# AUTOMATIC TUNING OF FEDERATED LEARNING HYPER-PARAMETERS FROM SYSTEM PERSPECTIVE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Federated learning (FL) is a distributed model training paradigm that preserves clients' data privacy. FL hyper-parameters significantly affect the training overheads in terms of time, computation, and communication. However, the current practice of manually selecting FL hyper-parameters puts a high burden on FL practitioners since various applications prefer different training preferences. In this paper, we propose FedTuning, an automatic FL hyper-parameter tuning algorithm tailored to applications' diverse system requirements of FL training. FedTuning is lightweight and flexible, achieving an average of 41% improvement for different training preferences on time, computation, and communication compared to fixed FL hyper-parameters.

## 1 INTRODUCTION

Federated learning (FL) has been applied to a wide range of applications, including mobile keyboard (Hard et al., 2019), speech recognition (Paulik et al., 2021), and human stroke prevention (Ju et al., 2020). Compared to other paradigms of model learning (e.g., centralized machine learning (Jordan & Mitchell, 2015) and conventional distributed machine learning (Verbraeken et al., 2020)), FL has unique properties in terms of (1) *Massively Distributed*: the number of clients is much larger than the clients' average number of data points; (2) *Unbalanced Data*: clients have a different amount of data points; and (3) *Non-IID Data*: each client's data cannot represent the overall distribution (McMahan et al., 2017). In addition to the common hyper-parameters of model training such as learning rates, optimizers, and mini-batch sizes, FL has unique hyper-parameters, including aggregation algorithms and participant selection (Wang et al., 2021). Nonetheless, many FL algorithms, e.g., FedAvg (McMahan et al., 2017), have been proved to converge to the global optimum even different FL hyper-parameters are adopted (Li et al., 2020b; Wang et al., 2020b).

Although FL hyper-parameters do not invalidate FL convergence (i.e., the same final global model and accuracy), they significantly affect the training overheads of reaching the final model. Time, computation, and communication are the most important system overheads.

- *Time overhead*. It measures how long an FL system takes to train a final model. For applications that require fast adaptation to environments (e.g., security problems), overall training time must be short.
- *Computation overhead*. It measures computation-related consumption, such as FLOPs, CPU/GPU usage, energy, or carbon dioxide emissions. For low-profile devices such as Internet-of-Things (IoT) nodes, computation overhead must be small as these clients are equipped with weak hardware and/or powered on batteries.
- *Communication overhead*. It measures data transmission between the server and clients. Communication overhead is critical when a free and high-speed transmission is not available. For example, outdoor applications usually rely on cellular communications, which need to pay a considerable price for uploading/downloading a large amount of data.

Application scenarios have different training preferences regarding time, computation, and communication overheads. (1) Attack and anomaly detection in computer networks (Haji & Ameen, 2021) is time-sensitive, as it needs to adapt to malicious traffic rapidly; (2) Smart home control systems for indoor environment automation (Mekuria et al., 2021), e.g., Heating, Ventilation, and Air Conditioning (HVAC), are sensitive to computational overhead because IoT devices are limited in computation

capabilities; (3) A traffic monitoring system for vehicles (Won, 2020) is communication-sensitive because cellular communications are usually adopted to provide city-scale connectivity. Many applications are sensitive to more than one system aspects. (4) Precision agriculture based on IoT sensing (Sharma et al., 2020) is both computation and communication sensitive; (5) Home healthcare systems for elderly people (Hassan et al., 2019), e.g., fall detection, are both time and computation sensitive; and (6) Human stampedes detection/prevention (de Almeida & von Schreeb, 2018) require time, computation, and communication efficient systems.

There is a plethora of work that have studied FL training performance under different hyper-parameters (Wang et al., 2021). However, they do not consider time, computation, and communication altogether, which are essential from the system's perspective. In addition, it is challenging to tune multiple hyper-parameters in order to achieve diverse training preferences, especially when we need to optimize multiple system aspects. For example, it is unclear how to select hyper-parameters to build an FL training solution that is both time and computation-efficient.

**Contributions.** In this paper, we formulate time, computation, and communication overheads and conduct extensive measurements to understand FL training performance. We investigate three of the most important FL hyper-parameters: the number of clients selected on each round, the number of training passes that each client makes over its local data on each round, and model complexity. Our measurement results show that time, computation, and communication compete for conflicting FL hyper-parameters. To alleviate the burden of manual hyper-parameter selection, we propose FedTuning, an algorithm that automatically tunes FL hyper-parameters to meet application training preference on time, computation, and communication. This paper sheds light on optimizing FL hyper-parameters from the system perspective.

## 2 Background and System Model

FedAvg (McMahan et al., 2017) is a standard FL algorithm that has been widely used by the community. It updates the global model by aggregating clients' model parameters with weights proportional to each client's number of local data points. Equivalently, FedAvg minimizes the following objective

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w) \tag{1}$$

where $f_i(w)$ is the loss of the model on data point $(x_i, y_i)$, that is, $f_i(w) = \ell(x_i, y_i; w)$, $K$ is the total number of clients, $\mathcal{P}_k$ is the set of indexes of data points on client $k$, with $n_k = |\mathcal{P}_k|$, and $n$ is the total number of data points from all clients, i.e., $n = \sum_{k=1}^{K} n_k$. Due to a large number of clients in a typical FL application (e.g., millions of clients in Google Gboard (Hard et al., 2019)), a common practice is to randomly select a small fraction of clients on each training round. In the rest of this paper, we refer to selected clients as participants and denote by $M$ the number of participants on each training round. Each participant makes $E$ training passes over its local data on each round before uploading its model parameters to the server for aggregation. Afterward, participants wait to receive an updated global model from the server, and a new training round starts. FedAvg has been proved to converge to the global optimum on Non-IID data (Li et al., 2020b; Wang et al., 2020b). In other words, FL hyper-parameters do not affect the final model accuracy when the same model is applied. In this paper, we use FedAvg to illustrate our hyper-parameter tuning algorithm. We optimize three hyper-parameters of FedAvg: the number of participants $M$, the number of training passes $E$, and model complexity.

