y}$ in the model space, and taking a range of λ, we can obtain a set of upper bound values that μ should be smaller than or equal to. Note that for models like CNN wherein there are pooling layers, we can still test the convexity of other layers such as convolutional layer and fully-connected layer. We show the settings of μ in different tasks in Table 4 after observing the upper bounds computed through Inequalities 5 to 7.

5.2.2 Hyperparameters and Benchmark

The hyper-parameter and scenario settings of experiments are listed in Table 4. Due to the different time required for transmission in various communication scenarios, we customize $T_{desired}$ and assume that the prediction for T_{pred} steps are available in all cases, which aims to maximize the benefits of the flexible-uploading algorithm. α_{max} is limited to be smaller than 1 in experiments for aggregation stability. During model aggregation, the case of $\alpha_k > 1$ means that all of the parameters of a model are scaled up numerically and the output value of the model is scaled up exponentially based on the number of layers of the model. Therefore, α_{max} is set to be ≤ 1 to bound each α_k so that the aggregated model does not yield unreasonable outputs.

The benchmark to evaluate our solution with are denoted as 1. polynomial attenuation method (abbr. polyn.) with α_k in Algorithm 2 alternated as $\beta \cdot (k-\tau_k+1)^{-\lambda}$, which is adopted in [5], [30]; 2. data weight (abbr. dw) aggregation adopted in [42], with α_k in Algorithm 2 alternated as $\frac{|D^i|}{\sum_{j=1}^{N}|D^j|}$, where i is the index of client uploading at the k-th round. Both are tested with and without flexibility in Algorithm 4 enabled.

There are in total 12 combinations of different link throughput profile and computation resource variations, we choose 2 representative scenarios for each of the tasks (learning MNSIT with MLP, learning MNIST with CNN, learning CIFAR10 with CNN, learning CIFAR100 with SqueezeNet). For each scenario, we experiment with Algorithm 4 enabled (abbr. with f.) and disabled (abbr. w/o f.) to verify the effect of the flexible uploading scheme for clients. When Algorithm 4 is disabled, function `DetermineUpload` returns $E \geq E_d$. Moreover, we also experiment with uncertainties in link predictions (abbr. with unc.) as described in Section 5.1.2.

Each experiment (of one setting) is done multiple times with 6 different random seeds, except for CIFAR100 with 3 different random seeds. The results are then averaged. For each random seed, the data distribution mentioned in Section 5.1.1 is also randomly distributed with the corresponding seed.

5.3 Results

Heterogeneity of Model Updates: We show the heterogeneity of model updates in terms of the variation of the number of optimizations to yield a model. In Table 5, we observe that after enabling flexibility, the standard deviation of the number of optimizations increased in each scenario, which is caused by allowing flexible training progress in Algorithm 4. When prediction for flexible actions are uncertain, the standard deviation of the number of optimizations still increases.

Resource Consumption: We show the changes brought by enabling flexible actions in terms of invested computation resources and time spent on transmission/training. In Figure 4, we show the distribution of the training time and transmission time of clients' individual update in three different scenarios in which experiments are conducted. We saw an increase of training time and reduced transmission time when enabling flexibility.

In Table 6, we show the statistics related to time and resource usage of the communication and training stage in FL tasks of learning MNIST. The trend of decreasing trans-

TABLE 4: Parameter settings in learning and flexibility algorithm, and in benchmark methods.

Parameter Group	Notation		Value
Common	γ		0.004
	T		10000 for CIFAR100, 8000 for CIFAR10 slow, 1000 for others
	E_d		1 for MNIST, 4 for CIFAR10 fast, 0.5 for CIFAR10 slow, 1 for CIFAR100
	E_{min}		0.75 E_d
	E_{max}		1.50 E_d
	B		8 for MNIST, 16 for other datasets
	α_{max}		0.25
Algorithm 2	a, b, c, μ	MNIST MLP	5.42, -0.0185, -6.53, 0.002
		MNIST CNN	5.34, -0.0112, -7.22, 0.1
		CIFAR10 fast	1.74, -0.0165, -5.39, 2.0
		CIFAR10 slow	6.93, -0.0057, -7.65, 2.0
Algorithm 4	T_{pred} $T_{desired}$	Profile 1	20, 16
		Profile 2	40, 34
		Profile 3	30, 16
		Profile 4	8, 6
Other's method	β		1
	λ		0.8

TABLE 5: Standard Deviation of the Num. of Optimizations before and after enabling Training and Uploading Flexibility in various Scenarios of Classifying MNIST

Scenarios	std. of Num. Optim. per update		
	with f.	with unc.	w/o f.
Link Profile 1 Comp. Profile c	167.7	153.5	124.8
Link Profile 2 Comp. Profile b	185.0	168.7	126.4
Link Profile 3 Comp. Profile a	183.2	165.2	129.0

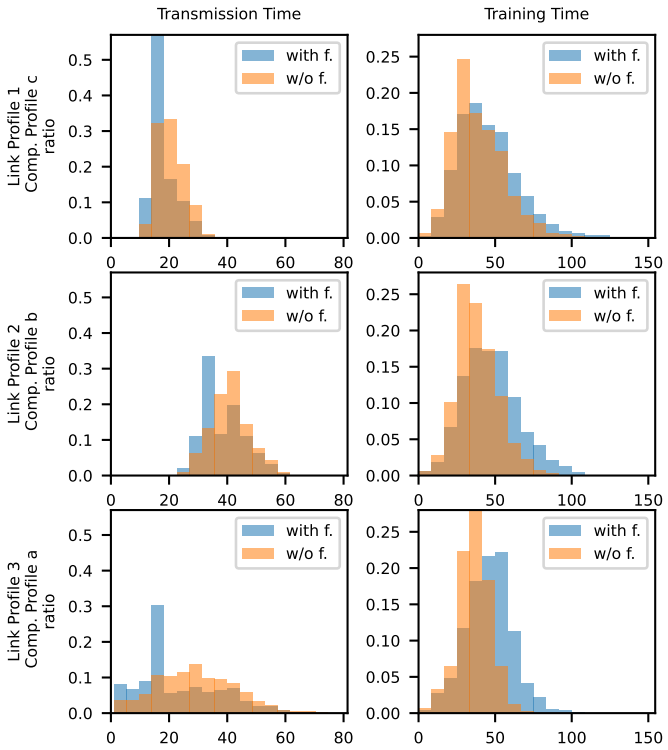


Fig. 4: Histogram of training time and transmission time of clients' updates under difference scenarios.

