Federated Hyperparameter Tuning: Challenges, Baselines, and Connections to Weight-Sharing

Mikhail Khodak, Renbo Tu, Tian Li
Carnegie Mellon University
{khodak,renbo,tianli}@cmu.edu

Liam Li
Hewlett Packard Enterprise
me@liamcli.com

Maria-Florina Balcan, Virginia Smith
Carnegie Mellon University
ninamf@cs.cmu.edu,smithv@cmu.edu

Ameet Talwalkar
Carnegie Mellon University & Hewlett Packard Enterprise
talwalkar@cmu.edu

Abstract

Tuning hyperparameters is a crucial but arduous part of the machine learning pipeline. Hyperparameter optimization is even more challenging in federated learning, where models are learned over a distributed network of heterogeneous devices; here, the need to keep data on device and perform local training makes it difficult to efficiently train and evaluate configurations. In this work, we investigate the problem of federated hyperparameter tuning. We first identify key challenges and show how standard approaches may be adapted to form baselines for the federated setting. Then, by making a novel connection to the neural architecture search technique of *weight-sharing*, we introduce a new method, `FedEx`, to accelerate federated hyperparameter tuning that is applicable to widely-used federated optimization methods such as `FedAvg` and recent variants. Theoretically, we show that a `FedEx` variant correctly tunes the on-device learning rate in the setting of online convex optimization across devices. Empirically, we show that `FedEx` can outperform natural baselines for federated hyperparameter tuning by several percentage points on the Shakespeare, FEMNIST, and CIFAR-10 benchmarks—obtaining higher accuracy using the same training budget.

1 Introduction

Federated learning (FL) is a popular distributed computational setting where training is performed locally or privately [30, 36] and where hyperparameter tuning has been identified as a critical problem [18]. Although general hyperparameter optimization has been the subject of intense study [3, 16, 26], several unique aspects of the federated setting make tuning hyperparameters especially challenging. However, to the best of our knowledge there has been no dedicated study on the specific challenges and solutions in federated hyperparameter tuning. In this work, we first formalize the problem of hyperparameter optimization in FL, introducing the following three key challenges:

1. **Federated validation data:** In federated networks, as the validation data is split across devices, the entire dataset is not available at any one time; instead a central server is given access to some number of devices at each communication round, for one or at most a few runs of local training and validation. Thus, because the standard measure of complexity in FL is the number of communication rounds, computing validation metrics exactly dramatically increases the cost.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

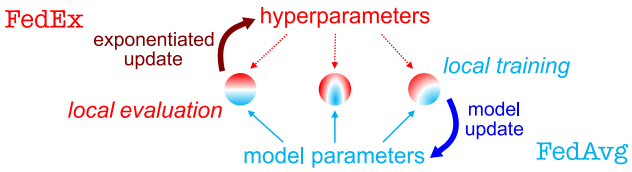

Figure 1: FedEx can be applied to any local training-based FL method, e.g. FedAvg, by interleaving standard updates to model weights (computed by aggregating results of local training) with exponentiated gradient updates to hyperparameters (computed by aggregating results of local validation).

2. **Extreme resource limitations:** FL applications often involve training using devices with very limited computational and communication capabilities. Furthermore, many require the use of privacy techniques such as differential privacy that limit the number times user data can be accessed. Thus we cannot depend on being able to run many different configurations to completion.
3. **Evaluating personalization:** Finally, even with non-federated data, applying common hyperparameter optimization methods to standard personalized FL approaches (such as finetuning) can be costly because evaluation may require performing many additional training steps locally.

With these challenges[1] in mind, we propose reasonable baselines for federated hyperparameter tuning by showing how to adapt standard non-federated algorithms. We further study the challenge of noisy validation signal due to federation, and show that simple state-estimation-based fixes do not help.

Our formalization and analysis of this problem leads us to develop FedEx, a method that exploits a novel connection between hyperparameter tuning in FL and the weight-sharing technique widely used in neural architecture search (NAS) [4, 34, 40]. In particular, we observe that weight-sharing is a natural way of addressing the three challenges above for federated hyperparameter tuning, as it incorporates noisy validation signal, simultaneously tunes and trains the model, and evaluates personalization as part of training rather than as a costly separate step. Although standard weight-sharing only handles architectural hyperparameters such as the choice of layer or activation, and not critical settings such as those of local stochastic gradient descent (SGD), we develop a formulation that allows us to tune most of these as well via the relationship between local-training and fine-tuning-based personalization. This make FedEx a general hyperparameter tuning algorithm applicable to many local training-based FL methods, e.g. FedAvg [36], FedProx [31], and SCAFFOLD [19].