Hyper-Parameter Optimization (HPO) is a field that has been extensively studied (Yang & Shami, 2020). Many classical HPO algorithms, e.g., Bayesian optimization (Snoek et al., 2012), successive halving (Karnin et al., 2013), and hyperband (Li et al., 2017), are designed to optimize hyper-parameters of machine learning models. However, they cannot be directly applied to our scenario of optimizing FL hyper-parameter for different FL training preferences. (1) Time (in seconds), computation (in FLOPs), and communication (in bytes) are not comparable with each other. Incorporating training preferences in HPO is not trivial. (2) Hyper-parameter tuning needs to be done during the FL training. No "comeback" is allowed as the FL model keeps training until its final model accuracy. Otherwise, it will cause significantly more system overheads.

## 3 SYSTEM OVERHEAD FORMULATION

We apply the same configurations of FedAvg to formulate FL system overheads. Assume that clients are homogeneous regarding hardware (e.g., CPU/GPU). Let $b_{k,r}$ indicates whether client $k$ participates at the training round $r$. Then, we have $\sum_{k=1}^{K} b_{k,r} = M$, i.e., each round selects $M$ participants. The number of training rounds to reach the final model accuracy is denoted by $R$, which is unknown *a priori* and varies when different sets of FL hyper-parameters are used in FL training. The time, computation, and communication overheads of an FL training can be formulated as follows.

**Time Overhead**. If client $k$ is selected on a training round, it spends time in local training and model parameters transmission to/from the server. The local training delay can be represented by $C_1 \cdot E \cdot n_k$, where $C_1$ is a constant. It is proportional to its number of data points (i.e., $n_k$) because $n_k$ decides the number of local updates (number of mini-batches) for one-epoch, and each local update includes one forward-pass and one backward-pass. The transmission delay is constant, denoted by $C'$, since each participant involves one download and one upload of model parameters from/to the server. Therefore, participant $k$ takes time of $C_1 \cdot E \cdot n_k + C'$ on each training round. The time length of the training round $r$ is determined by the slowest participant and thus is represented by $\max_{k=1}^{K} b_{k,r} \cdot (C_1 \cdot E \cdot n_k + C')$. In total, the time overhead of the FL training can be formulated as

$$TIME = \sum_{r=1}^{R} \max_{k=1}^{K} b_{k,r} \cdot (C_1 \cdot E \cdot n_k + C') \tag{2}$$

The transmission delay $C'$ is a constant adding to each participant's time length on each training round. Thus it does not affect the comparison of different configurations of FL hyper-parameters. We assign $C'$ to zero in this paper.

**Computation Overhead**. Similarly, if client $k$ is selected on a training round, it causes $C_2 \cdot E \cdot n_k$ computation cost, where $C_2$ is a constant. The computation cost of the training round $r$ is the summation of each participant's computation cost and thus is $C_2 \cdot E \cdot \sum_{k=1}^{K} b_{k,r} \cdot n_k$. In total, the computation overhead can be represented by

$$COMP = C_2 \cdot E \cdot \sum_{r=1}^{R} \sum_{k=1}^{K} b_{k,r} \cdot n_k \tag{3}$$

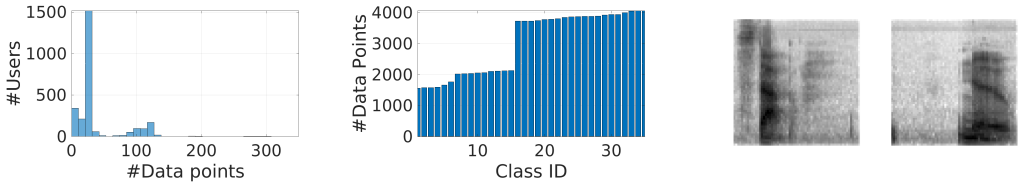
**Communication Overhead**. Since each training round selects $M$ participants, the communication cost for a training round is $C_3 \cdot M$ where $C_3$ is a constant. The total number of training rounds is $R$, and thus, the total communication overhead is represented by

$$COMM = C_3 \cdot R \cdot M \tag{4}$$

As we assume that clients are equipped with the same hardware, clients have the same $C_1$, $C_2$, and $C_3$. Thus, these constants do not affect the comparison of training overheads under different hyper-parameters when the same model is used. For studying different model complexities, $C_1$ and $C_2$ are weighted by models' FLOPs for one input, and $C_3$ is weighted by models' number of parameters.

## 4 MEASUREMENT STUDY

We conduct measurements to study the relationship among time, computation, and communication overheads when different FL hyper-parameters are used for training. We use the Google speech-to-command dataset (Warden, 2018). which classifies one second's audio clip into 35 commands such as yes, no, right, and up. The dataset includes audio clips that are crowd-sourced from 2618 clients. As officially suggested (Warden, 2018), we use 2112 clients' data for training and the remaining 506 clients' data for testing. Fig. 1(a) shows the distribution of the number of clients versus the number of data points. It is clear that clients' data are heterogeneous: many clients have only one data point, while others can have up to 316 data points. Fig. 1(b) plots the histogram of each class's number of data points, which shows that the overall data distribution is unbalanced. The speech-to-command dataset meets the three data properties of FL: massively distributed, unbalanced, and non-IID. In the measurement study, we investigate the FL training overheads regarding the following three hyper-parameters.




