Federated Learning Hyper-Parameter Tuning From A System Perspective

Huanle Zhang, Lei Fu, Mi Zhang, Pengfei Hu, Xiuzhen Cheng, *Fellow, IEEE*, Prasant Mohapatra, *Fellow, IEEE*, and Xin Liu, *Fellow, IEEE*,

***Abstract*—Federated learning (FL) is a distributed model training paradigm that preserves clients' data privacy. It has gained tremendous attention from both academia and industry. FL hyper-parameters (e.g., the number of selected clients and the number of training passes) significantly affect the training overhead in terms of computation time, transmission time, computation load, and transmission load. However, the current practice of manually selecting FL hyper-parameters imposes a heavy burden on FL practitioners because applications have different training preferences. In this paper, we propose FedTune, an automatic FL hyper-parameter tuning algorithm tailored to applications' diverse system requirements in FL training. FedTune iteratively adjusts FL hyper-parameters during FL training and can be easily integrated into existing FL systems. Through extensive evaluations of FedTune for diverse applications and FL aggregation algorithms, we show that FedTune is lightweight and effective, achieving 8.48%-26.75% system overhead reduction compared to using fixed FL hyper-parameters. This paper assists FL practitioners in designing high-performance FL training solutions. The source code of FedTune is available at https://github.com/DataSysTech/FedTune.**

***Index Terms*—Federated Learning, Hyper-parameter Tuning, System Overhead, Training Preference, Optimization**

I. Introduction

FEDERATED Learning (FL) has been applied to a wide range of applications such as mobile keyboard [1], speech recognition [2], and human stroke prevention [3] on top of mobile devices and Internet of Things (IoT) [4], [5]. Compared to other model training paradigms (e.g., centralized machine learning [6], conventional distributed machine learning [7]), FL has unique properties with regards to (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 may not represent the overall distribution [8]. 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 methods and participant selection [9], [10]. Nonetheless, many FL algorithms, e.g., FedAvg [8], have been proved to converge to the global optimum even different FL hyper-parameters are adopted [11] [12].

Huanle Zhang, Pengfei Hu, and Xiuzhen Cheng are with the School of Computer Science and Technology, Shandong University, China. E-mail: {dtczhang, phu, xzcheng}@sdu.edu.cn; Lei Fu is with the Bank of Jiangsu and Fudan University, China. E-mail: leileifu@163.sufe.edu; Mi Zhang is with the Department of Computer Science and Engineering, Ohio State University, USA. E-mail: mizhang.1@osu.edu; Prasant Mohapatra and Xin Liu are with the Department of Computer Science, University of California, Davis, USA. E-mail: {pmohapatra,xinliu}@ucdavis.edu.

Pengfei Hu is the corresponding author.

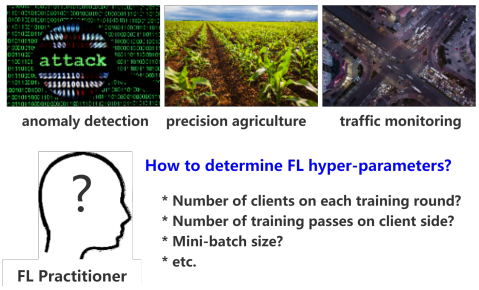


Fig. 1. FL practitioners have various constraints in determining FL hyper-parameters for different applications.

Although FL hyper-parameters do not affect FL convergence (i.e., the same final global model accuracy), they significantly affect the training overheads of reaching the final model. Specifically, Computation Time (CompT), Transmission Time (TransT), Computation Load (CompL), and Transmission Load (TransL) are the four most crucial system overhead:

- *Computation Time (CompT)*. It measures how long an FL system spends in model training. Overall model training time must be short for applications that require fast adaptation to environments (e.g., security problems).
- *Transmission Time (TransT)*. It represents how long an FL system spends in model parameter transmission between clients and servers. Without a high-speed transmission solution, transmission time can overwhelm the overall FL training since models are generally large and multiple rounds of model parameter transmissions are required.
- *Computation Load (CompL)*. It is the number of Floating-Point Operation (FLOP) that an FL system consumes. For low-profile devices such as IoT nodes, the computation load must be small as these devices are equipped with low computational resources.
- *Transmission Load (TransL)*. It is the total data size transmitted between the clients and the server. This is critical when data transmission is expensive or its power consumption is a concern. For example, outdoor applications usually rely on cellular communications, which need to pay a considerable price for transmitting a large amount of data.

Fig. 1 summarizes the different application scenarios faced by FL practitioners in determining FL hyper-parameters. Application scenarios have different training preferences in terms of CompT, TransT, CompL, and TransL. Consider the

following examples: (1) attack and anomaly detection in computer networks [13] is time-sensitive (CompT and TransT) as it needs to adapt to malicious traffic rapidly; (2) smart home control systems for indoor environment automation [14], e.g., heating, ventilation, and air conditioning (HVAC), are sensitive to computation (CompT and CompL) because sensor devices are limited in computation capabilities; (3) traffic monitoring systems for vehicles [15] are communication-sensitive (TransT and TransL) because cellular communications are usually adopted to provide city-scale connectivity; (4) precision agriculture based on IoT sensing [16] is not time-urgent but requires energy-efficient solutions (CompL and TransL); (5) healthcare systems, e.g., fall detection for elderly people [17], require both fast response and small energy consumption (CompT, TransT, CompL, and TransL); and (6) human stampedes detection/prevention [18] require time, computation, and communication efficient systems.

A few approaches have studied FL training performance under different hyper-parameters [19]. However, they do not consider CompT, TransT, CompL, and TransL together, which is essential from a system's perspective. Besides, they almost all use the number of training rounds for comparison, i.e., round-to-accuracy [12], [20] and convergence rate [8], [11]. The corresponding observations cannot be directly applied to guide FL system design in our context. This is because clients in FL are heterogeneous in terms of their different amounts of local training data (i.e., *unbalanced* data), computation speeds, and transmission rates, and thus each training round has a different time length, computation cost, and transmission cost. For example, selecting more clients in each training round results in a better round-to-accuracy performance [8], [11]. But it does not necessarily mean a better time-efficiency as the time length of each training round increases with the number of selected clients, nor a better transmission-efficiency as more clients need to transmit model parameters in each training round. 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 CompT and TransL-efficient.

This paper targets a new research problem of optimizing the hyper-parameters for FL from a system perspective. To do so, we formulate the system overheads of FL training and conduct extensive measurements to understand FL training performance. Our measurement results illustrate how to tune hyper-parameters for simple application scenarios and are the basis of our tuning algorithm. To avoid manual hyper-parameter selection, we propose FedTune, an algorithm that automatically tunes FL hyper-parameters during model training, respecting application training preferences. In summary, we make the following contributions:

1) To the best of our knowledge, FedTune is the first FL hyper-parameter tuning algorithm that jointly considers applications' training preferences for CompT, TransT, CompL, and CompL. We conduct measurements to understand system training overhead and design an automatic tuning algorithm based on our measurement results.
2) FedTune is a general framework that applies to various FL hyper-parameters. In this paper, we take the number of participants, the number of training passes, and the model complexity as examples to illustrate FL hyper-parameter tuning.
3) We extensively evaluate FedTune by various applications/datasets (e.g., command detection, handwriting recognition) and aggregation methods (e.g., FedNova and FedAdagrad). FedTune reduces an average of 8.48%-26.75% system overhead compared to using fixed FL hyper-parameters for different applications and aggregation methods. In addition, FedTune reduces some application scenarios by up to 70.51%, showing that FedTune greatly benefits application scenarios that common hyper-parameters perform poorly.
4) We release our code at https://github.com/DataSysTech/FedTune to facilitate the research of FL hyper-parameter tuning.