In Section 4, we next conduct a theoretical study of FedEx in a simple setting: tuning the client step-size. Using the ARUBA framework for analyzing meta-learning [20], we show that a variant of FedEx correctly tunes the on-device step-size to minimize client-averaged regret by adapting to the intrinsic similarity between client data. We improve the convergence rate compared to some past meta-learning theory [20, 25] while not depending on knowing the (usually unknown) task-similarity.

Finally, in Section 5, we instantiate our baselines and FedEx to tune hyperparameters of FedAvg, FedProx, and Reptile, evaluating on three standard FL benchmarks: Shakespeare, FEMNIST, and CIFAR-10 [5, 36]. While our baselines already obtain performance similar to past hand-tuning, FedEx further surpasses them in most settings examined, including by 2-3% on Shakespeare.

Related Work To the best of our knowledge, we are the first to systematically analyze the formulation and challenges of hyperparameter optimization in the federated setting. Several papers have explored limited aspects of hyperparameter tuning in FL [7, 23, 38], focusing on a small number of hyperparameters (e.g. the step-size and sometimes one or two more) in less general settings (studying small-scale problems or assuming server-side validation data). In contrast our methods are able to tune a wide range of hyperparameters in realistic federated networks. Some papers also discussed the challenges of finding good configurations while studying other aspects of federated training [41]. We argue that it is critical to properly address the challenges of federated hyperparameter optimization in practical settings, as we discuss in detail in Section 2.

Methodologically, our approach draws on the fact that local training-based methods such as FedAvg can be viewed as optimizing a surrogate objective for personalization [20], and more broadly leverages the similarity of the personalized FL setup and initialization-based meta-learning [6, 11, 17, 25]. While FedEx's formulation and guarantees use this relationship, the method itself is general-purpose

[1] A further challenge we do *not* address is that of the time-dependency of federated evaluation, c.f. [9].

and applicable to federated training of a single global model. Many recent papers address FL personalization more directly [13, 29, 35, 43, 47]. This connection and our use of NAS techniques also makes research connecting NAS and meta-learning relevant [10, 33], but unlike these methods we focus on tuning *non*-architectural parameters. In fact, we believe our work is the first to apply weight-sharing to regular hyperparameter search. Furthermore, meta-learning does not have the data-access and computational restrictions of FL, where such methods using the DARTS mixture relaxation [34] are less practical. Instead, `FedEx` employs the lower-overhead stochastic relaxation [8, 28], and its exponentiated update is similar to the recently proposed GAEA approach for NAS [27]. Running NAS itself in federated settings has also been studied [12, 15, 46]; while our focus is on non-architectural hyperparameters, in-principle our algorithms can also be used for federated NAS.

Theoretically, our work makes use of the average regret-upper-bound analysis (ARUBA) framework [20] to derive guarantees for learning the initialization, i.e. the global model, while simultaneously tuning the step-size of the local algorithm. The step-size of gradient-based algorithms has also been tuned on its own in the settings of data-driven algorithm design [14] and of statistical learning-to-learn [45].

2 Federated Hyperparameter Optimization

In this section we formalize the problem of hyperparameter optimization for FL and discuss the connection of its personalized variant to meta-learning. We also review `FedAvg` [36], a common federated optimization method, and present a reasonable baseline approach for tuning its hyperparameters.

Global and Personalized FL In FL we are concerned with optimizing over a network of heterogeneous clients $i = 1, \dots, n$, each with training, validation, and testing sets T_i, V_i, and E_i, respectively. We use $L_S(\mathbf{w})$ to denote the average loss over a dataset S of some $\mathbf{w}$-parameterized ML model, for $\mathbf{w} \in \mathbb{R}^d$ some real vector. For hyperparameter optimization, we assume a class of algorithms $\texttt{Alg}_a$ hyperparameterized by $a \in \mathcal{A}$ that use *federated access* to training sets T_i to output some element of $\mathbb{R}^d$. Here by "federated access" we mean that each iteration corresponds to a *communication round* at which $\texttt{Alg}_a$ has access to a batch of B clients[2] that can do local training and validation.