(a) User distribution (b) Class distribution (c) Example spectrograms

Figure 1: Google speech-to-command dataset used in measurements.

| Model | ResNet-10 | ResNet-18 | ResNet-26 | ResNet-34 |
|---|---|---|---|---|
| #BasicBlock | [1, 1, 1, 1] | [2, 2, 2, 2] | [3, 3, 3, 3] | [3, 4, 6, 3] |
| #FLOP ($\times 10^6$) | 12.5 | 26.8 | 41.1 | 60.1 |
| #Params ($\times 10^3$) | 79.7 | 177.2 | 274.6 | 515.6 |
| Accuracy | 0.88 | 0.90 | 0.90 | 0.92 |

Table 1: Different model complexities used for the speech-to-command dataset.

- The number of participants (i.e., $M$). It is well-known that more participants on each training round have a better round-to-accuracy performance (McMahan et al., 2017). In the measurement study, we set $M$ to 1, 10, 20, and 50.
- The number of training passes (i.e., $E$). Increasing the number of training passes as a method to improve communication efficiency has been adopted in several works, such as FedAvg (McMahan et al., 2017) and FedNov (Li et al., 2020b). In the measurement study, we set $E$ to 0.5, 1, 2, 4, 8, where 0.5 means that only half of each client's local data are used for local training on each round.
- Model complexity. We also investigate that, if a target accuracy is met, how does the model complexity influence the training overheads. We use ResNet (He et al., 2016) to build different models. Table 1 tabulates the details of the ResNet models.

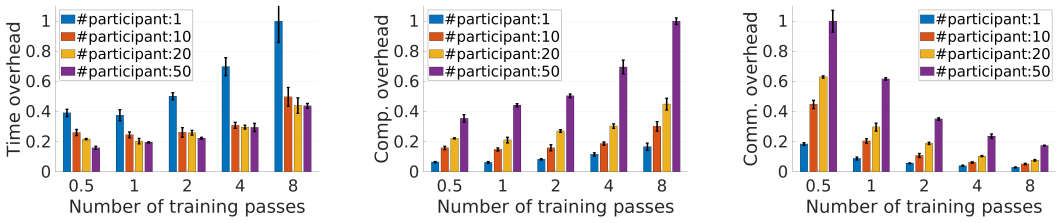
We transform audio clips of the speech-to-command dataset to 64-by-64 spectrograms and then downsize them to 32-by-32 images. Fig. 1(c) illustrates two spectrograms while the left one is from command stop and the right one is from command no. We normalize the input images with the mean (0.627) and the standard deviation (0.224) of the training dataset before feeding them to models for training and testing. We implement FedAvg in PyTorch. In our measurement study, we set the mini-batch size to 5, considering that many clients have few data points. We set the learning rate to 0.01, with a momentum of 0.9. The target model accuracy is set to 0.8. The results are averaged by three experiments. All experiments are conducted in a server with 24-GB Nvidia RTX A5000 GPUs.

### 4.1 Measurement Results

**Time.** Fig. 2(a) compares the time overhead for a different number of participants $M$ and a different number of training passes $E$. In the experiments, we use ResNet-18 and normalize their overheads. As we can see, more participants lead to smaller time overhead, i.e., it takes a shorter time to converge. However, the difference is not significant among 10, 20, and 50 participants, especially when the number of training passes is big. In addition, we can see that larger $E$ has worse time overheads. There is no apparent difference between $E = 0.5$ and $E = 1$ though.

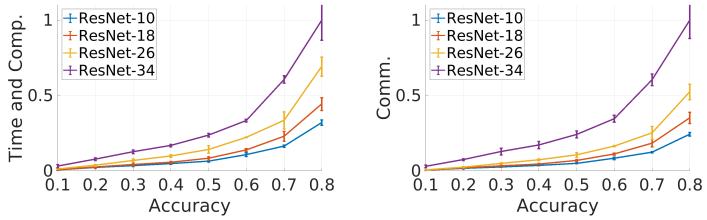
**Computation.** Fig. 2(b) shows the computation overhead. We make the following observations: (1) More participants result in worse computation overhead. The results indicate that the gain of faster model convergence from more participants does not compensate for more participants' much higher computation costs. (2) The communication overhead is increased when a larger number of training passes is used. This is probably because that larger $E$ diverges the model training (Li et al., 2020a) and thus, the data utility per unit of computation cost is reduced.

**Communication.** Fig. 2(c) plots the communication overhead. As we can see, more participants greatly increase the communication overheads. This is because more participants can only weakly reduce the number of training rounds $R$ (Li et al., 2020b), however, on each round the number of




(a) Time Overhead (b) Computation Overhead (c) Communication Overhead

Figure 2: Time, computation, and communication overheads when a different number of participants and a different number of training passes are used. The lower the better.




(a) Time and computation overheads (b) Communication overhead

Figure 3: Time, computation, and communication overheads versus model complexity. The lower the better.

transmissions is linearly increased with the number of participants. Regarding the number of training passes, larger $E$ reduces the total number of training rounds $R$ and thus better communication efficiency. On the other hand, the gain of larger $E$ is diminishing. The results are consistent with the analysis of Li et al. (2020b) that $R$ is hyperbolic with $E$ (the turning point happens around 100-1000 in their experiments).

**Model Complexity.** Table 1 tabulates the models for comparing training overheads versus model complexity. In this experiment, we select one participant ($M = 1$) to train one pass ($E = 1$) on each training round. The results are expected to be applicable to other numbers of participants and training passes. Fig. 3 shows the normalized time, computation, and communication overheads for different models. The x-axis is the target model accuracy, and the y-axis is the corresponding overheads to reach that model accuracy. Since only one client is selected on each round, the time and computation overheads have the same normalized performance. The results show that smaller models are better with regards to time (Fig. 3(a)), computation (Fig. 3(a)) and communication (Fig. 3(b)). Therefore, it is preferred to select a smaller model as long as the model accuracy satisfies the accuracy requirement. In addition, it is interesting to note that heavier models have higher increase rate of overheads versus model accuracy. This means that model selection is especially essential for applications that require high model accuracy. In our ongoing work, we iteratively increase model complexities by using transfer learning, in order to achieve better FL training efficiencies and reach any target model accuracy.