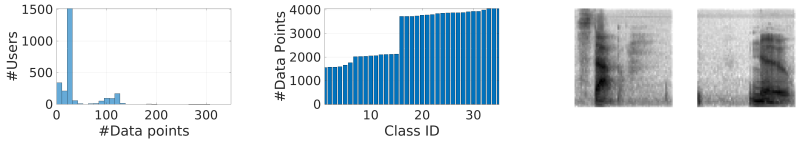
TABLE I
RELATED WORK ON FL HYPER-PARAMETER OPTIMIZATION. WE TAG IF (1) THE WORK CAN RUN IN AN ONLINE AND SINGLE TRAIL MANNER AND (2) THE WORK TARGETS SYSTEM OVERHEADS OF FL TRAINING.

Work	Description	Single trial	System
FTS [22]	optimize client models	✗	✗
Zhiyuan et al. [23]	PSO-based optimization	✗	✗
DP-FTS-DE [24]	trade-off privacy and utility	✗	✗
Auto-FedRL [25]	improve model accuracy	✓	✗
[26]	improve training robustness	✓	✗
FedEx [27]	NAS based framework	✗	✗
FLoRA [28]	NAS based framework	✓	✗
FedTune (Ours)	a lightweight framework	✓	✓

A preliminary version of this work was published in MILCOM 2022 [21]. In comparison, this paper makes the following extensions. (1) We conduct more experiments to illustrate the FL training, which helps readers to understand the nature of FL training when different hyper-parameters are used. For example, Fig. 3 shows the properties of FL training from different aspects. (2) We highlight the measurement results and provide intuitive explanations to aid FL practitioners in determining hyper-parameters when only a single system overhead is concerned. The results are depicted in Fig. 6, which guide FL practitioners to select hyper-parameters. (3) We conduct significantly more experiments to evaluate FedTune, e.g., the study of the penalty mechanism (Fig. 8 and Fig. 9). (4) We thoroughly revise the whole paper to make it more readable and concise, and extend our 6-page conference paper to this 12-page extension version.

II. RELATED WORK

Hyper-Parameter Optimization (HPO) is a field that has been extensively studied [29]. Many classical HPO algorithms, e.g., Bayesian Optimization (BO) [30], successive halving [31], and hyperband [32], are designed to optimize hyper-parameters of machine learning models. However, they cannot be directly applied to FL because FL has unique hyper-parameters such as the client-side and server-side aggregation



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

Fig. 2. Google speech-to-command dataset used in measurements.

TABLE II
NOTATIONS USED IN THIS PAPER.

Notation	Meaning
K	the total number of clients
M	the number of participants in each training round
E	the number of training passes over its local data
R	the number of training rounds to reach final model
$b_{k,r}$	whether client k participates in training round r
t, q, z, v	system overheads of computation time, transmission time, computation load, and transmission load
α, β, γ, δ	preference for omputation time, transmission time, computation load, and transmission load

TABLE III
DIFFERENT MODELS USED FOR THE MEASUREMENT STUDY.

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

methods, and FL is a different training paradigm from centralized machine learning and conventional distributed machine learning.

Designing HPO methods for FL is a new research area. Only a few approaches have touched FL HPO problems. Table I lists several representative approaches, where we highlight whether the work can execute in an online and single trial manner and whether the work targets system overhead in FL training. For example, BO has been integrated with FL to improve different client models [22] and strength client privacy [24]; Zhiyuan *et al.* apply Particle Swarm Optimization (PSO) to accelerate the search speed of FL hyper-parameters [23], but without the support for the single-trial and system overhead. Several approaches apply reinforcement learning to adjust FL hyper-parameters [25], [26], which introduces extra complexity and loss of generality. FedEx is a general framework to optimize the round-to-accuracy of FL by exploiting the Neural Architecture Search (NAS) techniques of weight-sharing, which improves the baseline by several percentage points [27]; FLoRA determines the global hyper-parameters by selecting the hyper-parameters that have good performances in local clients [28]. Recently, a benchmark suite for federated hyper-parameter optimization [33] is designed, whose effectiveness remains to be investigated.

For two reasons, existing approaches cannot be directly applied to our scenario of optimizing FL hyper-parameter for different FL training preferences. First, CompT (in seconds), TransL (in seconds), CompL (in FLOPs), and TransL (in bytes) are not comparable with each other. Incorporating training preferences in HPO is not trivial. Second, 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 overhead.

III. UNDERSTANDING THE PROBLEM

We first quantify the system overhead of FedAvg to illustrate the problem. Table II tabulates the notations that are used in this paper for quick reference. FedAvg minimizes the following objective

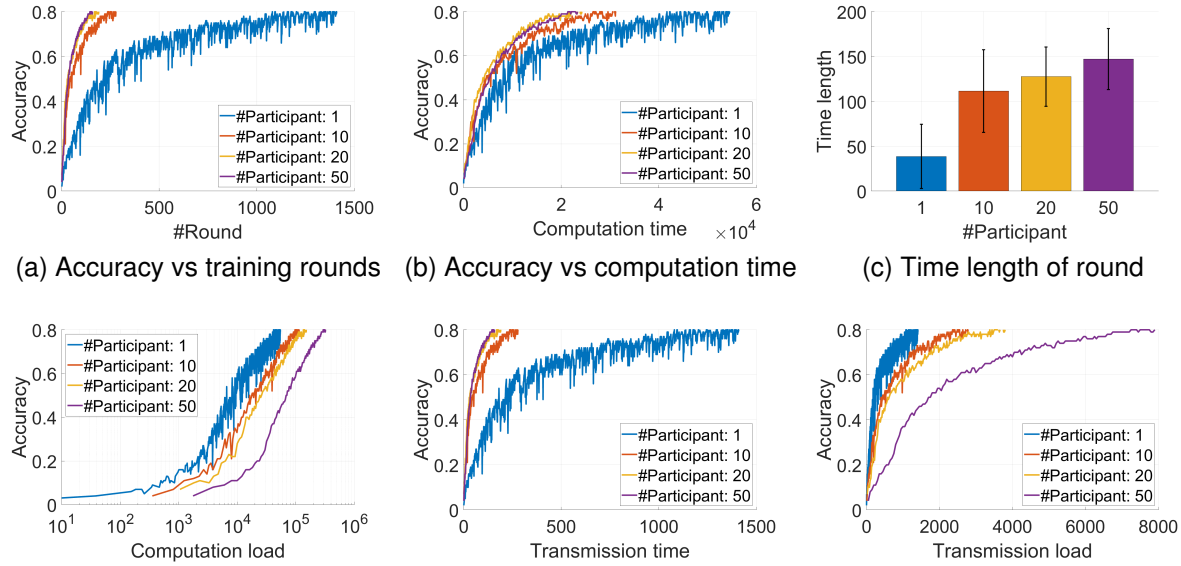
$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w) \tag{1}$$

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 the large number of clients in a typical FL application (e.g., millions of clients in the Google Gboard project [1]), a common practice is to randomly select a small fraction of clients in each training round. In the rest of this paper, we refer to the selected clients as participants and denote M as the number of participants in each training round. Each participant makes E training passes over its local data in 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.

A. System Model

Assume that clients are homogeneous regarding hardware (e.g., CPU/GPU) and network (e.g., transmission speeds). Let $b_{k,r}$ indicate 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. CompT, TransT, CompL, and TransL can be formulated as follows.

- **Computation Time (CompT)**. If client k is selected in a training round, its training time 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



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

(d) Accuracy vs computation load (e) Accuracy vs transmission time (f) Accuracy vs transmission load

Fig. 3. Illustration of FL training when a different number of participants are used in the measurement.