Specifically, we assume $\texttt{Alg}_a$ can be described by two subroutines with hyperparameters encoded by $b \in \mathcal{B}$ and $c \in \mathcal{C}$, so that $a = (b, c)$ and $\mathcal{A} = \mathcal{B} \times \mathcal{C}$. Here c encodes settings of a local training algorithm $\texttt{Loc}_c$ that take a training set S and initialization $\mathbf{w} \in \mathbb{R}^d$ as input and outputs a model $\texttt{Loc}_c(S, \mathbf{w}) \in \mathbb{R}^d$, while b sets those of an aggregation $\texttt{Agg}_b$ that takes the initialization $\mathbf{w}$ and outputs of $\texttt{Loc}_c$ as input and returns a model parameter. For example, in standard `FedAvg`, $\texttt{Loc}_c$ is T steps of gradient descent with step-size η and $\texttt{Agg}_b$ takes a weighted average of the outputs of $\texttt{Loc}_c$ across clients; here $c = (\eta, T)$ and $b = ()$. As detailed in the appendix, many FL methods can be decomposed this way, including well-known ones such as `FedAvg` [36], `FedProx` [31], `SCAFFOLD` [19], and `Reptile` [39] as well as more recent methods [1, 2, 29]. Our analysis and our proposed `FedEx` algorithm will thus apply to all of them, up to an assumption detailed next.

Starting from this decomposition, the global hyperparameter optimization problem can be written as

$$\min_{a \in \mathcal{A}} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Alg}_a(\{T_j\}_{j=1}^n)) \tag{1}$$

In many cases we are also interested in obtaining a device-specific local model, where we take a model trained on all clients and finetune it on each individual client before evaluating. A key assumption we make is that the finetuning algorithm will be the same as the local training algorithm $\texttt{Loc}_c$ used by $\texttt{Alg}_a$. This assumption can be justified by recent work in meta-learning that shows that algorithms that aggregate the outputs of local SGD can be viewed as optimizing for personalization using local SGD [20]. Then, in the personalized setting, the tuning objective becomes

$$\min_{a=(b,c) \in \mathcal{A}} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Loc}_c(T_i, \texttt{Alg}_a(\{T_j\}_{j=1}^n)) \tag{2}$$

Our approach will focus on the setting where the hyperparameters c of local training make up a significant portion of all hyperparameters $a = (b, c)$; by considering the personalization objective we will be able to treat such hyperparameters as architectural and thus apply weight-sharing.

Algorithm 1: Successive halving algorithm (SHA) applied to personalized FL. For the non-personalized objective (1), replace $L_{V_{ti}}(\mathbf{w}_i)$ by $L_{V_{ti}}(\mathbf{w}_a)$. For random search (RS) with N samples, set $\eta = N$ and $R = 1$.

Input: distribution $\mathcal{D}$ over hyperparameters $\mathcal{A}$, elimination rate $\eta \in \mathbb{N}$, elimination rounds $\tau_0 = 0, \tau_1, \ldots, \tau_R$

sample set of η^R hyperparameters $H \sim \mathcal{D}^{[\eta^R]}$
initialize a model $\mathbf{w}_a \in \mathbb{R}^d$ for each $a \in H$
for *elimination round* $r \in [R]$ **do**
  for *setting* $a = (b, c) \in H$ **do**
    for *comm. round* $t = \tau_{r-1} + 1, \ldots, \tau_r$ **do**
      for *client* $i = 1, \ldots, B$ **do**
        send $\mathbf{w}_a, c$ to client
        $\mathbf{w}_i \leftarrow \texttt{Loc}_c(T_{ti}, \mathbf{w}_a)$
        send $\mathbf{w}_i, L_{V_{ti}}(\mathbf{w}_i)$ to server
      $\mathbf{w}_a \leftarrow \texttt{Agg}_b(\mathbf{w}_a, \{\mathbf{w}_i\}_{i=1}^B)$
      $s_a \leftarrow \sum_{i=1}^B |V_{ti}| L_{V_{ti}}(\mathbf{w}_i) / \sum_{i=1}^B |V_{ti}|$
  $H \leftarrow \{a \in H : s_a \leq \frac{1}{\eta}\text{-quantile}(\{s_a : a \in H\})\}$