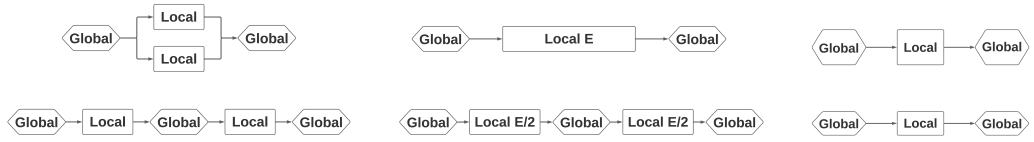
### 4.2 Heuristic Explanation of Measurement Results

Table 2 summarizes our measurement observations. As we can see, time, computation, and communication performance conflict with each other in terms of the number of participants $M$ and the number of training passes $E$. Regarding model complexity, smaller models have better time, computation, and communication efficiencies if the model accuracy is satisfied. We provide heuristics on the measurement results of $M$, $E$, and model complexity. Fig. 4 visualizes our heuristics.

- *The number of participants* $M$. In Fig. 4(a), the top scheme and the bottom scheme have the same computation cost (two local training) and communication cost (four transmis-

| Training aspect | Number of participants | Number of training passes | Model complexity |
|---|---|---|---|
| Time | $>$ | $<$ | $<$ |
| Computation | $<$ | $<$ | $<$ |
| Communication | $<$ | $>$ | $<$ |
| Model Accuracy | $=$ | $=$ | $>$ |

Table 2: Summary of measurement results. '$<$', '$=$', and '$>$' means the smaller the better, does not matter, and the larger the better, respectively.




(a) Number of participants $M$ (b) Number of training passes $E$ (c) Model complexity

Figure 4: Heuristic explanation of our measurement results. (a) The number of participants. The bottom scheme is better regarding computation and communication efficiency. (b) The number of training passes. The bottom scheme is better regarding time and computation efficiency. (c) Model complexity. The bottom scheme is better for time, computation, and communication efficiency.

sions). However, the bottom scheme has a better overall computation and communication efficiency. The bottom scheme is better probably because the clients in the bottom scheme always work on the updated global model, whereas the clients in the top scheme work on the same global model. In other words, narrow-and-deep FL schemes have better computation and communication performance than wide-and-shallow FL schemes.

- *The number of training passes* $E$. In Fig. 4(b), both the top scheme and the bottom scheme take $E$ total training passes. However, the bottom scheme has better time and computation efficiency at the expense of higher communication costs. The results indicate that the usefulness per local update decreases with the number of local updates. Therefore, for time and/or computation-sensitive FL applications, large $E$ should be avoided.
- *Model complexity*. Fig. 4(c) shows that a smaller model has better time, computation, and communication efficiencies than a heavier model, as long as the accuracy requirement is met. In addition, our results in Fig. 3 indicate that if the target model accuracy is low, then using a heavy model does not introduce significantly more overheads than a lightweight model. However, if the goal is to achieve a high-accuracy model, carefully selecting a model complexity is essential, as over-large models cause significantly more training overheads of time, computation, and communication.

## 5 FedTuning: Automatic Tuning of FL Hyper-Parameters

We propose FedTuning for automatic tuninig of FL hyper-parameters. FedTuning considers training preferences for time, computation, and communication, denoted by $\alpha$, $\beta$, and $\lambda$, respectively. Without losing generality, we can adopt $\alpha + \beta + \lambda = 1$. For example, $\alpha = 0.7$, $\beta = 0.2$, and $\lambda = 0.1$ represent that the application is greatly concerned about the training time, while slightly about the computation overhead, with the communication overhead the least concern. For two sets of FL hyper-parameters $S_1$ and $S_2$, FedTuning defines the comparison function $I(S_1, S_2)$ as

$$I(S_1, S_2) = \alpha \times \frac{t_2 - t_1}{t_1} + \beta \times \frac{q_2 - q_1}{q_1} + \lambda \times \frac{z_2 - z_1}{z_1} \tag{5}$$

where $t_1$ and $t_2$ are time overheads for $S_1$ and $S_2$ achieving the same model accuracy. Correspondingly, $q_1$ and $q_2$ are the computation overheads, and $z_1$ and $z_2$ are the communication overheads. If $I(S_1, S_2) < 0$, then $S_2$ is better than $S_1$. A set of hyper-parameters is better than another set if the weighted improvement of some training aspects (e.g., time) is higher than the weighted degradation of the remaining training aspects (e.g., computation and communication). The weights are training preferences on time, computation, and communication.