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 computation time of the training round r is determined by the slowest participant and thus is represented by $C_1 \cdot E \cdot \max_{k=1}^{K} b_{k,r} \cdot n_k$. In total, the computation time of an FL model training can be formulated as

$$CompT = C_1 \cdot E \cdot \sum_{r=1}^{R} \max_{k=1}^{K} b_{k,r} \cdot n_k \tag{2}$$

- **Transmission Time (TransT)**. Each participant in a training round needs one download and one upload of model parameters from and to the server [19]. Thus, the transmission time is the same for all participants in any training round, i.e., a constant C_2. The total transmission time is represented by

$$TransT = C_2 \cdot R \tag{3}$$

- **Computation Load (CompL)**. Client k has $C_3 \cdot E \cdot n_k$ computation load if it is selected in a training round, where C_3 is a constant. The computation load of the training round r is the summation of each participant's computation load and thus is $C_3 \cdot E \cdot \sum_{k=1}^{K} b_{k,r} \cdot n_k$. We can formulate the overall computation load as

$$CompL = C_3 \cdot E \cdot \sum_{r=1}^{R} \sum_{k=1}^{K} b_{k,r} \cdot n_k \tag{4}$$

- **Transmission Load (TransL)**. Since each training round selects M participants, the transmission load for a training round is $C_4 \cdot M$ where C_4 is a constant. The total number of training rounds is R, and thus, the total transmission load of an FL training is represented by

$$TransL = C_4 \cdot R \cdot M \tag{5}$$

As we assume that clients have homogeneous hardware and network, clients have the same C_1, C_2, C_3, and C_4. Thus, these constants do not affect the comparison of training overhead under different hyper-parameters when the same model is used. In the experiments, we assign the model's number of FLOPs for one input to C_1 and C_3, and the model's number of parameters to C_2 and C_4.

B. Measurement Setup

We conduct measurements to study the system overheads when different FL hyper-parameters are used for training. The measurements aim to help us better understand FL training and are the basis of our automatic tuning algorithm. We use the Google speech-to-command dataset [34], 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 in [34], we use 2112 clients' data for training and the remaining 506 clients' data for testing. Fig. 2a 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. 2b 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 demonstrates the three data properties of FL: massively distributed, unbalanced, and non-IID. Thus, the measurement observations from the Google speech-to-command dataset apply to other practical FL applications/datasets. In the measurement study, we investigate the FL training overhead in terms of the following three hyper-parameters.

- The number of participants (i.e., M). It is well-known that more participants in each training round have a better round-to-accuracy performance [8]. However, the time efficiency is not necessarily better because the time

length of each training round also increases with more participants. Besides, it is unclear how computation and communication overheads behaves for different number of participants. 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 [8] and FedNova [11]. However, how does the number of training passes affect the time overhead and computation overhead is unclear. 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 in each round.
- Model complexity. We also investigate how the model complexity influences the training overheads if a target accuracy is met. Although it is common knowledge that smaller models have better time and computation performance in other paradigms of model training, we are the first to report the four system overhead versus model complexity in the FL setting. We use ResNet [35] to build different models, as listed in Table III.

C. Illustration of FL Training

To gain an intuitive understanding of FL training, we plot FL training profiles in Fig. 3 where a different number of participants M is used on each round. We use ResNet-18 and set the target model accuracy to 0.8. In this measurement, E, C_1, C_2, C_3 and C_4 are all set to 1, for illustration purpose. Results are normalized to the largest overhead.

As expected, more participants lead to a better accuracy-to-round performance (Fig. 3a). However, if the FL training is stopped too early, it is misleading to conclude that more participants result in higher final model accuracy. Instead, they all reach the same model accuracy as FL hyper-parameters do not invalidate the convergence. Correspondingly, more participants have a better accuracy-to-CompT performance, as shown in Fig. 3b. However, the performance gap between a different number of participants for the accuracy-to-CompT is less significant than the accuracy-to-round performance. This is because the time duration of each training round increases when more participants are selected on each training round (Fig. 3c). In this measurement, there are no obvious accuracy-to-CompT difference when 20 and 50 participants are selected. On the other hand, fewer participants lead to better accuracy-to-CompL performance, as shown in Fig. 3d, which means the effectiveness of each unit computation operation is higher when fewer participants are involved in the FL training. The accuracy-to-TransT (Fig. 3e) is the same as the accuracy-to-round (Fig. 3a) since we adopt $C_3 = 1$ for plotting. That is, more participants have a better accuracy-to-TransT performance. Fig. 3f shows that the accuracy-to-TransL performance has the opposite trend, i.e., more participants result in a worse performance.

D. Measurement Results

In Section III-C, we illustrate CompT, CompL, TransT, and TransL performance when a different number of participants

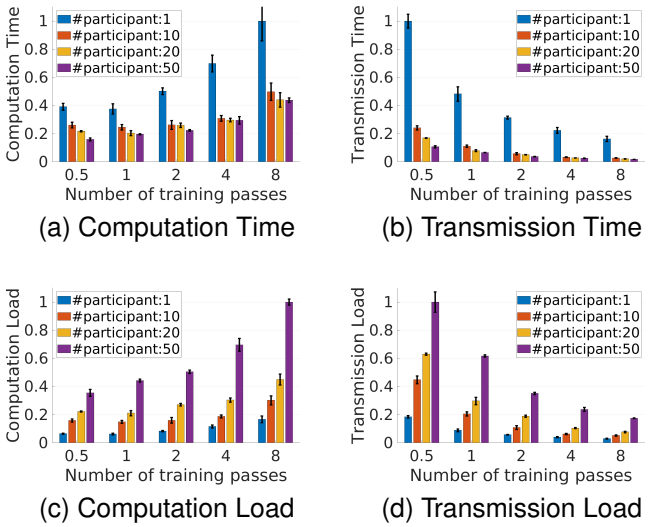


Fig. 4. CompT, TransT, CompL, and TransL when a different number of participants and a different number of training passes are used. The lower the better.

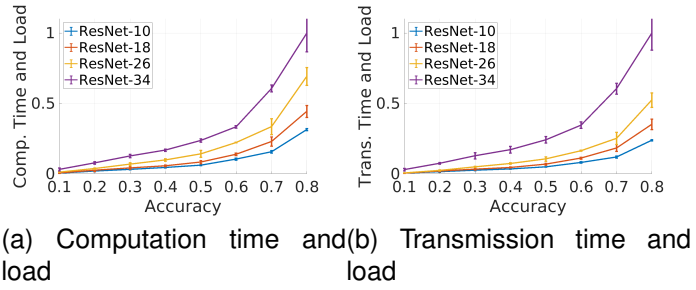


Fig. 5. CompT, TransT, CompL, and TransL versus model complexity. The lower the better.

are adopted in FL training. This section presents results from more extensive measurements for the number of participants M, the number of training passes E, and the model complexity. The results are averaged over three runs.

- **Computation Time (CompT)**. Fig. 4a compares CompT 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 CompT, i.e.; it takes a shorter time to converge. However, the difference is insignificant among 10, 20, and 50 participants, especially when the number of training passes is large. In addition, we can see that larger E has worse CompT. There is no apparent difference between $E = 0.5$ and $E = 1$ though. In a nutshell, the common knowledge of more participants is faster for FL model training is valid. However, the gain of more participants is insignificant when the number of participants is moderate. In addition, it is preferred to adopt a small number of training passes to achieve good time efficiency.
- **Transmission Time (TransT)**. Fig. 4b plots TransT, which clearly shows that TransT favors larger M and E. Since TransT is dependent on the number of training rounds R (Eq. (3)), it is equivalent to the metric of round-

to-accuracy. Our measurement result is consistent with common knowledge (e.g., [12]) that more participants and more training passes have a better round-to-accuracy performance. We can also observe that when M is small, e.g., 1, TransL is much worse than the other cases.