Output: remaining $a \in H$ and associated model $\mathbf{w}_a$

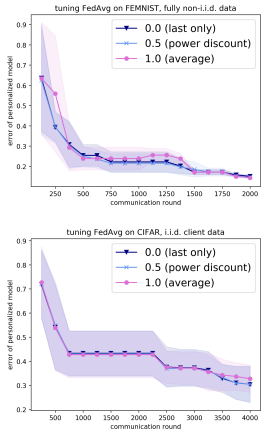

Figure 2: Tuning FL with SHA but making elimination decisions based on validation estimates using different discount factors. On both FEMNIST (top) and CIFAR (bottom) using more of the validation data does not improve upon just using the most recent round's validation error.

Tuning FL Methods: Challenges and Baselines In the non-federated setting, the objective (1) is amenable to regular hyperparameter optimization methods; for example, a random search approach would repeatedly sample a setting a from some distribution over $\mathcal{A}$, run $\texttt{Alg}_a$ to completion, and evaluate the objective, saving the best setting and output [3]. With a reasonable distribution and enough samples this is guaranteed to converge and can be accelerated using early stopping methods [26], in which $\texttt{Alg}_a$ is not always run to completion if the desired objective is poor at intermediate stages, or by adapting the sampling distribution using the results of previous objective evaluations [44]. As mentioned in the introduction, applying such methods to FL is inherently challenging due to

1. **Federated validation data:** Separating data across devices means we cannot immediately get a good estimate of the model's validation performance, as we only have access to a possibly small batch of devices at a time. This means that decisions such as which models to flag for early stopping will be noisy and may not fully incorporate all the available validation signal.
2. **Extreme resource limitations:** As FL algorithms can take a very long time to run in-practice due to the weakness and spotty availability of devices, we often cannot afford to conduct many training runs to evaluate different configurations. This issue is made more salient in cases where we use privacy techniques that only allow a limited number of accesses to the data of any individual user.
3. **Evaluating personalization:** While personalization is important in FL due to client heterogeneity, checking the performance of the current model on the personalization objective (2) is computationally intensive because computing may require running local training multiple times. In particular, while regular validation losses require computing one forward pass per data point, personalized losses require several forward-backward passes, making it many times more expensive if this loss is needed to make a tuning decision such as eliminating a configuration from consideration.

Despite these challenges, we can still devise sensible baselines for tuning hyperparameters in FL, most straightforward of which is to use a regular hyperparameter method but use validation data from a single round as a noisy surrogate for the full validation objective. Specifically, one can use random search (RS)—repeatedly evaluate random configurations—and a simple generalization called successive halving (SHA), in which we sample a set of configurations and partially run all of them for some number of communication rounds before eliminating all but the best $\frac{1}{\eta}$ fraction, repeating until only one configuration remains. Note both are equivalent to a "bracket" in Hyperband [26] and their adaptation to FL is detailed in Algorithm 1.

[2]For simplicity the number of clients per round is fixed, but all methods can be easily generalized to varying B.

As shown in Section 5, SHA performs reasonably well on the benchmarks we consider. However, by using validation data from one round it may make noisy elimination decisions, early-stopping potentially good configurations because of a difficult set of clients on a particular round. Here the problem is one of insufficient utilization of the validation data to estimate model performance. A reasonable approach to use more is to try some type of state-estimation: using the performance from previous rounds to improve the noisy measurement of the current one. For example, instead of using only the most recent round for elimination decisions we can use a weighted sum of the performances at all past rounds. To investigate this, we study a power decay weighting, where a round is discounted by some constant factor for each time step it is in the past. We consider factors 0.0 (taking the most recent performance only, as before), 0.5, and 1.0 (taking the average). However, in Figure 2 we show that incorporating more validation data this way than is used by Algorithm 1 by default does not significantly affect results.

Thus we may need a better algorithm to use more of the validation signal, most of which is discarded by using the most recent round's performance. We next propose `FedEx`, a new method that does so by using validation on each round to update a client hyperparameters distribution used to sample configurations to send to devices. Thus it alleviates issue (1) above by updating at each step, not waiting for an elimination round as in RS or SHA. By simultaneously training the model and tuning (client) hyperparameters, it also moves towards a fully single-shot procedure in which we only train once (we must still run multiple times due to server hyperparameters), which would solve issue (2). Finally, `FedEx` addresses issue (3) by using local training to both update the model and to estimate personalized validation loss, thus not spending extra computation on this more expensive objective.