mission time and increasing training time after enabling flexibility is also confirmed by the average value. Besides, we observe a reduction in communication resource usage, which is computed by $\frac{m \cdot N_{updates}}{\sum_{t=1}^{T} \sum_{i \in [N]} c_t^i}$, where $N_{updates}$ is summed number of model updates of all clients. Specifically, we achieve a reduction of communication resources (used by the clients for uploading models to server) of 6.0%, 8.5%, and 4.6%, in three scenarios respectively. Such effect is achieved by the generally reduced number of uploads (the last k) of each client and also the summed number N_{update}. Equally as important, introduced flexibility results in the average transmission time reduction by 12.7%, 5.6%, and 21.0%, respectively, for the three scenarios. Finally, owing to the flexibility, the clients can spend additional time training the models, resulting in increased computational resource invested and thus 0.32% to 1.15% increase in averaged final testing accuracy for learning MNIST, 1.5% to 4.5% increase for learning CIFAR10, and 2.7% to 9.7% increase for learning CIFAR100. We also provide the statistics of used resources of CIFAR10 task in Appendix C Table 14. Details related to testing accuracy are elaborated on in the following sections.

Learning MNIST: We show that in Table 7 and 8, for learning MNIST in asynchronous FL, our solution (exponential-decay weight design combined with flexible training and transmission) outperforms other methods in more that half of the different data distributions (in terms of data std. and number of classes per client). Moreover, our solution also performs the best in terms of the averaged accuracy over all data distribution in each scenario. It is also noticeable that CNN always performs better than MLP in the same scenario (Link Profile 1 and Computation Profile c).

TABLE 6: Comparison of Statistics of Time and Resource Usage of FL clients in various Scenarios of Classifying MNIST

	Average Transmission Time			Communication Resource Usage		
	with f.	with unc.	w/o f.	with f.	with unc.	w/o f.
Link P. 1 Comp. P. c	17.8	18.1	20.4	0.312	0.323	0.332
Link P. 2 Comp. P. b	38.6	38.5	40.9	0.453	0.477	0.495
Link P. 3 Comp. P. a	22.9	22.8	29.0	0.309	0.327	0.324
	Average Training Time			Avg. Num. Optimization per client		
	with f.	with unc.	w/o f.	with f.	with unc.	w/o f.
Link P. 1 Comp. P. c	43.8	41.5	37.7	4388.8	4296.0	3919.0
Link P. 2 Comp. P. b	46.2	42.2	37.3	3398.9	3260.9	2948.1
Link P. 3 Comp. P. a	44.3	41.0	35.2	4231.0	4134.9	3525.0

Moreover, to investigate the effect of clients' flexible actions, we compare the performance of each method with and without flexibility enabled, under different data distribution and environments. When learning MNIST either with MLP or CNN, the method with flexibility outperforms that without flexibility in 62 out of 72 comparisons.

One sample accuracy evolution of MLP learning MNSIT with Link Profile 3, Computation Profile a, and data distribution std. 750 is plotted in Figure 5. Our method is generally less effective during the starting stage of the learning process, especially when class imbalance is prominent, having lower accuracies than others. Nevertheless, it is fast to catch up and outperforms ours in 3 cases in the end.

Learning CIFAR10: We show in Table 9 the final accuracy of learning CIFAR10 with CNN. When learning CIFAR10 with relatively fast pace, extreme class imbalance cause great fluctuation in the learning progress. Thus, we only show the learning results with relatively balanced data distribution with number of classes being 7 and 10 when learning CIFAR10 fast. We conducted experiments with slow speed settings (computation profiles d, e, and f) and longer time (8000 steps) to investigate the performance of learning CIFAR10 under more unbalanced data distributions with number of classes available at clients being 4 and 7.

In slow learning settings, our solution outperforms the other two methods. In fast learning settings performs worse than using polynomial weight attenuation and better than using data weight. Moreover, when comparing methods with and without flexibility, we see that methods with flexibility has a dominant advantage over those without flexibility in scenarios where link throughput varies prominently (Link Profile 1 and 3). In experiments with Link Profile 4, the advantage of enabling flexibility is less obvious. The weakened effect of our strategy originates from the stable link throughput of Link Profile 4, as shown in Figure 3. Therefore, the number of optimizations are not increased by

TABLE 7: Test accuracy after 1000 steps learning MNIST with 2-layer perceptron

Data Distr.	D std.	300				750				1000				avg.
Scenario	# classes p. client	4	6	8	10	4	6	8	10	4	6	8	10	
Link Profile 1 Comp. Profile c	dw with f.	0.850	0.885	0.864	0.870	0.845	0.887	0.882	0.870	0.850	0.885	0.898	0.870	0.871
	dw with unc.	0.837	0.886	0.864	0.887	0.851	0.885	0.863	0.886	**0.860**	0.869	0.897	0.886	0.873
	dw w/o f.	0.847	0.867	0.861	0.868	0.855	0.887	0.881	0.867	0.846	0.863	0.879	0.853	0.864
	polyn. with f.	0.859	0.890	**0.869**	0.875	0.841	**0.891**	**0.885**	0.875	0.848	**0.888**	0.883	0.874	0.873
	polyn. with unc.	0.841	**0.892**	0.868	0.893	0.855	0.890	0.866	0.891	0.856	0.872	0.900	0.890	0.876
	polyn. w/o f.	0.850	0.870	0.865	0.873	0.854	0.890	0.885	0.873	0.838	0.864	0.881	0.872	0.868
	ours with f.	**0.863**	0.892	0.869	**0.894**	**0.857**	0.889	0.868	**0.892**	0.851	0.870	**0.901**	**0.891**	**0.878**
	ours with unc.	0.848	0.892	0.868	**0.894**	0.856	0.886	0.866	**0.892**	0.855	0.873	0.901	0.891	0.877
	ours w/o f.	0.844	0.870	0.867	0.892	0.835	0.885	0.866	0.891	0.832	0.865	0.884	0.890	0.868
Link Profile 3 Comp. Profile a	dw with f.	0.874	0.914	0.918	0.925	0.873	0.913	0.917	0.926	0.865	0.912	0.918	0.927	0.907
	dw with unc.	0.873	0.912	0.918	0.925	0.871	0.912	0.917	0.925	0.871	0.911	0.918	0.926	0.907
	dw w/o f.	0.870	0.909	0.916	0.922	0.873	0.908	0.915	0.922	0.867	0.907	0.915	0.923	0.904
	polyn. with f.	0.883	**0.918**	0.924	**0.931**	0.880	**0.917**	**0.922**	0.931	0.866	**0.915**	**0.921**	0.931	0.911
	polyn. with unc.	0.886	0.916	0.924	0.930	0.880	0.915	0.921	0.930	**0.874**	0.914	0.920	0.930	**0.912**
	polyn. w/o f.	0.878	0.913	0.922	0.928	0.877	0.912	0.919	0.928	0.871	0.909	0.918	0.927	0.908
	ours with f.	0.884	0.915	0.923	0.931	**0.883**	0.914	**0.922**	**0.932**	0.869	0.914	**0.921**	0.930	**0.912**
	ours with unc.	**0.890**	0.912	**0.925**	0.931	0.874	0.914	0.922	0.931	0.874	0.914	**0.921**	**0.931**	0.911
	ours w/o f.	0.866	0.909	0.922	0.930	0.873	0.911	0.918	0.929	0.872	0.910	0.917	0.928	0.907