- **Computation Load (CompL)**. Fig. 4c shows CompL. We make the following observations: (1) More participants result in worse CompL. The results indicate that the gain of faster model convergence from more participants does not compensate for the higher computation costs introduced by more participants. (2) CompL is increased when more training passes are used. This is probably because that larger E diverges the model training [20], and thus, the data utility per unit of computation cost is reduced.
- **Transmission Load (TransL)**. Fig. 4d illustrates TransL. As shown, more participants greatly increase TransL. This is because more participants can only weakly reduce the number of training rounds R [11], however, in each round, the number of transmissions increases linearly with the number of participants. Regarding the number of training passes, larger E reduces the total number of training rounds R and thus has better TransL. On the other hand, the gain of larger E diminishes. The results are consistent with the analysis of [11] that R is hyperbolic with E (the turning point happens around 100-1000 in their experiments).
- **Model Complexity.** Table III 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. Fig. 5 shows the normalized CompT, TransT, CompL, and TransL for different models. The x-axis is the target model accuracy, and the y-axis is the corresponding overhead to reach that model accuracy. Since only one client and one training pass are used on each round, CompT and CompL have the same normalized comparison, and so are TransT and TransL. The results show that smaller models are better in terms of all training aspects. In addition, it is interesting to note that heavier models have higher increase rates of overheads versus model accuracy. This means that model selection is especially essential for high model accuracy applications.

E. Summary of System Overhead

Based on our measurement study, we summarize systems overheads versus FL hyper-parameters in Table IV. As we can see, CompT, TransT, CompL, and TransL conflict with each other in selecting the optimal M and E. Regarding model complexity, smaller models have better system overheads if the model accuracy is satisfied. Please note that Table IV is consistent with existing work (e.g., [12]), but is more comprehensive. Thus, Table IV is also valid for other datasets and ML models.

F. Intuitive Explanation of Measurement Results

We present intuitive explanation of the measurement results to help FL practitioners better understand FL hyper-parameters. Fig. 6 visualizes the intuition.

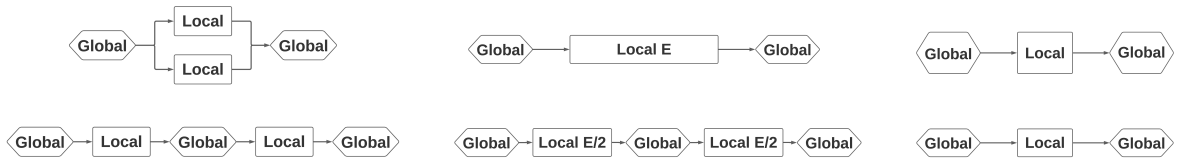
TABLE IV
SYSTEM OVERHEADS VERSUS THE NUMBER OF PARTICIPANTS M, THE NUMBER OF TRAINING PASSES E, AND MODEL COMPLEXITY. '$<$', '$=$', AND '$>$' MEANS THE SMALLER THE BETTER, DOES NOT MATTER, AND THE LARGER THE BETTER, RESPECTIVELY.

Training aspect	M	E	Model complexity
CompT	$>$	$<$	$<$
CompL	$<$	$<$	$<$
TransT	$>$	$>$	$<$
TransL	$<$	$>$	$<$
Model Accuracy	$=$	$=$	$>$

- *The number of participants* M. In Fig. 6a, the top scheme and the bottom scheme have the same computation cost (two local training) and communication cost (four transmissions). However, the bottom scheme has a better overall CompL and TransL at the expense of degraded CompT and TransT. 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 load and transmission load than wide-and-shallow FL schemes.
- *The number of training passes* E. In Fig. 6b, both the top scheme and the bottom scheme take E total training passes. However, the bottom scheme has better CompT and CompL 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 computation-sensitive FL applications, large E should be avoided.
- *Model complexity*. Fig. 6c shows that a smaller model has better CompT, TransT, CompL, and TransL than a heavier model, as long as the accuracy requirement is met. In addition, our results in Fig. 5 indicate that if the target model accuracy is low, then using a heavy model does not introduce significantly more overhead 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 system overhead.

IV. FEDTUNE: AUTOMATIC TUNING OF FEDERATED LEARNING HYPER-PARAMETERS FROM A SYSTEM PERSPECTIVE

Due to the conflicting FL hyper-parameters for CompT, TransT, CompL, and TransL, practitioners bear the burden of selecting a set of FL hyper-parameters, which if not chosen well may lead to poor system performance. For example, it is unclear how to set FL hyper-parameters that are both CompT and TransT friendly, since these two system aspects prefer different number of training passes. We propose FedTune to adjust FL hyper-parameters during the FL training automatically. FedTune considers the application's preference for CompT, TransT, CompL, and TransL, denoted by α, β, γ, and δ, respectively. We have $\alpha + \beta + \gamma + \delta = 1$. For example,



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

Fig. 6. Intuitive explanation of our measurement results. (a) The number of participants. The bottom scheme is better regarding CompL and TransL. (b) The number of training passes. The bottom scheme is better regarding CompT and CompL. (c) Model complexity. The bottom scheme is better for CompT, CompL, TransT, and TransL.

$\alpha = 0.6$, $\beta = 0.2$, $\gamma = 0.1$, and $\delta = 0.1$ represent that the application cares most about CompT, somewhat about TransT, and the least on CompL and TransL.

A. Problem Formulation

For two sets of FL hyper-parameters S_1 and S_2, FedTune defines the comparison function $I(S_1, S_2)$ as

$$I(S_1, S_2) = \alpha \times \frac{t_2 - t_1}{t_1} + \beta \times \frac{q_2 - q_1}{q_1} + \gamma \times \frac{z_2 - z_1}{z_1} + \delta \times \frac{v_2 - v_1}{v_1} \tag{6}$$

where t_1 and t_2 are CompT for S_1 and S_2 when achieving the same model accuracy. Correspondingly, q_1 and q_2 are TransT, z_1 and z_2 are CompL, and v_1 and v_2 are TransL. 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., CompT and CompL) is higher than the weighted degradation, if any, of the remaining training aspects (e.g., TransT and TransL). The weights are training preferences on CompT, TransT, CompL, and TransL.

However, the training overhead 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, minimizes the following objective function:

$$G(S_{nxt}) = \alpha \times \frac{t_{nxt} - t_{cur}}{t_{cur}} + \beta \times \frac{q_{nxt} - q_{cur}}{q_{cur}} + \gamma \times \frac{z_{nxt} - z_{cur}}{z_{cur}} + \delta \times \frac{v_{nxt} - v_{cur}}{v_{cur}} \tag{7}$$

where t_{cur}, q_{cur}, z_{cur}, and v_{cur} are CompT, TransT, CompL, and TransL under the current hyper-parameters S_{cur}; t_{nxt}, q_{nxt}, z_{nxt}, and v_{nxt} are CompT, TransT, CompL, and TransL for the next hyper-parameters S_{nxt}. We focus on the number of participants M and the number of training passes E, since training overhead is monotonous with model complexity. Therefore, we need to optimize $S_{nxt} = \{M_{nxt}, E_{nxt}\}$.