3 Weight-Sharing for Federated Learning

We now present `FedEx`, a way to tune local FL hyperparameters. This section contains the general algorithm and its connection to weight-sharing; we instantiate it on several FL methods in Section 5.

Weight-Sharing for Architecture Search We first review the weight-sharing approach in NAS, which for a set $\mathcal{C}$ of network configurations is often posed as the bilevel optimization

$$\min_{c \in \mathcal{C}} \; L_{\text{valid}}(\mathbf{w}, c) \quad \text{s.t.} \quad \mathbf{w} \in \arg\min_{\mathbf{u} \in \mathbb{R}^d} \; L_{\text{train}}(\mathbf{u}, c) \tag{3}$$

where $L_{\text{train}}, L_{\text{valid}}$ evaluate a single configuration with the given weights. If, as in NAS, all hyperparameters are architectural, then they are effectively themselves trainable model parameters [27], so we could instead consider solving the following "single-level" empirical risk minimization (ERM):

$$\min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \; L(\mathbf{w}, c) \quad = \quad \min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \; L_{\text{train}}(\mathbf{w}, c) + L_{\text{valid}}(\mathbf{w}, c) \tag{4}$$

Solving this instead of the bilevel problem (3) has been proposed in several recent papers [24, 27].

Early approaches to solving either formulation of NAS were costly due to the need for full or partial training of many architectures in a very large search space. The weight-sharing paradigm [40] reduces the problem to that of training a single architecture, a "supernet" containing all architectures in the search space $\mathcal{C}$. A straightforward way of constructing a supernet is via a "stochastic relaxation" where the loss is an expectation w.r.t. sampling c from some distribution over $\mathcal{C}$ [8]. Then the shared weights can be updated using SGD by first sampling an architecture c and using an unbiased estimate of $\nabla_{\mathbf{w}} L(\mathbf{w}, c)$ to update $\mathbf{w}$. The distribution over $\mathcal{C}$ may itself be adapted or stay fixed. We focus on the former case, adapting some θ-parameterized distribution $\mathcal{D}_\theta$; this yields the stochastic relaxation objective

$$\min_{\theta \in \Theta, \mathbf{w} \in \mathbb{R}^d} \; \mathbb{E}_{c \sim \mathcal{D}_\theta} L(\mathbf{w}, c) \tag{5}$$

Since architectural hyperparameters are often discrete decisions, e.g. a choice of which of a fixed number of operations to use, a natural choice of $\mathcal{D}_\theta$ is as a product of categorical distributions over simplices. In this case, any discretization of an optimum θ of the relaxed objective (5) whose support is in the support of θ will be an optimum of the original objective (4). A natural update scheme here is exponentiated gradient [22], where each successive θ is proportional to $\theta \odot \exp(-\eta \tilde{\nabla})$, η is a step-size, and $\tilde{\nabla}$ an unbiased estimate of $\nabla_\theta \mathbb{E}_{c \sim \mathcal{D}_\theta} L(\mathbf{w}, c)$ that can be computed using the re-parameterization trick [42]. By alternating this exponentiated update with the standard SGD update to $\mathbf{w}$ discussed earlier we obtain a simple block-stochastic minimization scheme that is guaranteed to converge, under certain conditions, to the ERM objective, and also performs well in practice [27].

The FedEx Method To obtain FedEx from weight-sharing we restrict to the case of tuning only the hyperparameters c of local training $\texttt{Loc}_c$.[3] Our goal then is just to find the best initialization $\mathbf{w} \in \mathbb{R}^d$ and local hyperparameters $c \in \mathcal{C}$, i.e. we replace the personalized objective (2) by

$$\min_{c \in \mathcal{C}, \mathbf{w} \in \mathbb{R}^d} \quad \sum_{i=1}^{n} |V_i| L_{V_i}(\texttt{Loc}_c(T_i, \mathbf{w})) \tag{6}$$