TABLE 8: Test accuracy after 1000 steps learning MNIST with CNN

Data Distr.	D std.	300				750				1000				avg.
Scenario	# classes p. client	4	6	8	10	4	6	8	10	4	6	8	10	
Link Profile 1 Comp. Profile c	dw with f.	0.885	0.930	0.933	0.939	0.889	0.932	0.932	0.939	0.889	0.930	0.930	0.940	0.922
	dw with unc.	0.885	0.932	0.932	0.938	0.892	0.929	0.933	0.940	0.894	0.930	0.914	0.940	0.921
	dw w/o f.	0.887	0.928	0.930	0.934	0.886	0.924	0.930	0.936	0.892	0.927	0.913	0.936	0.918
	polyn. with f.	0.893	0.942	0.940	**0.964**	0.898	0.941	**0.939**	0.946	0.900	0.941	0.921	**0.962**	0.932
	polyn. with unc.	0.898	0.941	0.949	0.963	0.906	0.940	0.939	0.945	**0.907**	0.941	**0.954**	0.961	0.937
	polyn. w/o f.	0.904	0.940	0.938	0.944	0.900	0.934	0.937	**0.961**	0.899	0.936	0.919	0.942	0.929
	ours with f.	0.896	**0.943**	**0.957**	0.963	0.902	**0.942**	0.939	0.945	0.902	0.941	0.937	0.961	**0.936**
	ours with unc.	0.901	0.942	0.956	0.945	**0.907**	0.940	0.938	0.946	0.906	**0.941**	0.953	0.959	**0.936**
	ours w/o f.	**0.908**	0.940	0.938	0.944	0.899	0.934	0.937	0.944	0.903	0.937	0.952	0.942	0.931
Link Profile 2 Comp. Profile b	dw with f.	0.859	0.922	0.928	0.930	0.851	0.918	0.929	0.935	0.848	0.915	0.929	0.935	0.908
	dw with unc.	0.852	0.922	0.927	0.932	0.854	0.918	0.928	0.933	0.855	0.912	0.927	0.933	0.908
	dw w/o f.	0.847	0.919	0.924	0.914	0.850	0.918	0.924	0.927	0.851	0.914	0.925	0.928	0.903
	polyn. with f.	0.881	0.935	0.937	**0.941**	0.876	0.935	0.935	**0.941**	**0.882**	0.934	**0.935**	**0.940**	0.923
	polyn. with unc.	0.865	0.934	0.935	0.941	0.871	0.932	**0.935**	0.940	0.874	0.931	0.933	0.938	0.919
	polyn. w/o f.	0.863	0.931	0.932	0.937	0.865	0.932	0.931	0.936	0.870	0.929	0.931	0.935	0.916
	ours with f.	**0.882**	**0.938**	**0.954**	0.940	**0.882**	**0.937**	0.935	0.940	0.879	**0.935**	0.934	0.939	**0.925**
	ours with unc.	0.862	0.934	0.935	0.939	0.873	0.935	0.935	0.939	0.874	0.933	0.932	0.938	0.919
	ours w/o f.	0.865	0.932	0.949	0.936	0.867	0.934	0.931	0.936	0.872	0.930	0.930	0.934	0.918

a big scale as in other scenarios (7% increase in Link Profile 4 vs. 13% and 21% increase in Link Profile 1 and 2 shown in Appendix C Table 14).

One sample accuracy evolution of slow CIFAR10 learning with Link Profile 1, Computation Profile f is plotted in Figure 6. For each way of computing aggregation weight, we see a clear elevation of accuracy when flexible strategy is enabled.

Learning CIFAR100:

We show in Figure 7 of the testing accuracy evolution of learning CIFAR100 with SqueezeNet. SqueezeNet is non-convex and therefore we do not apply our weight-computing method in Algorithm 2. Instead, we focus on the effect of flexible training/uploading actions. From Figure 7 we see that the testing accuracy of the global model increase faster when the flexible strategy is enabled. Moreover, uncertainties in the prediction of link status does not deteriorate the benefits brought by flexible strategy.

Verification of Proof of Convergence: Greater aggregation weight α_k leads to a smaller Q_1 and Q_2, and thus reduces the gap between the initialized model and the optimal model $F(\mathbf{x}_{k+1}) - F(\mathbf{x}_*)$ more quickly. However, smaller Q_1 also increases the term $O\left(\frac{1}{1-Q_1}\right)$ in Inequality 4. Therefore, the effect of aggregation weight cannot be determined explicitly. The number of aggregations k also have impact on multiple terms and its effect cannot be determined simply either. For example, when flexibility of training/uploading actions is enabled on clients, we noticed a decrease of total aggregations (k) throughout the FL tasks, which increases the loss term such as $O\left(D_0 \cdot Q_1^{k+1}\right)$ if Q_1 is kept steady. At the same time, the number of local optimizations (h) increases, thus reducing Q_1 and decreasing the loss. The improved performance of FL concludes that the increased h has stronger impact on the loss.

We instead focus on other indicators that have a single-directional impact on the loss and accuracy. The first one

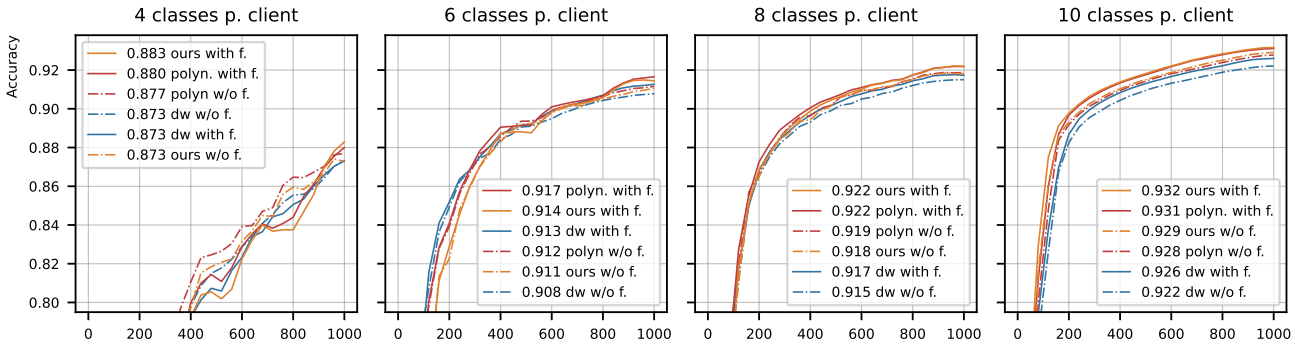


Fig. 5: FL tasks with MLP learning MNIST, in scenario with link profile 3 and computation profile a. Sampled evolving accuracy of different methods with data distribution std. 750 and different number of classes available at clients.