B. S_{nxt} Optimization

To find the optimal S_{nxt}, we take the derivatives of $G(S_{nxt})$ over M and E, obtaining

$$\Delta M = \frac{\partial G(S_{nxt})}{\partial M} = \frac{\alpha}{t_{cur}} \times \frac{\partial t_{nxt}}{\partial M} + \frac{\beta}{q_{cur}} \times \frac{\partial q_{nxt}}{\partial M} + \frac{\gamma}{z_{cur}} \times \frac{\partial z_{nxt}}{\partial M} + \frac{\delta}{v_{cur}} \times \frac{\partial v_{nxt}}{\partial M} \tag{8}$$

$$\Delta E = \frac{\partial G(S_{nxt})}{\partial E} = \frac{\alpha}{t_{cur}} \times \frac{\partial t_{nxt}}{\partial E} + \frac{\beta}{q_{cur}} \times \frac{\partial q_{nxt}}{\partial E} + \frac{\gamma}{z_{cur}} \times \frac{\partial z_{nxt}}{\partial E} + \frac{\delta}{v_{cur}} \times \frac{\partial v_{nxt}}{\partial E} \tag{9}$$

We illustrate how to approximate ΔM. The process for ΔE is similar. Considering that each step makes a small adjustment of M, $\partial t_{nxt}/\partial M$ can be represented by $(+1) \times |t_{nxt} - t_{cur}|$, where $(+1)$ means CompT prefers larger M according to Table IV. To estimate $|t_{nxt} - t_{cur}|$, we apply a linear function $\eta_{t-1} \times |t_{cur} - t_{prv}|$ where $\eta_{t-1} = \frac{|t_{cur} - t_{prv}|}{|t_{prv} - t_{prvprv}|}$ (t_{prvprv} is the CompT at two steps before). Thus, η_{t-1} represents the slope of the linear function. Similarly, we have η_{q-1}, η_{z-1}, η_{v-1} for TransT, CompL, and TransL when calculating their derivatives over M. As a result, ΔM can be approximated as

$$\Delta M = \frac{(+1) \times \alpha \times \eta_{t-1} \times |t_{cur} - t_{prv}|}{t_{cur}} + \frac{(+1) \times \beta \times \eta_{q-1} \times |q_{cur} - q_{prv}|}{q_{cur}} + \frac{(-1) \times \gamma \times \eta_{z-1} \times |z_{cur} - z_{prv}|}{z_{cur}} + \frac{(-1) \times \delta \times \eta_{v-1} \times |v_{cur} - v_{prv}|}{v_{cur}} \tag{10}$$

Similarly, we can calculate ΔE as

$$\Delta E = \frac{(-1) \times \alpha \times \zeta_{t-1} \times |t_{cur} - t_{prv}|}{t_{cur}} + \frac{(+1) \times \beta \times \zeta_{q-1} \times |q_{cur} - q_{prv}|}{q_{cur}} + \frac{(-1) \times \gamma \times \zeta_{z-1} \times |z_{cur} - z_{prv}|}{z_{cur}} + \frac{(+1) \times \delta \times \zeta_{v-1} \times |v_{cur} - v_{prv}|}{v_{cur}} \tag{11}$$

where ζ_{t-1}, ζ_{q-1}, ζ_{z-1}, and ζ_{v-1} are the parameters for calculating the derivatives of CompT, TransT, CompL, and TransL over E.

TABLE V
PERFORMANCE OF FEDTUNE FOR THE SPEECH-TO-COMMAND DATASET WHEN FEDADAGRAD IS USED FOR AGGREGATION. '+' IS IMPROVEMENT AND '−' IS DEGRADATION. STANDARD DEVIATION IN PARENTHESES AND THE BEST RESULTS ARE IN BOLD TEXTS.

α	β	γ	δ	CompT (10^{12})	TransT (10^6)	CompL (10^{12})	TransL (10^6)	Final M	Final E	Overall
-	-	-	-	0.94 (0.01)	11.61 (0.10)	5.97 (0.04)	232.24 (1.99)	20	20	-
1.0	0.0	0.0	0.0	**0.42** (0.02)	50.19 (2.57)	4.57 (0.22)	2418.71 (240.91)	57.33 (4.50)	1.00 (0.00)	+55.23% (2.22%)
0.0	1.0	0.0	0.0	1.34 (0.22)	**7.68** (1.12)	14.99 (2.73)	289.82 (46.98)	48.00 (2.16)	48.00 (2.16)	+33.87% (9.67%)
0.0	0.0	1.0	0.0	1.02 (0.10)	615.98 (97.52)	**1.76** (0.16)	672.21 (91.62)	1.00 (0.00)	1.00 (0.00)	**+70.51%** (2.75%)
0.0	0.0	0.0	1.0	2.18 (0.47)	35.47 (7.51)	3.30 (0.22)	**76.47** (1.68)	1.00 (0.00)	46.67 (3.30)	+67.07% (0.72%)
0.5	0.5	0.0	0.0	0.82 (0.13)	9.17 (1.26)	9.13 (1.66)	347.11 (54.31)	47.33 (2.05)	21.33 (4.78)	+16.97% (9.68%)
0.5	0.0	0.5	0.0	0.48 (0.04)	81.42 (9.83)	3.23 (0.14)	1875.99 (155.21)	25.00 (1.63)	1.00 (0.00)	+47.57% (3.43%)
0.5	0.0	0.0	0.5	0.79 (0.10)	11.59 (0.55)	5.04 (0.89)	241.86 (68.65)	22.33 (5.79)	15.67 (4.50)	+5.82% (11.28%)
0.0	0.5	0.5	0.0	0.83 (0.03)	10.66 (0.15)	5.16 (0.31)	207.79 (6.08)	21.00 (1.41)	21.00 (1.41)	+10.87% (2.83%)
0.0	0.5	0.0	0.5	1.54 (0.16)	11.48 (3.83)	9.59 (3.52)	190.52 (61.53)	19.67 (14.82)	49.00 (0.00)	+9.55% (7.08%)
0.0	0.0	0.5	0.5	1.69 (0.26)	50.14 (8.21)	2.70 (0.26)	93.21 (8.48)	1.00 (0.00)	23.33 (2.49)	+57.32% (3.76%)
0.33	0.33	0.33	0.0	0.82 (0.07)	11.59 (1.01)	5.65 (0.27)	255.35 (9.65)	22.33 (2.62)	15.67 (1.25)	+6.09% (6.67%)
0.33	0.33	0.0	0.33	1.06 (0.08)	10.07 (0.90)	8.10 (0.34)	247.54 (29.18)	26.33 (2.05)	27.00 (2.16)	-1.93% (7.40%)
0.33	0.0	0.33	0.33	0.91 (0.19)	18.23 (5.83)	4.15 (1.13)	229.26 (63.40)	12.00 (1.41)	14.00 (5.72)	+11.66% (11.76%)
0.0	0.33	0.33	0.33	1.13 (0.13)	16.16 (3.36)	4.51 (0.59)	169.93 (25.84)	9.00 (5.35)	23.00 (4.55)	+3.99% (6.19%)
0.25	0.25	0.25	0.25	0.91 (0.10)	9.73 (1.81)	6.19 (0.76)	207.34 (3.34)	23.33 (5.44)	22.67 (3.30)	+6.51% (6.13%)

C. Decision Making and Parameter Update

FedTune is activated when the model accuracy is improved by at least ϵ. Then, it normalizes current overheads and calculates the comparison function of the previous hyper-parameters S_{prv} and the current hyper-parameters S_{cur}. Afterward, FedTune updates the parameters, i.e., four η and four ζ. Next, it computes ΔM and ΔE, and determines the next M and E based on the signs of ΔM and ΔE. Specifically, $M_{nxt} = M_{cur} + 1$ if $\Delta M > 0$, otherwise, $M_{nxt} = M_{cur} - 1$. Likewise, FedTune increases E_{nxt} by one if $\Delta E > 0$; else FedTune decreases E_{nxt} by one. The FL training continues using the new hyper-parameters. It is clear that FedTune is lightweight and its computational burden is negligible to the FL training: it only requires dozens of multiplication and addition calculations (refer to our source code for more details).