**Algorithm 1:** FedTuning: Auto tuning of federated learning hyper-parameters.

**Input:**
$\alpha$, $\beta$, $\lambda$: training preference for time, computation, and communication
$\epsilon$: minimum improvement of model accuracy for decision making

**Begin**
$S_{prv}$, $S_{cur}$: previous and current sets of hyper-parameters
$a_{prv}$, $a_{cur}$: previous and current model accuracy
$t_{prv}$, $q_{prv}$, $z_{prv}$: time, computation, and communication overheads under $S_{prv}$
$t_{cur}$, $q_{cur}$, $z_{cur}$: time, computation, and communication overheads under $S_{cur}$

**for** *training round* $r = 1, 2, ...$ **do**
  Train FL model for one round as usual
  Update $a_{cur}, t_{cur}, q_{cur}, z_{cur}$
  **if** $a_{cur} - a_{prv} > \epsilon$ **then**
    $t_{cur} = t_{cur}/(a_{cur} - a_{prv})$ // Normalize time overhead
    $q_{cur} = q_{cur}/(a_{cur} - a_{prv})$ // Normalize computation overhead
    $z_{cur} = z_{cur}/(a_{cur} - a_{prv})$ // Normalize communication overhead
    // Calculate $\Delta M$
    $\Delta M = \frac{(+1)\times\alpha\times|t_{cur}-t_{prv}|}{t_{cur}} + \frac{(-1)\times\beta\times|q_{cur}-q_{prv}|}{q_{cur}} + \frac{(-1)\times\alpha\times|z_{cur}-z_{prv}|}{z_{cur}}$
    // Calculate $\Delta E$
    $\Delta E = \frac{(-1)\times\alpha\times|t_{cur}-t_{prv}|}{t_{cur}} + \frac{(-1)\times\beta\times|q_{cur}-q_{prv}|}{q_{cur}} + \frac{(+1)\times\lambda\times|z_{cur}-z_{prv}|}{z_{cur}}$
    **if** $\Delta M > 0$ **then**
      $M_{nxt} = M_{cur} + 1$
    **else**
      $M_{nxt} = M_{cur} - 1$
    **end**
    **if** $\Delta E > 0$ **then**
      $E_{nxt} = E_{cur} + 1$
    **else**
      $E_{nxt} = E_{cur} - 1$
    **end**
    $S_{nxt} = \{M_{nxt}, E_{nxt}\}$
    $a_{prv} = a_{cur}, t_{prv} = t_{cur}, q_{prv} = q_{cur}, z_{prv} = z_{cur}, S_{prv} = S_{cur}, S_{cur} = S_{nxt}$
    Change FL hyper-parameters according to $S_{nxt}$
  **end**
**end**

However, the training overheads for different sets of FL hyper-parameters are unknown *a priori*. As a result, directly identifying the optimal hyper-parameters before FL training is impossible. Instead, we propose an iterative method to optimize the next set of hyper-parameters. Given the current set of hyper-parameters $S_{cur}$, the goal is to find a set of hyper-parameters $S_{nxt}$ that improves the training performance the most, that is, minimizing the following objective function:

$$G(S_{nxt}) = \alpha \times \frac{t_{nxt} - t_{cur}}{t_{cur}} + \beta \times \frac{q_{nxt} - q_{cur}}{q_{cur}} + \lambda \times \frac{z_{nxt} - z_{cur}}{z_{cur}} \tag{6}$$

where $t_{cur}$, $q_{cur}$, and $z_{cur}$ are time, computation, and communication overhead under the current hyper-parameters $S_{cur}$; $t_{nxt}$, $q_{nxt}$, and $z_{nxt}$ are the time, computation, and communication overheads for the next hyper-parameters $S_{nxt}$. We focus on the number of participants $M$ and the number of training passes $E$, since model complexity is monotonous with training overheads. We need to optimize $S_{nxt} = \{M, E\}$. To find the optimal $S_{nxt}$, we take the derivatives of $G(S_{nxt})$ over $M$ and $E$, obtaining

$$\Delta M = \frac{\partial G(S_{nxt})}{\partial M} = \frac{\alpha}{t_{cur}} \times \frac{\partial t_{nxt}}{\partial M} + \frac{\beta}{q_{cur}} \times \frac{\partial q_{nxt}}{\partial M} + \frac{\lambda}{z_{cur}} \times \frac{\partial z_{nxt}}{\partial M} \tag{7}$$

$$\Delta E = \frac{\partial G(S_{nxt})}{\partial E} = \frac{\alpha}{t_{cur}} \times \frac{\partial t_{nxt}}{\partial E} + \frac{\beta}{q_{cur}} \times \frac{\partial q_{nxt}}{\partial E} + \frac{\lambda}{z_{cur}} \times \frac{\partial z_{nxt}}{\partial E} \tag{8}$$

We illustrate how to obtain $\Delta M$. The process of solving $\Delta E$ is similar. Considering that we make a small adjustment of hyper-parameters in each step, we can approximate $\partial t_{nxt}/\partial M$ as $(+1)\times|t_{cur} - t_{prv}|$ where $t_{prv}$ is the time overhead of the previous hyper-parameter set $S_{prv}$, and $(+1)$ means the time efficiency prefers larger $M$ from Table 2. Similarly, we have $\partial q_{nxt}/\partial M = (-1)\times|q_{cur} - q_{prv}|$, and $\partial z_{nxt}/\partial M = (-1) \times |z_{cur} - z_{prv}|$. As a result, $\Delta M$ can be approximated as

$$\Delta M = \frac{(+1) \times \alpha \times |t_{cur} - t_{prv}|}{t_{cur}} + \frac{(-1) \times \beta \times |q_{cur} - q_{prv}|}{q_{cur}} + \frac{(-1) \times \alpha \times |z_{cur} - z_{prv}|}{z_{cur}} \tag{9}$$