TABLE 9: Test accuracy after 1000 steps learning CIFAR10 with CNN in fast learning settings with Comp. Prof. a-c.

Data Distr.	D std.	750		1250		avg.
Scena -rio	# classes p. client	7	10	7	10	
Link Prof. 1 Comp. Prof. c	dw with f.	0.421	0.486	**0.423**	**0.488**	0.455
	dw with unc.	0.423	0.483	0.420	0.486	0.453
	dw w/o f.	0.410	0.472	0.407	0.475	0.441
	polyn. with f.	0.438	**0.508**	0.422	0.487	**0.464**
	polyn. with unc.	**0.442**	0.506	0.422	0.487	**0.464**
	polyn. w/o f.	0.422	0.493	0.415	0.477	0.452
	ours with f.	0.437	0.500	0.422	0.485	0.461
	ours with unc.	0.437	0.500	0.422	0.485	0.461
	ours w/o f.	0.422	0.494	0.422	0.480	0.454
Link Prof. 3 Comp. Prof. a	dw with f.	0.423	0.480	0.424	0.485	0.453
	dw with unc.	0.426	0.477	0.428	0.481	0.453
	dw w/o f.	0.408	0.462	0.409	0.464	0.436
	polyn. with f.	0.446	**0.504**	0.434	**0.491**	0.469
	polyn. with unc.	**0.449**	0.500	**0.442**	0.489	**0.470**
	polyn. w/o f.	0.428	0.483	0.421	0.471	0.451
	ours with f.	0.437	0.495	0.433	0.487	0.463
	ours with unc.	0.440	0.494	0.435	0.485	0.464
	ours w/o f.	0.426	0.483	0.423	0.473	0.451
Link Prof. 4 Comp. Prof. b	dw with f.	0.458	0.512	0.457	**0.514**	0.485
	dw with unc.	0.453	0.500	0.444	0.503	0.475
	dw w/o f.	0.456	0.501	0.455	0.504	0.479
	polyn. with f.	**0.482**	**0.531**	0.465	0.500	**0.495**
	polyn. with unc.	0.479	0.523	0.457	0.503	0.491
	polyn. w/o f.	0.478	0.521	0.467	0.503	0.492
	ours with f.	0.471	0.517	0.464	0.495	0.487
	ours with unc.	0.479	0.523	0.464	0.512	**0.495**
	ours w/o f.	0.476	0.521	**0.467**	0.509	0.493

TABLE 10: Test accuracy after 8000 steps learning CIFAR10 with CNN in slow learning settings with Comp. Prof. d-f.

Data Distr.	D std.	750		1250		avg.
Scena -rio	# classes p. client	4	7	4	7	
Link Prof. 1 Comp. Prof. f	dw with f.	0.441	0.484	0.437	0.483	0.461
	dw with unc.	0.439	0.478	**0.437**	0.481	0.459
	dw w/o f.	0.425	0.465	0.428	0.470	0.447
	polyn. with f.	**0.448**	0.510	0.433	0.492	0.471
	polyn. with unc.	0.440	0.503	0.430	0.494	0.467
	polyn. w/o f.	0.428	0.486	0.426	0.481	0.455
	ours with f.	0.443	**0.513**	0.430	0.500	**0.472**
	ours with unc.	0.434	0.506	0.437	**0.500**	0.469
	ours w/o f.	0.433	0.491	0.426	0.492	0.460
Link Prof. 3 Comp. Prof. d	dw with f.	0.442	0.474	0.451	0.480	0.462
	dw with unc.	0.442	0.473	**0.452**	0.482	0.462
	dw w/o f.	0.425	0.454	0.430	0.459	0.442
	polyn. with f.	0.440	0.502	0.416	0.488	0.461
	polyn. with unc.	0.452	0.503	0.426	0.494	0.469
	polyn. w/o f.	0.430	0.481	0.411	0.479	0.450
	ours with f.	0.449	0.504	0.417	0.493	0.465
	ours with unc.	**0.460**	**0.506**	0.435	**0.497**	**0.474**
	ours w/o f.	0.441	0.486	0.424	0.481	0.458
Link Prof. 4 Comp. Prof. e	dw with f.	0.485	0.534	**0.490**	0.530	0.510
	dw with unc.	0.480	0.529	0.481	0.531	0.505
	dw w/o f.	0.476	0.525	0.475	0.528	0.501
	polyn. with f.	0.483	0.554	0.459	0.541	0.509
	polyn. with unc.	0.464	0.550	0.466	0.543	0.506
	polyn. w/o f.	0.466	0.546	0.450	0.535	0.499
	ours with f.	**0.489**	**0.558**	0.480	0.549	**0.519**
	ours with unc.	0.471	0.554	0.488	**0.552**	0.516
	ours w/o f.	0.477	0.552	0.464	0.546	0.510

is ϕ, which is related to the data imbalance. From Table 7, Table 8, Table 9, and Table 10, we observe that there is a general trend of increasing testing accuracy when data is more balanced, where there are more classes available at a client and thus smaller ϕ leading to smaller loss in Inequality 4.

The other indicators are a and c, which are the parameters of function fitted from trial run before each set of experiments. By comparing the settings in Table 4, we notice that the value e^{a+c} is lower when learning MNIST with CNN. Considering that we have identical settings of α_{max} for both tasks leading to similar Q_1, we infer that the better learning capability of CNN is manifested in the lower values of e^{a+c} (the last term of Inequality 4), which in turn results in fewer errors and higher evaluation accuracy.