FedTune automatically updates η_{t-1}, η_{q-1}, η_{z-1}, η_{v-1}, ζ_{t-1}, ζ_{q-1}, ζ_{z-1}, and ζ_{v-1} during FL training. At each step, FedTune updates the parameters that favor the current decision. For example, if M_{cur} is larger than M_{prv}, FedTune updates η_{t-1} and η_{q-1} as CompT and TransT prefer larger M; otherwise, FedTune updates η_{z-1} and η_{v-1}.

Furthermore, FedTune incorporates a penalty mechanism to mitigate bad decisions. Given the previous hyper-parameters S_{prv} and the current hyper-parameters S_{cur}, FedTune calculates the comparison function $I(S_{prv}, S_{cur})$. A bad decision occurs if the sign of $I(S_{prv}, S_{cur})$ is positive. In this case, FedTune multiplies the parameters that are against the current decision by a constant penalty factor, denoted by D ($D \geq 1$). For example, if $I(S_{prv}, S_{cur}) > 0$ and $M_{cur} > M_{prv}$, FedTune updates η_{t-1} and η_{q-1} as explained before, but also multiplies η_{z-1} and η_{v-1} by D.

Please note that the new hyper-parameters are based on the eight parameters (four η and four ζ) that are updated automatically. These eight parameters along with the training overheads determine the next hyper-parameters as shown in Eq. (10) and Eq. (10).

V. EVALUATION

In this section, we provide the evaluation results of FedTune. First, we explain our experiment setup in Section V-A. Next, we present the overall performance in Section V-B. Then, we conduct trace analysis in Section V-C. Last, the penalty mechanism is studied in Section V-D.

A. Experiment Setup

Benchmarks and Baseline. We evaluate FedTune on three datasets: speech-to-command [34], EMNIST [36], and Cifar-100 [37], and three aggregation methods: FedAvg [8], FedNova [12], and FedAdagrad [38]. We set equal values for the combination of training preferences α, β, γ and δ (see the first column in Table V). Therefore, for each dataset, we conduct 15 combinations of training preferences. We set target model accuracy for each dataset and measure CompT, TransT, CompL, and TransL for reaching the target model accuracy. We regard the practice of using fixed M and E as the baseline and compare FedTune to the baseline by calculating Eq. (6). In the evaluation, the positive performance means FedTune reduces the system overheads and the negative performance means the degradation. We implemented FedTune in PyTorch. All the experiments are conducted in a server with 24-GB Nvidia RTX A5000 GPUs.

Training Setup. (1) *speech-to-command* dataset. It classifies audio clips to 35 commands (e.g., 'yes', 'off'). We transform audio clips to 64-by-64 spectrograms and then downsize them to 32-by-32 gray-scale images. As officially suggested [34], we use 2112 clients' data for training and the remaining 506 clients' data for testing. We set the mini-batch size to 5, considering that many clients have few data points. We use ResNet-10 and the target model accuracy of 0.8. (2) *EMNIST* dataset. It classifies handwriting (28-by-28 gray-scale images) into 62 digits and letters (lowercase and uppercase). We split the dataset based on the writer ID. We randomly select 70% writers' data for training and the remaining for testing. We use a Multiplayer Perception (MLP) model with one hidden layer (200 neurons with ReLu activation). We set the mini-batch size to 10 and the target model accuracy of 0.7. (3)

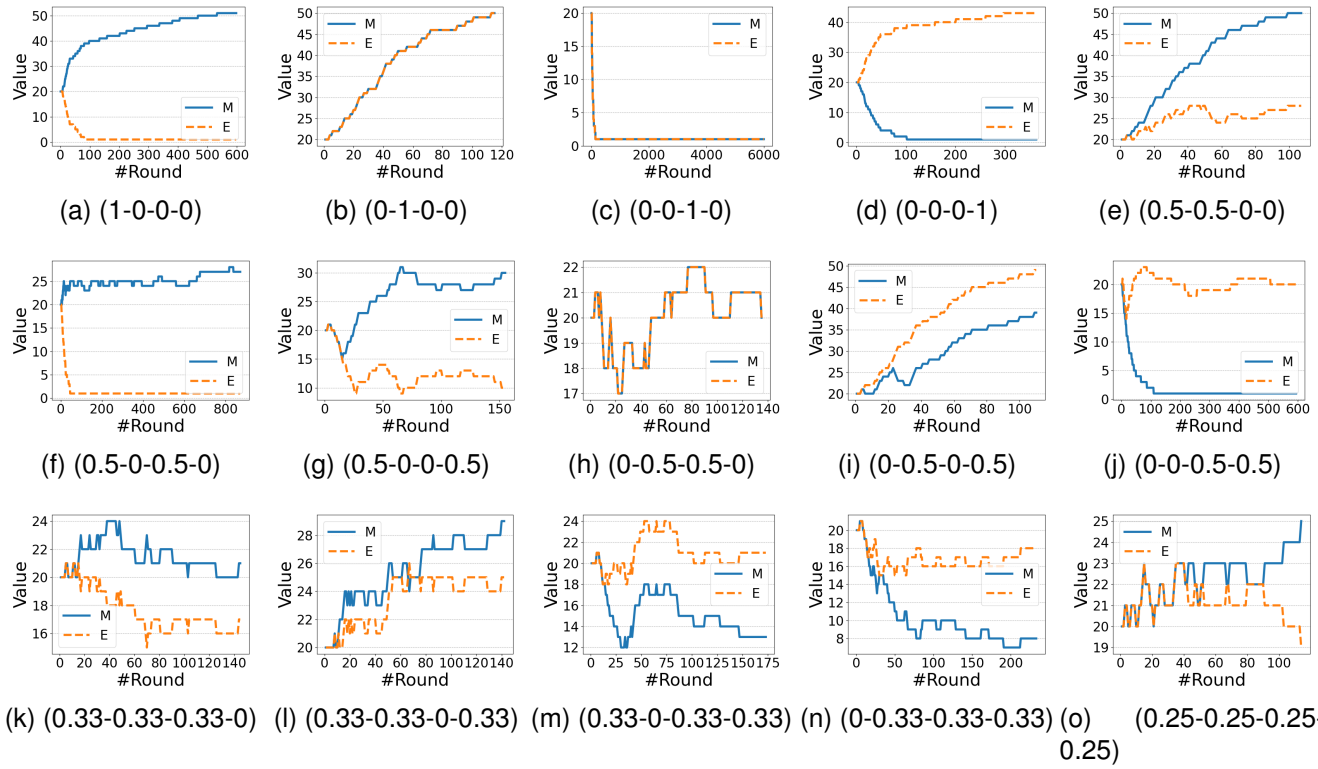


Fig. 7. Illustrations of the number of participant M and the number of training passes E during FL training for different application preferences.

TABLE VI
PERFORMANCE OF FEDTUNE FOR DIVERSE DATASETS WHEN FEDAVG AGGREGATION METHOD IS APPLIED.

Dataset Data Feature ML Model	Speech-command Voice ResNet-10	EMNIST Handwriting 2-layer MLP	Cifar-100 Image ResNet-10
Performance	**+22.48%** (17.97%)	+8.48% (5.51%)	+9.33% (5.47%)

TABLE VII
PERFORMANCE OF FEDTUNE FOR DIVERSE AGGREGATION ALGORITHMS. SPEECH-TO-COMMAND DATASET AND RESNET-10 ARE USED IN THIS EXPERIMENT.

Aggregator	FedAvg	FedNova	FedAdagrad
Performance	+22.48% (17.97%)	+23.53% (6.64%)	**+26.75%** (6.10%)

Cifar-100 dataset. It classifies 32-by-32 RGB images to 100 classes. We randomly split the dataset into 1200 users, where each user has 50 data points. Then, we randomly select 1000 users for training and the remaining 200 users for testing. We set the mini-batch size to 10. ResNet-18 is used, and the target model accuracy is set to 0.2 (due to our limited computational capability, we set a low threshold for Cifar-100).