| $\alpha$ | $\beta$ | $\lambda$ | Time ($\times 10^3$) | Comp ($\times 10^3$) | Comm | Final $M$ | Final $E$ | Overall Performance |
|---|---|---|---|---|---|---|---|---|
| - | - | - | 181.52 | 1089.98 | 1380 | 20 | 20 | - |
| 1 | 0 | 0 | 48.02 (+73.55%) | 429.21 (+60.55%) | 3942 (-185.65%) | 44 | 1 | +73.55% |
| 0.8 | 0.1 | 0.1 | 54.42 (+70.02%) | 434.65 (+60.12%) | 2443 (-77.03%) | 39 | 1 | +54.33% |
| 0 | 1 | 0 | 85.93 (+52.66%) | 298.27 (+72.64%) | 1015 (+26.45%) | 1 | 1 | +72.64% |
| 0.1 | 0.8 | 0.1 | 67.84 (+62.63%) | 235.24 (+78.42%) | 1384 (-0.29%) | 1 | 1 | +68.97% |
| 0 | 0 | 1 | 896.85 (-394.08%) | 1242.21 (-13.97%) | 902 (+34.64%) | 1 | 44 | +34.64% |
| 0.1 | 0.1 | 0.8 | 879.87 (-384.72%) | 1376.91 (-26.32%) | 1048 (+24.06%) | 1 | 39 | -21.63% |
| 0 | 0.5 | 0.5 | 314.93 (-73.50%) | 564.25 (+48.23%) | 969 (+29.78%) | 1 | 15 | +39.01% |
| 0.1 | 0.45 | 0.45 | 317.51 (-74.92%) | 562.98 (+48.35%) | 960 (+30.43%) | 1 | 11 | +27.67% |
| 0.5 | 0 | 0.5 | 185.27 (-2.07%) | 1111.57 (-1.98%) | 1371 (+0.65%) | 18 | 22 | -0.71% |
| 0.45 | 0.1 | 0.45 | 179.33 (+1.21%) | 463.69 (+57.46%) | 1022 (+25.94%) | 3 | 7 | +17.96% |
| 0.5 | 0.5 | 0 | 56.36 (+68.95%) | 388.98 (+64.31%) | 2580 (-86.96%) | 32 | 1 | +66.63% |
| 0.45 | 0.45 | 0.1 | 56.47 (+68.89%) | 335.23 (+69.24%) | 1662 (-20.43%) | 19 | 1 | +60.12% |
| 0.33 | 0.33 | 0.33 | 64.22 (+64.62%) | 303.17 (+71.59%) | 1592 (-15.36%) | 5 | 1 | +40.28% |

Table 3: Performance of FedTuning for different training preferences $\alpha$, $\beta$, and $\lambda$. The initial $\{M, E\}$ are $\{20, 20\}$. '$+$' is improvement and '$-$' is degradation.

Similarly, we can calculate $\Delta E$ as

$$\Delta E = \frac{(-1) \times \alpha \times |t_{cur} - t_{prv}|}{t_{cur}} + \frac{(-1) \times \beta \times |q_{cur} - q_{prv}|}{q_{cur}} + \frac{(+1) \times \lambda \times |z_{cur} - z_{prv}|}{z_{cur}} \quad (10)$$

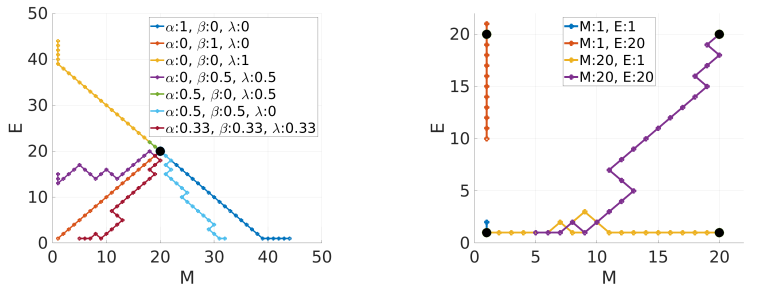
In each step of hyper-parameter optimization, FedTuning increases or decreases $M$ and $E$ by one based on the signs of $\Delta M$ and $\Delta E$. Algorithm 1 details the decision process in FedTuning. After each training round, FedTuning first checks if the current model accuracy is higher than the model accuracy under its previous hyper-parameters $S_{prv}$ by at least $\epsilon$ (Line 14). If so, FedTuning normalizes current overheads (Line 15-17) and calculates $\Delta M$ and $\Delta E$ (Line 18-19). If $\Delta M$ is positive, FedTuning increases $M$ by one; else it decreases $M$ by one (Line 20-25). The optimization process for $E$ is similar (Line 25-29). The FL training is continued with the new hyper-parameters $M_{nxt}$ and $E_{nxt}$. FedTuning is very lightweight, and thus, the overhead of its decision-making process is negligible to the FL training.

## 6 EVALUATION

We use ResNet-10 in the evaluation. We set the target accuracy to 0.8 for the speech-to-command dataset. In the experiments, we set $C_1$, $C_2$, and $C_3$ to 1, as they do not affect the comparison.

Table 3 shows the time, computation, and communication overheads when the initial $M$ and $E$ are set to 20. The first row is the baseline, which does not change hyper-parameters during the FL training. We show the final $M$ and $E$ when the training is finished. The overall performance is calculated using Eq. (5). As we can see from Table 3, FedTuning can adapt to different training preferences. When $\alpha = 0.5, \beta = 0, \lambda = 0.5$, however, FedTuning does not improve performance. This is because time and communication have the opposite performance against $M$ and $E$. Therefore, when their weights ($\alpha$ and $\gamma$) are the same, FedTuning is stuck with the initial hyper-parameters. If we change $\alpha$, $\beta$, and $\gamma$ from 0.5, 0, 0.5 to 0.45, 0.1, 0.45, we can improve the overall performance by 17.96%. In addition, we found that setting a small number (e.g., 0.1) to the unconcerned training aspects result in a similar performance for the target training aspects while greatly reducing the overhead of the unconcerned system aspects. Therefore, we recommend setting a small value instead of 0 to unconcerned training aspects, so all training aspects are included in the optimization process. We observe a degraded performance for $\alpha = 0.1, \beta = 0.1, \lambda = 0.8$, which needs further exploration. Table 3 shows that FedTuning achieves an average of $41.04\%$ improvement compared to the baseline.