6 Conclusion

We propose a scheme that adapts the actions of FL clients to match their learning and model transmission actions to the status of link resources in the environment, aiming to save the time spent on communication and allow more time for training the local models. Moreover, the FL process is analyzed from the perspective of distributed convex optimization. The evolution of loss function during the learning process is decomposed into the reduction of initial error and the extra errors introduced during the aggregation of client updates. Based on the theoretic analysis, we customize the computation of aggregation weight based on model distances and the number of optimizations performed to get a client update, aiming to strike a balance between

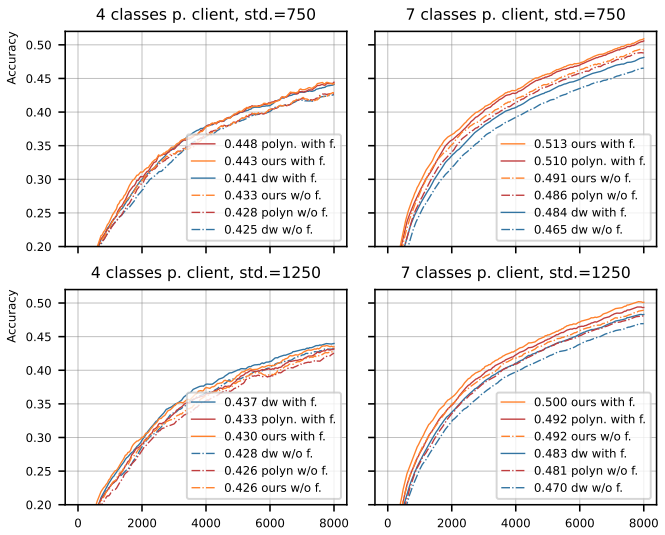
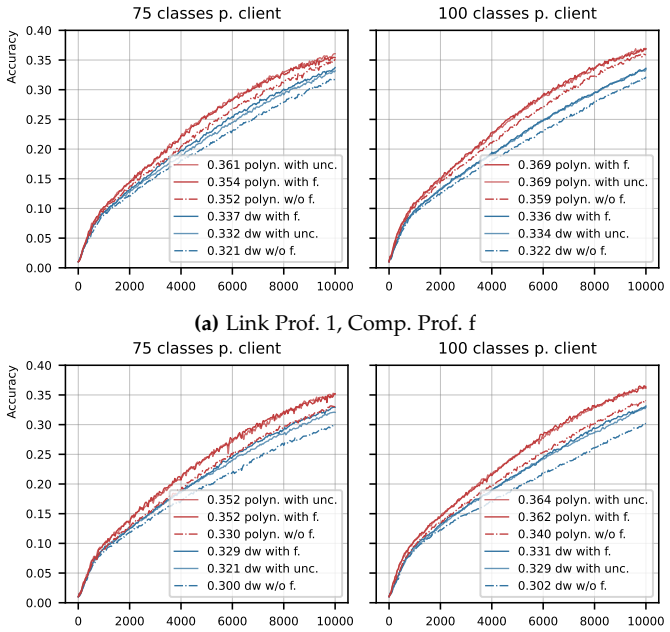


Fig. 6: FL tasks with CNN learning CIFAR10, in scenario with link profile 1 and computation profile f. Sampled evolving accuracy of different methods with data distribution std. 750 and 1250, 4 and 7 classes available at clients.



(a) Link Prof. 1, Comp. Prof. f

(b) Link Prof. 3, Comp. Prof. d

Fig. 7: FL tasks with SqueezeNet learning CIFAR100, in scenario with (a) link profile 1 and computation profile f (b) link profile 3 and computation profile d. Sampled evolving accuracy of different methods with data distribution std. 750, either 75 and 100 classes available at clients.

controlling the aggregation error and minimizing the initial loss.

We show through experiments that our solution is more effective in dynamic communication environments. Statistical analysis on training and uploading actions shows that our solution – by targeting opportune link conditions – measurably decreases the overall consumption of communication resources by at least 5%, reduces the transmission time by 13% on average. Thanks to the decrease in communication time, the computational resources invested are generally increased by 10%, which in turn improves the predicting accuracy of the learned model by 3% and 0.5% for CIFAR10 and MNIST respectively. The improvement brought by the flexibility is especially prominent when the task is more computationally resource-demanding (CIFAR10). When combined with our newly-proposed method for computing aggregation weight, the FL with flexibility has improved performance by 3.8% for learning CIFAR10 and 1% for learning MNIST.

Our solution is especially suitable for scenarios with ample computational resources and dynamic communication conditions. Future research directions include adding data imbalance factors into aggregation weight design, adapting FL clients' training speed based on learning progress, and more accurate estimation of task loss to enable a more accurate aggregation weight design.

References

[1] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[2] F. Dressler, C. F. Chiasserini, F. H. P. Fitzek, et al., "V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing," *IEEE Network*, vol. 36, no. 3, pp. 24–31, May 2022.

[3] W. Y. B. Lim, N. C. Luong, D. T. Hoang, et al., "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[4] J. C. Mertens, L. Galluccio, and G. Morabito, "Federated learning through model gossiping in wireless sensor networks," in *9th IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom 2021)*, Bucharest, Romania: IEEE, May 2021, pp. 1–6.

[5] J. Nguyen, K. Malik, H. Zhan, et al., "Federated Learning with Buffered Asynchronous Aggregation," in *25th International Conference on Artificial Intelligence and Statistics (AISTATS 2022)*, G. Camps-Valls, F. Ruiz, and I. Valera, Eds., vol. 151, Valencia, Spain: PMLR, Mar. 2022, pp. 3581–3607.

[6] T. H. Nguyen, W. Bao, A. Y. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated Learning over Wireless Networks: Optimization Model Design and Analysis," in *38th IEEE Conference on Computer Communications (INFOCOM 2019)*, Paris, France: IEEE, Apr. 2019, pp. 1387–1395.

[7] H. Sun, X. Ma, and R. Q. Hu, "Adaptive Federated Learning With Gradient Compression in Uplink NOMA," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 16 325–16 329, Dec. 2020.

[8] M. Boban, D. Dupleich, N. Iqbal, et al., "Multi-Band Vehicle-to-Vehicle Channel Characterization in the Presence of Vehicle Blockage," *IEEE Access*, vol. 7, pp. 9724–9735, Jan. 2019.

[9] Y. Guo, R. Zhao, S. Lai, L. Fan, X. Lei, and G. K. Karagiannidis, "Distributed Machine Learning for Multiuser Mobile Edge Computing Systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 3, pp. 460–473, 2022.

[10] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *IEEE International Conference on Communications (ICC 2019)*, Shanghai, China: IEEE, May 2019.

[11] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[12] J. Wang, S. Wang, R.-R. Chen, and M. Ji, "Demystifying why local aggregation helps: Convergence analysis of hierarchical SGD," in *36th AAAI Conference on Artificial Intelligence (AAAI-22)*, vol. 36, Virtual Conference: AAAI Press, Feb. 2022, pp. 8548–8556.