For all datasets, we normalize the input images with the mean and the standard deviation of the training data before feeding them to models for training and testing. Both M and E are initially set to 20. FedTune is activated when the model accuracy is increased by at least 0.01 (i.e., $\epsilon = 0.01$). The penalty factor D is set to 10. All results are averaged by three experiments. Note that FedTune is the first work of its kind and we choose the experiment settings as reasonably as possible.

B. Overall Performance

Results for Diverse Datasets. Table VI shows the overall performance of FedTune for different datasets when FedAvg is applied. We set the learning rate to 0.01 for the speech-to-command dataset and the EMNIST dataset, and 0.1 for the Cifar-100 dataset, all with the momentum of 0.9. We show the standard deviation in parenthesis. As shown, FedTune consistently improves the system performance across all the three datasets. In particular, FedTune reduces 22.48% system overhead of the speech-to-command dataset compared to the baseline by averaging the 15 combinations of training preferences. We also observe that the FL training benefits more from FedTune if the training process needs more training rounds to converge. Our experiments with EMNIST (small model) and Cifar100 (low target accuracy) only require a few dozens of training rounds to reach their target model accuracy, and thus their performance gains from FedTune are not significant. The observation is consistent with the decision-making process in FedTune, which increases/decreases hyper-parameters by only one at each step. We leave it as future work to augment FedTune to change hyper-parameters with adaptive degrees.

Results for Different Aggregation Methods. Table VII shows the overall performance of FedTune for different aggregation methods when we use the speech-to-command dataset and the ResNet-10 model. We set the learning rate to 0.1, β_1 to 0, and τ to 1e-3 in FedAdagrad. As shown, FedTune achieves consistent performance gain for diverse aggregation

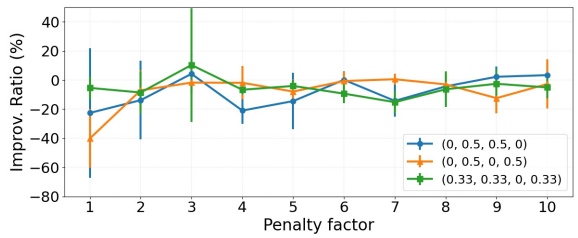


Fig. 8. Performance of the degraded cases versus the penalty factor. Speech-to-command dataset and FedAvg is used in this experiment.

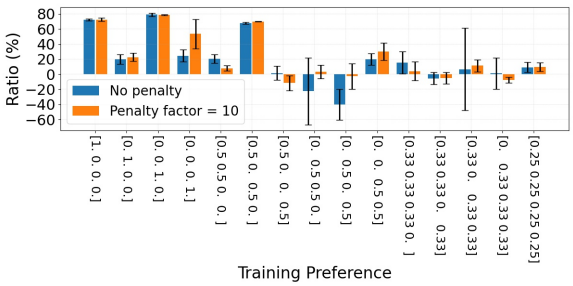


Fig. 9. Performance comparison of FedTune without penalty mechanism and with penalty mechanism for the Speech-to-command dataset and FedAvg aggregation method.

methods. In particular, FedTune reduces the system overhead of FedAdagrad by 26.75%.

C. Trace Analysis

We present the details of traces when the speech-to-command dataset and the FedAdagrad aggregation method are used. Table V tabulates the results, where we show the application preference (α, β, γ, and δ), the system overheads (CompT, TransT, CompL, and TransL), the final M and E when the training is finished, and the overall performance. We report the average performance, as well as their standard deviations in parentheses. The first row is the baseline, which does not change hyper-parameters during the FL training. As we can see from Table V, FedTune can adapt to different training preferences. Specifically, FedTune reduces the system overhead up to 70.51% when the application only cares about computation load (i.e., $\gamma = 1$). Only one preference (0.33, 0.33, 0, 0.33) results in a slightly degraded performance. On average, FedTune improves the overall performance by 26.75% for the speech-to-command dataset and the FedAdagrad aggregation method.

Fig. 7 illustrates one trace of M and E during the FL training for each application preference. The experiment is conducted with the speech-to-command dataset and the FedAdagrad aggregation method. The experiment result clearly shows that FedTune can automatically adjust FL hyper-parameters during the FL training while respecting the application's preference on system overhead. We also observe that FedTune does not keep increasing or decreasing M and E. Instead, in many cases, it intelligently tunes the M and E during the different phases of the FL training.

D. Study of Penalty Mechanism

We investigate our penalty mechanism by conducting experiments with the Google speech-to-command dataset and the FedAvg aggregation method. Without the penalty mechanism (equivalently $D = 1$ since the penalty factor is a multiplier), we find that FedTune results in three degraded cases, i.e., the preferences of (0, 0.5, 0.5, 0), (0, 0.5, 0, 0.5), and (0.33, 0.33, 0, 0.33).

First, we conduct experiments to explore whether the penalty mechanism can mitigate the degradation. Results are averaged over three runs. Fig. 8 plots the performance of FedTune for the degraded cases versus penalty factors, where error bars represent the standard deviation. Although the penalty mechanism does not guarantee positive performance for the degraded cases, it mitigates the degradation. Fig. 8 also shows that FedTune remains stable for a moderate value of penalty factors. Empirically, we set the penalty factor to 10 in FedTune.

Fig. 9 compares the performance of FedTune when the penalty factor is 10 (full-fledged) versus 1 (no penalty) for different training preferences. Results are averaged by three runs and the error bars represent the standard deviation. Overall, FedTune increases the performance gain of the FedAvg aggregation method from 17.97% to 22.48%. In addition, FedTune with the penalty mitigates the highly degraded cases and thus is more practical than the non-penalty counterpart. We also observe that FedTune with penalty mechanism is more stable, with the averaged standard deviation of 7.77% versus 14.14% in the non-penalty counterpart.

VI. Discussion

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

Heterogeneous Devices. This paper assumes that clients are homogeneous, i.e., the same hardware and network. In practice, however, client devices are heterogeneous. Measurements of the computation capabilities of mobile devices and their network throughput exhibit order-of-magnitude difference [39]–[41]. As a result, for clients even with the same amount of local data points, they result in different computation and transmission costs. Currently, FedTune only tunes the system-wide hyper-parameters, which might not be optimal for heterogeneous devices that may require device-level hyper-parameter tuning. However, compared to the overwhelming and laboring effort of manually determining the FL hyper-parameters for each device, tuning system-wide hyper-parameter seems to be a more practical approach for FL practitioners. We leave it as future work to evaluate FedTune on heterogeneous devices.

Extensions. Many FL algorithms have been proposed to tackle the limitations of FedAvg. FedTune could be extended 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 [9]. Popular alternatives are to only wait for participants that are finished before a deadline [42] or only wait for the first M participants [11]. (2) Adaptive training passes across participants. Due to the heterogeneity of clients, setting the same number of training

passes E for all participants in each training round is not optimal. To support different E across participants, FedNova [12] relies on re-weighting of aggregation while FedProx [20] adds a proximal term to stabilize the convergence. We plan to incorporate these functionality to further improve the performance of FedTune. (3) Investigate more FL hyper-parameters. Although we only take the number of participants, the number of training passes, and the model complexity as examples to illustrate the workflow of FedTune, we believe FedTune is a general framework, which can be applied to other FL hyper-parameters, e.g., the mini-batch size.

VII. Conclusion

FL involves high system overhead in the training process, which hinders its research and real-world deployment. We argue that optimizing system overhead for FL applications is extremely valuable. To this end, we propose FedTune to adjust FL hyper-parameters, catering to the application's training preferences automatically. Although FedTune is simple to implement, it is powerful and effective. Our evaluation results show that FedTune is general, lightweight, flexible, and is able to significantly reduce system overhead in FL training. As the first work of its kind that considers comprehensive system overheads, we only compare FedTune with the baseline because existing FL hyper-parameter automatic tuning work is either incompatible with standard FL algorithms, complex and cumbersome to implement or insignificant (improved by only a few percentage points). Nonetheless, we plan to evaluate FedTune more thoroughly in scenarios such as heterogeneous devices and real-world deployments. We will also provide a theoretical analysis framework for FedTune in future work.