Note $\texttt{Alg}_a$ outputs an element of $\mathbb{R}^d$, so this new objective is upper-bounded by the original (2), i.e. any solution will be at least as good for the original objective. Note also that for fixed c this is equivalent to the classic train-validation split objective for meta-learning with $\texttt{Loc}_c$ as the base-learner. More importantly for us, it is also in the form of the r.h.s. of the weight-sharing objective (4), i.e. it is a single-level function of $\mathbf{w}$ and c. We thus apply a NAS-like stochastic relaxation:

$$\min_{\theta \in \Theta, \mathbf{w} \in \mathbb{R}^d} \quad \sum_{i=1}^{n} |V_i| \mathbb{E}_{c \in \mathcal{D}_\theta} L_{V_i}(\texttt{Loc}_c(T_i, \mathbf{w})) \tag{7}$$

In NAS we would now set the distribution to be a product of categorical distributions over different architectures, thus making θ an element of a product of simplices and making the optimum of the original objective (6) equivalent to the optimum of the relaxed objective (7) as an extreme point of the simplex. Unlike in NAS, FL hyperparameters such as the learning rate are not extreme points of a simplex and so it is less clear what parameterized distribution $\mathcal{D}_\theta$ to use. Nevertheless, we find that crudely imposing a categorical distribution over $k > 1$ random samples from some distribution (e.g. uniform) over $\mathcal{C}$ and updating θ using exponentiated gradient over the resulting k-simplex works well. We alternate this with updating $\mathbf{w} \in \mathbb{R}^d$, which in a NAS algorithm involves an SGD update using an unbiased estimate of the gradient at the current $\mathbf{w}$ and θ.

We call this alternating method for solving (7) FedEx and describe it for a general $\texttt{Alg}_a$ consisting of sub-routines $\texttt{Agg}_b$ and $\texttt{Loc}_c$ in Algorithm 2; recall from Section 2 that many FL methods can be decomposed this way, so our approach is widely applicable. FedEx has a minimal overhead, consisting only of the last four lines of the outer loop updating θ. Thus, as with weight-sharing, FedEx can be viewed as reducing the complexity of tuning local hyperparameters to that of training a single model. Each update to θ requires a step-size η_t and an approximation $\tilde{\nabla}$ of the gradient w.r.t. θ; for the latter we obtain an estimate $\tilde{\nabla}_j$ of each gradient entry via the reparameterization trick, whose variance we reduce by subtracting a baseline λ_t. How we set η_t and λ_t is detailed in the Appendix.

To see how FedEx is approximately optimizing the relaxed objective (7), we can consider the case where $\texttt{Alg}_a$ is Reptile [39], which was designed to optimize some approximation of (6) for fixed c, or equivalently the relaxed objective for an atomic distribution $\mathcal{D}_\theta$. The theoretical literature on meta-learning [20, 21] shows that Reptile can be interpreted as optimizing a surrogate objective minimizing the squared distance between $\mathbf{w}$ and the optimum of each task i, with the latter being replaced by the last iterate in practice. It is also shown that the surrogate objective is useful for personalization in the online convex setting.[4] As opposed to this past work, FedEx makes two gradient updates in the outer loop, on two disjoint sets of variables: the first is the sub-routine $\texttt{Agg}_b$ of $\texttt{Alg}_a$ that aggregates the outputs of local training and is using the gradient of the surrogate objective, since the derivative of the squared distance is the difference between the initialization $\mathbf{w}$ and the parameter at the last iterate of $\texttt{Loc}_c$; the second is the exponentiated gradient update that is directly using an unbiased estimate of the derivative of the second objective w.r.t. the distribution parameters θ. Thus, roughly speaking FedEx runs simultaneous stochastic gradient descent on the relaxed objective (7), although for the variables $\mathbf{w}$ we are using a first-order surrogate. In the theoretical portion of this work we employ this interpretation to show the approach works for tuning the step-size of online gradient descent in the online convex optimizations setting.

Wrapping FedEx We can view FedEx as an algorithm of the form tuned by Algorithm 1 that implements federated training of a supernet parameter $(\mathbf{w}, \theta)$, with the local training routine Loc including a step for sampling $c \sim \mathcal{D}_\theta$ and the server aggregation routine including an exponentiated update of θ. Thus we can wrap FedEx in Algorithm 1, which we find useful for a variety of reasons:

- The wrapper can tune the settings of b for the aggregation step $\texttt{Agg}_b$, which FedEx cannot.
- FedEx itself has a few hyperparameters, e.g. how to set the baseline λ_t, which can be tuned.

[3] We will use some wrapper algorithm to tune the hyperparameters b of $\texttt{Agg}_b$.