Fig. 5(a) illustrates the trajectory of hyper-parameters in FedTuning when the training preferences are different. The initial $M$ and $E$ are 20. As we can see, FedTuning can make hyper-parameters to favor different application preferences. Fig. 5(a) also shows that FedTuning tends to move towards the edge of either $M$ or $E$. We also investigate FedTuning's decisions when the initial sets of hyper-parameters are different. In this experiment, we fix the training preference to be the same for time, computation, and communication (i.e., $\alpha = \beta = \lambda = 0.33$), and we change the initial $\{M, E\}$ to be $\{1, 1\}$, $\{1, 20\}$, $\{20, 1\}$, and $\{20, 20\}$. Fig. 5(b) illustrates the trajectories of these four cases. The




(a) Different application preferences (b) Different starting points $\{M, E\}$

Figure 5: Decisions of FedTuning (a) Trajectories for different application preferences. The initial $M$ and $E$ are 20. (b) Trajectories for different initial $M$ and $E$, where the time, computation, and communication preferences are the same ($\alpha = \beta = \lambda = 0.33$).

case of the initial $\{1, 20\}$ stops at $\{1, 10\}$, while the $\{20, 1\}$ case stops at $\{5, 1\}$. The initial hyper-parameters influence the overall system performance because different hyper-parameters are used during the training process. Nonetheless, FedTuning is robust against the initial hyper-parameters, as it moves toward the same area (bottom-left corner for the training preferences).

## 7 Discussion

FedTuning has promising performance in tuning FL hyper-parameters. As one of the first work of its kind, FedTuning has some limitations/opportunities that deserve further exploration.

*Heterogeneous Devices*. This paper assumes that clients are equipped with the same hardware, i.e., the same computation speed and the same transmission rate. In practice, however, client devices are heterogeneous. Measurements of the computation capabilities of mobile devices and their network throughput exhibit order-of-magnitude difference (Ignatov, 2021; M-Lab, 2021; Lai et al., 2021a). As a result, for clients even with the same amount of local data points, they cause different computation costs and transmission delays. When clients' transmission delays are not constant and thus not negligible, we should include the transmission delay (i.e., $C'$ in Eq. (2)) in the time-overhead measurement. We leave it as future work to evaluate FedTuning with heterogeneous client devices.

*Other FL Algorithms*. Many FL algorithms have been proposed to tackle the limitations of FedAvg. We plan to extend FedTuning to the following scenarios. (1) Participant selection. Compared to the random selection of participants, guided participant selection that considers clients' data utility and device utility can improve overall training performance (Lai et al., 2021b). Popular alternatives are to only wait for participants that are finished before a deadline (Bonawitz et al., 2019) or only wait for the first $M$ participants (Li et al., 2020b). (2) Adaptive training passes across participants. Due to the heterogeneity of clients, setting the same number of training passes $E$ for all participants on each training round is not optimal. To support different $E$ across participants, FedNova (Wang et al., 2020b) relies on re-weighting of aggregation while FedProx (Li et al., 2020a) adds a proximal term to stabilize the convergence. (3) Aggregation methods. In addition to FedAvg, many aggregation methods are available, such as FedProx (Li et al., 2020a), FedIR (Hsu et al., 2020), and FedMA (Wang et al., 2020a).

## 8 Conclusion

Federated learning is gaining popularity for a variety of privacy-preserving applications. However, different FL applications have diverse training preferences, which puts burden on FL practitioners to select optimal FL hyper-parameters. In this paper, we propose FedTuning to automatically adjust FL hyper-parameters tailoring for application's training preferences on time, computation, and communication. Our evaluation results show that FedTuning is flexible and adaptive, achieving an average of 41% improvement compared to the baseline.

**Ethics Statement** This paper targets hyper-parameter tuning for federated learning applications, which is beneficial to society's endeavor to preserve data privacy. We are honest in conducting our experiments, and we do our best to reduce ambiguity in our writing. We will release our code for public review.

**Reproducibility Statement** We implement standard FedAvg in PyTorch. We carefully explain our experiments settings to be as straightforward to readers as possible. We believe our results can be readily reproduced. Our code will be made public.

## REFERENCES

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards Federated Learning at Scale: System Design. In *Conference on Machine Learning and Systems (MLSys)*, pp. 374–388, 2019.

Maria Moitinho de Almeida and Johan von Schreeb. A Smartphone-Enabled Fall Detection Framework for Elderly People in Connected Home Healthcare. *Prehospital and Disaster Medicine*, 34: 82–88, 2018.

Saad Hikmat Haji and Siddeeq Y. Ameen. Attack and Anomaly Detection in IoT Networks using Machine Learning Techniques: A Review. *Asian Journal of Research in Computer Science (AJRCOS)*, 9(2):30–46, 2021.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated Learning for Mobile Keyboard Prediction. *arXiv:1811.03604*, 2019.

Mohammad Mehedi Hassan, Abdu Gumaei, Gianluca Aloi, Giancarlo Fortino, and Mengchu Zhou. A Smartphone-Enabled Fall Detection Framework for Elderly People in Connected Home Healthcare. *IEEE Access*, 33:58–63, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Tzu Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated Visual Classification with Real-World Data Distribution. In *European Conference no Computer Vision (ECCV)*, pp. 76–92, 2020.

Andrey Ignatov. AI-Benchmark. `https://ai-benchmark.com/index.html`, 2021.

M. I. Jordan and T. M. Mitchell. Machine Learning: Trends, Perspectives, and Prospects. *Science*, 349(6245):255–260, 2015.