[13] Y. He, M. Yang, Z. He, and M. Guizani, "Computation Offloading and Resource Allocation Based on DT-MEC-Assisted Federated Learning Framework," *IEEE Transactions on Cognitive Communications and Networking*, vol. 9, no. 6, pp. 1707–1720, Dec. 2023.

[14] S. Wang, M. Chen, C. G. Brinton, C. Yin, W. Saad, and S. Cui, "Performance Optimization for Variable Bitwidth Federated Learning in Wireless Networks," *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 2340–2356, Mar. 2024.

[15] J. Feng, W. Zhang, Q. Pei, J. Wu, and X. Lin, "Heterogeneous Computation and Resource Allocation for Wireless Powered Federated Edge Learning Systems," *IEEE Transactions on Communications*, vol. 70, no. 5, pp. 3220–3233, 2022.

[16] M. Wu, M. Boban, and F. Dressler, "Parameter-less Asynchronous Federated Learning under Computation and Communication Constraints," in *97th IEEE Vehicular Technology Conference (VTC 2023-Spring)*, Florence, Italy: IEEE, Jun. 2023, pp. 1–7.

[17] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, Fort Lauderdale, FL: PMLR, Apr. 2017, pp. 1273–1282.

[18] Z. Qu, K. Lin, J. Kalagnanam, Z. Li, J. Zhou, and Z. Zhou, "Federated Learning's Blessing: FedAvg has Linear Speedup," arXiv, cs.LG 2007.05690, Jul. 2020.

[19] H. Jin, N. Yan, and M. Mortazavi, "Simulating Aggregation Algorithms for Empirical Verification of Resilient and Adaptive Federated Learning," in *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT 2020)*, Leicester, United Kingdom: IEEE, Dec. 2020.

[20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *3rd Conference on Machine Learning and Systems (MLSys 2020)*, Austin, TX, Mar. 2020.

[21] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning With Low Overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, May 2021.

[22] H. Guo, S. Li, B. Li, Y. Ma, and X. Ren, "A New Learning Automata-Based Pruning Method to Train Deep Neural Networks," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3263–3269, Oct. 2018.

[23] K. Pfeiffer, M. Rapp, R. Khalili, and J. Henkel, "CoCoFL: Communication- and Computation-Aware Federated Learning via Partial NN Freezing and Quantization," *Transactions on Machine Learning Research*, Jun. 2023.

[24] J. Mills, J. Hu, and G. Min, "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.

[25] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, and D. B. Ananda Theertha Suresh and, "Federated Learning: Strategies for Improving Communication Efficiency," arXiv, cs.LG 1610.05492, Oct. 2016.

[26] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication," in *International Joint Conference on Neural Networks (IJCNN 2019)*, Budapest, Hungary: IEEE, Jul. 2019.

[27] Z. Wang, Z. Zhang, Y. Tian, et al., "Asynchronous Federated Learning Over Wireless Communication Networks," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6961–6978, Sep. 2022.

[28] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "FedSA: A Semi-Asynchronous Federated Learning Mechanism in Heterogeneous Edge Computing," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3654–3672, Dec. 2021.

[29] X. Wu and C.-L. Wang, "KAFL: Achieving High Training Efficiency for Fast-K Asynchronous Federated Learning," in *42nd IEEE International Conference on Distributed Computing Systems (ICDCS 2022)*, Bologna, Italy: IEEE, Jul. 2022, pp. 873–883.

[30] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," arXiv, cs.DC 1903.03934, Mar. 2019.

[31] A. Imteaj and M. H. Amini, "FedPARL: Client Activity and Resource-Oriented Lightweight Federated Learning Model for Resource-Constrained Heterogeneous IoT Environment," *Frontiers in Communications and Networks*, vol. 2, Apr. 2021.

[32] Z. Zhou, Y. Li, X. Ren, and S. Yang, "Towards Efficient and Stable K-Asynchronous Federated Learning With Unbounded Stale Gradients on Non-IID Data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3291–3305, Dec. 2022.

[33] K. Bäckström, M. Papatriantafilou, and P. Tsigas, "ASAP.SGD: Instance-based Adaptiveness to Staleness in Asynchronous SGD," in *39th International Conference on Machine Learning (ICML 2022)*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, Baltimore, MD: PMLR, Jul. 2022, pp. 1261–1276.

[34] S. P. Singh and M. Jaggi, "Model Fusion via Optimal Transport," in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., Virtual Conference: Curran Associates Inc., Dec. 2020, pp. 22 045–22 055.

[35] Q. Li, B. He, and D. Song, "Model-Contrastive Federated Learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2021)*, Nashville, TN: IEEE, Jun. 2021, pp. 10 708–10 717.

[36] Y. Yu, A. Wei, S. P. Karimireddy, Y. Ma, and M. Jordan, "TCT: Convexifying Federated Learning using Bootstrapped Neural Tangent Kernels," in *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, New Orleans, LA: Curran Associates Inc., Nov. 2022, pp. 30 882–30 897.

[37] C. Xie, S. Koyejo, and I. Gupta, "Zeno++: Robust Fully Asynchronous SGD," in *37th International Conference on Machine Learning (PMLR 2020)*, Virtual Conference, Jul. 2020, pp. 10 495–10 503.

[38] T. Zeng, O. Semiari, W. Saad, and M. Bennis, "Wireless-Enabled Asynchronous Federated Fourier Neural Network for Turbulence Prediction in Urban Air Mobility (UAM)," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2023.

[39] J. Park, D.-J. Han, M. Choi, and J. Moon, "Sageflow: Robust Federated Learning against Both Stragglers and Adversaries," in *35th International Conference on Neural Information Processing Systems (NeurIPS 2021)*, M. a. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Virtual Conference: Curran Associates Inc., Dec. 2021, pp. 840–851.

[40] S. Park and O. Simeone, "Predicting Flat-Fading Channels via Meta-Learned Closed-Form Linear Filters and Equilibrium Propagation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2022)*, Singapore, Singapore: IEEE, May 2022, pp. 8817–8821.

[41] M. Boban, M. Giordani, and M. Zorzi, "Predictive Quality of Service: The Next Frontier for Fully Autonomous Systems," *IEEE Network*, vol. 35, no. 6, pp. 104–110, Nov. 2021.

[42] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data," in *IEEE International Conference on Big Data (BigData 2020)*, Virtual Conference: IEEE, Dec. 2020, pp. 15–24.

[43] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Researc," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[44] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Canada, Technical Report TR-2009, Apr. 2009.