References

[1] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated Learning for Mobile Keyboard Prediction," *arXiv:1811.03604*, 2019.

[2] M. Paulik, M. Seigel, H. Mason, D. Telaar, J. Kluivers, R. van Dalen, C. W. Lau, L. Carlson, F. Granqvist, C. Vandevelde, S. Agarwal, J. Freudiger, A. Byde, A. Bhowmick, G. Kapoor, S. Beaumont, A. Cahill, D. Hughes, O. Javidbakht, F. Dong, R. Rishi, and S. Hung, "Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications," *arXiv:2102.08503*, 2021.

[3] C. Ju, R. Zhao, J. Sun, X. Wei, X. Zhang, D. Gao, B. Tan, H. Yu, C. He, and Y. Jin, "Privacy-Preserving Technology to Help Millions of People: Federated Prediction Model for Stroke Prevention," *arXiv:2006.10517*, 2020.

[4] M. Zhang, F. Zhang, N. Lane, Y. Shu, X. Zeng, B. Fang, S. Yan, and H. Xu, "Deep Learning in the Era of Edge Computing: Challenges and Opportunities," in *Book chapter in Fog Computing: Theory and Practice, Wiley*, 2020.

[5] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and S. Avestimehr, "Federated learning for internet of things: Applications, challenges, and opportunities," *IEEE Internet of Things Magazine*, 2022.

[6] M. I. Jordan and T. M. Mitchell, "Machine Learning: Trends, Perspectives, and Prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[7] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[8] H. B. McMahan, D. R. Eider Moore, S. Hampson, and B. A. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1–10.

[9] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient Federated Learning via Guided Participant Selection," in *USENIX Symposium on Operating Systems Design and Implementation*, 2021.

[10] C. Li, X. Zeng, M. Zhang, and Z. Cao, "PyramidFL: A Fine-grained Client Selection Framework for Efficient Federated Learning," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2022.

[11] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the Convergence of FedAvg on Non-IID Data," in *International Conference on Learning Representations (ICLR)*, 2020, pp. 1–12.

[12] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. Vincent Poor, "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization," in *Conference on Neural Information Processing System (NeurIPS)*, 2020.

[13] S. H. Haji and S. Y. Ameen, "Attack and Anomaly Detection in IoT Networks using Machine Learning Techniques: A Review," *Asian Journal of Research in Computer Science (AJRCOS)*, vol. 9, no. 2, pp. 30–46, 2021.

[14] D. N. Mekuria, P. Sernani, N. Falcionelli, and A. F. Dragoni, "Smart Home Reasoning Systems: A Systematic Literature Review," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 4485–4502, 2021.

[15] M. Won, "Intelligent Traffic Monitoring Systems for Vehicle Classification: A Survey," *IEEE Access*, vol. 8, pp. 73 340–73 358, 2020.

[16] A. Sharma, A. Jain, P. Gupta, and V. Chowdary, "Machine Learning Applications for Precision Agriculture: A Comprehensive Review," *IEEE Access*, vol. 9, pp. 4843–4873, 2020.

[17] M. M. Hassan, A. Gumaei, G. Aloi, G. Fortino, and M. Zhou, "A Smartphone-Enabled Fall Detection Framework for Elderly People in Connected Home Healthcare," *IEEE Access*, vol. 33, pp. 58–63, 2019.

[18] M. M. de Almeida and J. von Schreeb, "A Smartphone-Enabled Fall Detection Framework for Elderly People in Connected Home Healthcare," *Prehospital and Disaster Medicine*, vol. 34, pp. 82–88, 2018.

[19] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, B. A. y. Arcas, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, S. Diggavi, H. Eichner, A. Gadhikar, Z. Garrett, A. M. Girgis, F. Hanzely, A. Hard, C. He, S. Horvath, Z. Huo, A. Ingerman, M. Jaggi, T. Javidi, P. Kairouz, S. Kale, S. P. Karimireddy, J. Konecny, and etc, "A Field Guide to Federated Optimization," *arXiv: 2107.06917*, 2021.

[20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *Conference on Machine Learning and Systems (MLSys)*, 2020, pp. 429–450.

[21] H. Zhang, M. Zhang, X. Liu, P. Mohapatra, , and M. DeLucia, "Fedtune: Automatic tuning of federated learning hyper-parameters from system perspective," in *IEEE Military Communications Conference (MILCOM)*, 2022.

[22] Z. Dai, B. K. H. Low, and P. Jaillet, "Federated bayesian optimization via thompson sampling," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[23] Z. Li, H. Li, and M. Zhang, "Hyper-parameter tuning of federated learning based on particle swarm optimization," in *IEEE International Conference on Cloud Computing and Intelligent Systems (CCIS)*, 2021.

[24] Z. Dai, B. K. Low, and P. Jaillet, "Differentially private federated bayesian optimization with distributed exploration," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[25] P. Guo, D. Yang, A. Hatamizadeh, A. Xu, Z. Xu, W. Li, C. Zhao, D. Xu, S. Harmon, E. Turkbey, B. Turkbey, B. Wood, F. Patella, E. Stellato, G. Carrafiello, V. M. Patel, and H. R. Roth, "Auto-FedRL: Federated Hyperparameter Optimization for Multi-institutional Medical Image Segmentation," *arXiv:2203.06338*, pp. 1–18, 2022.

[26] H. Mostafa, "Robust Federated Learning Through Representation Matching and Adaptive Hyperparameters," *arXiv:1912.13075*, pp. 1–11, 2019.

[27] M. Khodak, R. Tu, T. Li, L. Li, M.-F. Balcan, V. Smith, and A. Talwalkar, "Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[28] Y. Zhou, P. Ram, T. Salonidis, N. Baracaldo, H. Samulowitz, and H. Ludwig, "FLoRA: Single-shot Hyper-parameter Optimization for Federated Learning," *arXiv*, pp. 1–11, 2021.

[29] L. Yang and A. Shami, "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice," *Neurocomputing*, 2020.

[30] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012.

[31] Z. Karnin, T. Koren, and O. Somekh, "Almost Optimal Exploration in Multi-Armed Bandits," in *International Conference on Machine Learning (ICML)*, 2013, pp. 1238–1246.

[32] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization," *Journal of Machine Learning Research (JMLR)*, 2017.
[33] Z. WANG, W. Kuang, C. Zhang, B. Ding, and Y. Li, "FedHPO-B: A Benchmark Suite for Federated Hyperparameter Optimization," *arXiv:2206.03966*, pp. 1–27, 2022.
[34] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv: 1804.03209*, 2018.
[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE CVPR*, 2016.
[36] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," in *International Joint Conference on Neural Networks (IJCNN)*, 2017.
[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.
[38] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konecny, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *International Conference on Learning Representations (ICLR)*, 2021.
[39] A. Ignatov, "AI-Benchmark," https://ai-benchmark.com/index.html, 2021.
[40] M-Lab, "MobiPerf," https://www.measurementlab.net/tests/mobiperf, 2021.
[41] F. Lai, Y. Dai, X. Zhu, and M. Chowdhury, "FedScale: Benchmarking Model and System Performance of Federated Learning," *arXiv: 2105.11367*, pp. 1–15, 2021.
[42] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards Federated Learning at Scale: System Design," in *Conference on Machine Learning and Systems (MLSys)*, 2019, pp. 374–388.