Ce Ju, Ruihui Zhao, Jichao Sun, Xiguang Wei, Xinwei Zhang, Dashan Gao, Ben Tan, Han Yu, Chuning He, and Yuan Jin. Privacy-Preserving Technology to Help Millions of People: Federated Prediction Model for Stroke Prevention. *arXiv:2006.10517*, 2020.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost Optimal Exploration in Multi-Armed Bandits. In *International Conference on Machine Learning (ICML)*, pp. 1238–1246, 2013.

Fan Lai, Yinwei Dai, Xiangfeng Zhu, and Mosharaf Chowdhury. FedScale: Benchmarking Model and System Performance of Federated Learning. *arXiv: 2105.11367*, pp. 1–15, 2021a.

Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient Federated Learning via Guided Participant Selection. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 19–35, 2021b.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research (JMLR)*, 18(1):6765–6816, 2017.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated Optimization in Heterogeneous Networks. In *Conference on Machine Learning and Systems (MLSys)*, pp. 429–450, 2020a.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the Convergence of FedAvg on Non-IID Data. In *International Conference on Learning Representations (ICLR)*, pp. 1–12, 2020b.

M-Lab. MobiPerf. `https://www.measurementlab.net/tests/mobiperf`, 2021.

H. Brendan McMahan, Daniel Ramage Eider Moore, Seth Hampson, and Blaise Agueray Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1–10, 2017.

Dagmawi Neway Mekuria, Paolo Sernani, Nicola Falcionelli, and Aldo Franco Dragoni. Smart Home Reasoning Systems: A Systematic Literature Review. *Journal of Ambient Intelligence and Humanized Computing*, 12:4485–4502, 2021.

Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, Sudeep Agarwal, Julien Freudiger, Andrew Byde, Abhishek Bhowmick, Gaurav Kapoor, Si Beaumont, Aine Cahill, Dominic Hughes, Omid Javidbakht, Fei Dong, Rehan Rishi, and Stanley Hung. Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications. *arXiv:2102.08503*, 2021.

Abhinav Sharma, Arpit Jain, Prateek Gupta, and Vinay Chowdary. Machine Learning Applications for Precision Agriculture: A Comprehensive Review. *IEEE Access*, 9:4843–4873, 2020.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *International Conference on Neural Information Processing Systems (NIPS)*, pp. 2951–2959, 2012.

Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A Survey on Distributed Machine Learning. *ACM Computing Surveys*, 53(2):1–33, 2020.

Hongyi Wang, Mkkhail Yurochkin, Sun Yuekai, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated Learning with Matched Averaging. In *International Conference on Learning Representations (ICLR)*, pp. 1–16, 2020a.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Conference on Neural Information Procesing Systems (NeurIPS)*, pp. 1–13, 2020b.

Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Aguera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horvath, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konecny, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtarik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A Field Guide to Federated Optimization. *arXiv: 2107.06917*, pp. 1–88, 2021.

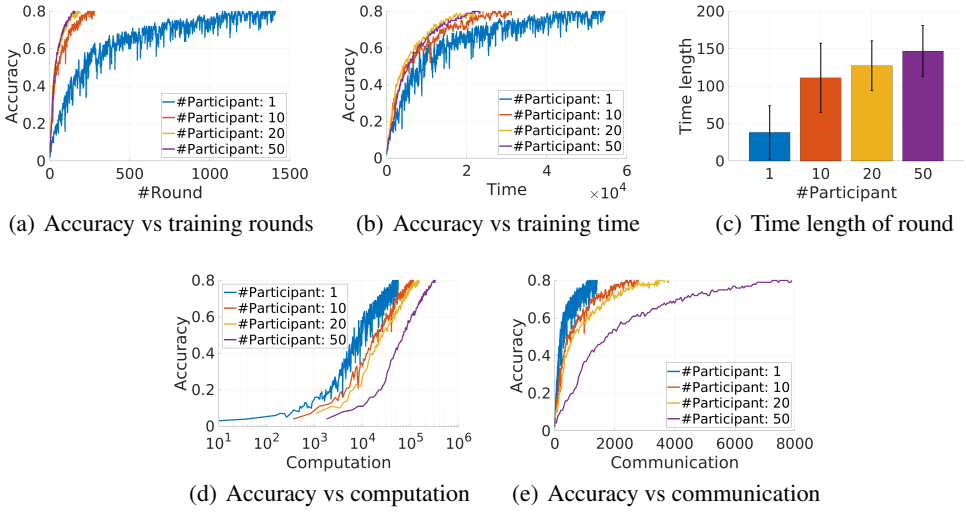
Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv: 1804.03209*, 2018.

Myounggyu Won. Intelligent Traffic Monitoring Systems for Vehicle Classification: A Survey. *IEEE Access*, 8:73340–73358, 2020.

Li Yang and Abdallah Shami. On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing*, 415:295–316, 2020.

# A APPENDIX

## A.1 ILLUSTRATION OF DIFFERENT TRAINING ASPECTS




(a) Accuracy vs training rounds (b) Accuracy vs training time (c) Time length of round

(d) Accuracy vs computation (e) Accuracy vs communication

Figure 6: Illustration of different training aspects.

Fig. 6 illustrates FL training when a different number of participants $M$ is used on each round. In this experiment, the number of training passes $E$, $C_1$, $C_2$ and $C_3$ are all set to 1.

- Fig. 6(a): more participants have better round-to-accuracy performance, which is consistent with the common knowledge.
- Fig. 6(b): regarding time-to-accuracy, more participants still are better, but the performance gain of more participants is reduced. This is because when more participants are selected, the time length of each training round is also enlarged, as shown in Fig. 6(c).
- With regard to the computation-to-accuracy, however, fewer participants are better, as shown in Fig. 6(d).
- Fewer participants also have better communication-to-accuracy performance, as illustrated in Fig. 6(e).