[45] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," DeepScale, arXiv, Nov. 2016. [Online]. Available: https://arxiv.org/abs/1602.07360.

APPENDIX A
DETAILS OF PROOF OF INEQUALITIES

A.1 Proof of Corollary 4.1

$$
\begin{aligned}
&\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k}\right)-F(\mathbf{x}_*)\right]\\
&=\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k}\right)-F\left(\mathbf{x}_{\tau_k,h_k-1}\right)+F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]\\
&\leq \mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]+\mathbb{E}\left[\left\langle\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right),\mathbf{x}_{\tau_k,h_k}-\mathbf{x}_{\tau_k,h_k-1}\right\rangle+\frac{L}{2}\left\|\mathbf{x}_{\tau_k,h_k}-\mathbf{x}_{\tau_k,h_k-1}\right\|^2\right]\\
&=\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]+\mathbb{E}\left[\left\langle\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right),-\gamma\frac{1}{B}\sum_{d\in D_B}\nabla f(\mathbf{x}_{\tau_k,h_k-1},d)\right\rangle+\underbrace{\frac{L}{2}\left\|\gamma\frac{1}{B}\sum_{d\in D_B}\nabla f(\mathbf{x}_{\tau_k,h_k-1},d)\right\|^2}_{\text{Set }\gamma\leq\frac{1}{L}}\right]\\
&=\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]+\frac{1}{2}\gamma\cdot\mathbb{E}\left[\underbrace{\left\|\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-\frac{1}{B}\sum_{d\in D_B}\nabla f(\mathbf{x}_{\tau_k,h_k-1},d)\right\|^2}_{\text{Bounded data imbalance}}-\underbrace{\left\|\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right)\right\|^2}_{\text{Convert with PL inequality}}\right]\\
&\leq\mathbb{E}\left[(1-\gamma\mu)\cdot\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]\right]\\
&+\frac{1}{2}\gamma\cdot\mathbb{E}\left[\left\|\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-\nabla f(\mathbf{x}_{\tau_k,h_k-1})+\nabla f(\mathbf{x}_{\tau_k,h_k-1})-\frac{1}{B}\sum_{d\in D_B}\nabla f(\mathbf{x}_{\tau_k,h_k-1},d)\right\|^2\right]\\
&\leq\mathbb{E}\left[(1-\gamma\mu)\cdot\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]\right]\\
&+\gamma\cdot\mathbb{E}\left[\left\|\nabla F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-\nabla f(\mathbf{x}_{\tau_k,h_k-1})\right\|^2+\left\|\nabla f(\mathbf{x}_{\tau_k,h_k-1})-\frac{1}{B}\sum_{d\in D_B}\nabla f(\mathbf{x}_{\tau_k,h_k-1},d)\right\|^2\right]\\
&\leq\mathbb{E}\left[(1-\gamma\mu)\cdot\left[F\left(\mathbf{x}_{\tau_k,h_k-1}\right)-F(\mathbf{x}_*)\right]\right]+\gamma(\phi^2+\sigma^2)\\
&\leq(1-\gamma\mu)^{h_k}\cdot\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,0}\right)-F(\mathbf{x}_*)\right]+\sum_{i=0}^{h_k-1}\gamma(\phi^2+\sigma^2)(1-\gamma\mu)^i\\
&\leq(1-\gamma\mu)^{h_k}\cdot\mathbb{E}\left[F\left(\mathbf{x}_{\tau_k,0}\right)-F(\mathbf{x}_*)\right]+(\phi^2+\sigma^2)\cdot\frac{1}{\mu}
\end{aligned}
$$

A.2 Complete Inequality Derivation

$$
\begin{aligned}
&\mathbb{E}[F(\mathbf{x}_{k+1}) - F(\mathbf{x}_*)] \\
&= \mathbb{E}[F((1-\alpha_k)\cdot \mathbf{x}_k + \alpha_k \cdot \mathbf{x}_{\tau_k,h_k}) - F(\mathbf{x}_*)] \\
&\le (1-\alpha_k)\cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \mathbb{E}[F(\mathbf{x}_{\tau_k,h_k}) - F(\mathbf{x}_*)] \\
&\le (1-\alpha_k)\cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left[(1-\gamma\mu)^{h_k} \cdot \mathbb{E}[F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_*)] + \frac{\phi^2+\sigma^2}{\mu}\right] \\
&= (1-\alpha_k)\cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left[(1-\gamma\mu)^{h_k} \cdot \mathbb{E}[F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k) + F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \frac{\phi^2+\sigma^2}{\mu}\right] \\
&\le \left[(1-\alpha_k) + \alpha_k \cdot (1-\gamma\mu)^{h_k}\right] \cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left[(1-\gamma\mu)^{h_k} \cdot \mathbb{E}[F(\mathbf{x}_{\tau_k}) - F(\mathbf{x}_k)] + \frac{\phi^2+\sigma^2}{\mu}\right] \\
&\le \left[(1-\alpha_k) + \alpha_k \cdot (1-\gamma\mu)^{h_k}\right] \cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left[\frac{(1-\gamma\mu)^{h_k}}{2} \cdot \mathbb{E}\left[\|\nabla F(\mathbf{x}_k)\|^2 + (1+L)\cdot \|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2\right] + \frac{\phi^2+\sigma^2}{\mu}\right] \\
&\le \left[(1-\alpha_k) + \alpha_k \cdot (1-\gamma\mu)^{h_k}\right] \cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left[\frac{(1-\gamma\mu)^{h_k}}{2} \cdot \mathbb{E}\left[G^2 + (1+L)\cdot \|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2\right] + \frac{\phi^2+\sigma^2}{\mu}\right]
\end{aligned}
$$

Inquality 2 proved

$$
\begin{aligned}
&\le Q_1 \cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \frac{1}{2}\alpha_k \cdot (1-\gamma\mu)^{h_k} \cdot G^2 + \frac{1+L}{2}\alpha_k \cdot (1-\gamma\mu)^{h_k} \cdot \|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2 + \alpha_k \cdot \frac{\phi^2+\sigma^2}{\mu} \\
&\le Q_1 \cdot \mathbb{E}[F(\mathbf{x}_k) - F(\mathbf{x}_*)] + \alpha_k \cdot \left(\frac{(1-\gamma\mu)^{h_k}}{2} G^2 + \frac{\phi^2+\sigma^2}{\mu}\right) + \frac{1+L}{2} e^{a\cdot e^{bk}+c}
\end{aligned}
$$

Use $\alpha_k \le \dfrac{\exp\left(a \cdot e^{bk} + c\right)}{(1-\gamma\mu)^{h_k} \cdot \|\mathbf{x}_{\tau_k} - \mathbf{x}_k\|^2}$

$$
\begin{aligned}
&\le Q_1 \cdot \left\{Q_1 \cdot \mathbb{E}[F(\mathbf{x}_{k-1}) - F(\mathbf{x}_*)] + \alpha_{k-1} \cdot \left(\frac{(1-\gamma\mu)^{h_{k-1}}}{2} \cdot G^2 + \frac{\phi^2+\sigma^2}{\mu}\right) + \frac{1+L}{2} e^{a\cdot e^{b(k-1)}+c}\right\} \\
&\quad + \alpha_k \cdot \left(\frac{(1-\gamma\mu)^{h_k}}{2} \cdot G^2 + \frac{\phi^2+\sigma^2}{\mu}\right) + \frac{1+L}{2} e^{a\cdot e^{bk}+c}
\end{aligned}
$$

Expand $F(\mathbf{x}_k) - F(\mathbf{x}_*)$ in similar way

$$
\begin{aligned}
&= Q_1^2 \cdot \mathbb{E}[F(\mathbf{x}_{k-1}) - F(\mathbf{x}_*)] + \sum_{j=k-1}^{k} Q_1^{k-j} \cdot \left\{\alpha_j \cdot \left(\frac{(1-\gamma\mu)^{h_j}}{2} \cdot G^2 + \frac{\phi^2+\sigma^2}{\mu}\right) + \frac{1+L}{2} e^{a\cdot e^{bj}+c}\right\} \\
&\le Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \sum_{j=0}^{k} Q_1^{k-j} \cdot \left\{\alpha_j \cdot \left(\frac{(1-\gamma\mu)^{h_j}}{2} \cdot G^2 + \frac{\phi^2+\sigma^2}{\mu}\right) + \frac{1+L}{2} I(j)\right\} \\
&\le Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \sum_{j=0}^{k} Q_1^{k-j} \cdot \left\{\frac{(Q_1 - 1 + \alpha_j)\cdot G^2}{2} + \alpha_j \cdot \frac{\phi^2+\sigma^2}{\mu}\right\} + \sum_{j=0}^{k} Q_1^{k-j} \cdot \frac{1+L}{2} J(j)
\end{aligned}
$$

Use $Q_1 \ge (1-\alpha_k) + \alpha_k \cdot (1-\gamma\mu)^{h_k}$

Substitute $I(\cdot)$ with constructed geometric progression $J(\cdot)$

$$
\le Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \frac{1-Q_1^{k+1}}{1-Q_1} \cdot \left\{\frac{Q_1 \cdot G^2}{2} + \frac{\alpha_{max} \cdot (\phi^2+\sigma^2)}{\mu}\right\} + \sum_{j=0}^{k} Q_1^{k-j} \cdot \frac{1+L}{2} \cdot J(0) \cdot r^j
$$

Take upper bound: $Q_1 - 1 + \alpha_j \le Q_1,\ \alpha_j \le \alpha_{max}$

$$
\le Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \frac{1-Q_1^{k+1}}{1-Q_1} \cdot \left\{\frac{Q_1 \cdot G^2}{2} + \frac{\alpha_{max} \cdot (\phi^2+\sigma^2)}{\mu}\right\} + \sum_{j=0}^{k} Q_2^{k} \cdot \frac{1+L}{2} \cdot e^{a+c}
$$

Use $Q_2 = \max\{Q_1, r\}$

$$
= Q_1^{k+1} \cdot \mathbb{E}[F(\mathbf{x}_0) - F(\mathbf{x}_*)] + \frac{1-Q_1^{k+1}}{1-Q_1} \cdot \left\{\frac{Q_1 \cdot G^2}{2} + \frac{\alpha_{max} \cdot (\phi^2+\sigma^2)}{\mu}\right\} + (1+k) \cdot \frac{Q_2^{k} \cdot (1+L) \cdot e^{a+c}}{2}
$$

APPENDIX B
MACHINE LEARNING MODEL STRUCTURE

B.1 2-Layer Perceptron for Convex Classification of MNIST

TABLE 11: Model Structure for 2-Layer Perceptron for MNIST classification

Layer	Parameters	Input Layer
flatten	/	data
fully_connected1	output=1024	flatten
activation1	ReLU	fully_connected1
fully_connected2	output=1024	activation1
activation2	ReLU	fully_connected2
fully_connected3	output=10	activation2
activation3	ReLU	fully_connected3

B.2 Convolutional Neural Network for MNIST Classification

TABLE 12: Model Structure for CNN used for MNIST classification

Layer	Parameters	Input Layer
conv1	out_channel=10, kernel_size=5, padding=0	data
activation1	ReLU	conv1
pooling1	pool_size=2, stride=2	activation1
conv2	out_channel=20, kernel_size=5, padding=0	pooling1
activation2	ReLU	conv2
pooling2	pool_size=2, stride=2	activation2
flatten	/	pooling2
fully_connected1	output=50	flatten
activation3	ReLU	fully_connected1
fully_connected2	output=10	activation3
activation4	ReLU	fully_connected2

B.3 Convolutional Neural Network for CIFAR10 Classification

TABLE 13: Model Structure for CNN used for CIFAR10 classification

Layer	Parameters	Input Layer
conv1	out_channel=6, kernel_size=5, padding=0	data
activation1	ReLU	conv1
pooling1	pool_size=2, stride=2	activation1
conv2	out_channel=16, kernel_size=5, padding=0	pooling1
activation2	ReLU	conv2
pooling2	pool_size=2, stride=2	activation2
flatten	/	pooling2
fully_connected1	output=120	flatten
activation3	ReLU	fully_connected1
fully_connected2	output=84	activation3
activation4	ReLU	fully_connected2
fully_connected3	output=10	activation4

APPENDIX C
EXPERIMENT RESULTS

C.1 Resource Usage of FL tasks with CIFAR10

TABLE 14: Comparison of Statistics of Time and Resource Usage of FL clients in various Scenarios of Classifying CIFAR10

	Avg. Transmission Time		Comm. Resource Usage	
	with f.	w/o f.	with f.	w/o f.
Link Profile 1 Comp. Profile c	18.3	20.4	0.373	0.394
Link Profile 2 Comp. Profile b	25.4	30.0	0.355	0.377
Link Profile 3 Comp. Profile a	7.6	7.6	0.152	0.200
	Avg. Training Time		Avg. Num. Optimization per client	
	with f.	w/o f.	with f.	w/o f.
Link Profile 1 Comp. Profile c	33.2	28.6	26665	23652
Link Profile 2 Comp. Profile b	35.1	27.4	25114	20800
Link Profile 3 Comp. Profile a	34.1	24.4	35248	32917