[4] Formally they study a sequence of upper bounds and not a surrogate objective, as their focus is online learning.

Algorithm 2: `FedEx`

Input: configurations $c_1, \dots, c_k \in \mathcal{C}$, setting b for $\texttt{Agg}_b$, schemes for setting step-size η_t and baseline λ_t, total number of steps $\tau \geq 1$

initialize $\theta_1 = \mathbf{1}_k/k$ and shared weights $\mathbf{w}_1 \in \mathbb{R}^d$

for *comm. round* $t = 1, \dots, \tau$ **do**

  for *client* $i = 1, \dots, B$ **do**

    send $\mathbf{w}_t, \theta_t$ to client

    sample $c_{ti} \sim \mathcal{D}_{\theta_t}$

    $\mathbf{w}_{ti} \leftarrow \texttt{Loc}_{c_{ti}}(T_{ti}, \mathbf{w}_t)$

    send $\mathbf{w}_{ti}, c_{ti}, L_{V_{ti}}(\mathbf{w}_{ti})$ to server

  $\mathbf{w}_{t+1} \leftarrow \texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_{ti}\}_{i=1}^B)$

  $\tilde{\nabla}_j \leftarrow \frac{\sum_{i=1}^B |V_{ti}|(L_{V_{ti}}(\mathbf{w}_{ti}) - \lambda_t)\mathbf{1}_{c_{ti}=c_j}}{\theta_{t\,[j]} \sum_{i=1}^B |V_{ti}|} \; \forall \; j$

  $\theta_{t+1} \leftarrow \theta_t \odot \exp(-\eta_t \tilde{\nabla})$

  $\theta_{t+1} \leftarrow \theta_{t+1}/\|\theta_{t+1}\|_1$

Output: model $\mathbf{w}$, hyperparameter distribution θ

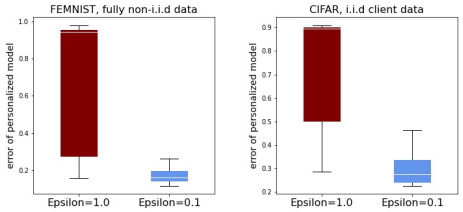

Figure 3: Comparison of the range of performance values attained using different perturbation settings. Although the range is much smaller for $\epsilon = 0.1$ than for $\epsilon = 1.0$ (the latter is the entire space), it still covers a large (roughly 10-20%) range of different performance levels on both FEMNIST (left) and CIFAR (right).

- By running multiple seeds and potentially using early stopping, we can run `FedEx` using more aggressive steps-sizes and the wrapper will discard cases where this leads to poor results.
- We can directly compare `FedEx` to a regular hyperparameter optimization scheme run over the original algorithm, e.g. `FedAvg`, by using the same scheme to both wrap `FedEx` and tune `FedAvg`.
- Using the wrapper allows us to determine the configurations $c_1, \dots, c_k$ given to Algorithm 2 using a local perturbation scheme (detailed next) while still exploring the entire hyperparameter space.

Local Perturbation It remains to specify how to select the configurations $c_1, \dots, c_k \in \mathcal{C}$ to pass to Algorithm 2. While the simplest approach is to draw from $\text{Unif}^k(\mathcal{C})$, we find that this leads to unstable behavior if the configurations are too distinct from each other. To interpolate between sampling c_i independently and setting them to be identical (which would just be equivalent to the baseline algorithm), we use a simple *local perturbation* method in which c_1 is sampled from $\text{Unif}(\mathcal{C})$ and $c_2, \dots, c_k$ are sampled uniformly from a local neighborhood of $\mathcal{C}$. For continuous hyperparameters (e.g. step-size, dropout) drawn from an interval $[a, b] \subset \mathbb{R}$ the local neighborhood is $[c \pm (b-a)\varepsilon]$ for some $\varepsilon \geq 0$, i.e. a scaled ε-ball; for discrete hyperparameters (e.g. batch-size, epochs) drawn from a set $\{a, \dots, b\} \subset \mathbb{Z}$, the local neighborhood is similarly $\{c - \lfloor (b-a)\varepsilon \rfloor, \dots, c + \lceil (b-a)\varepsilon \rceil\}$; in our experiments we set $\varepsilon = 0.1$, which works well, but run ablation studies varying these values in the appendix showing that a wide range of them leads to improvement. Note that while local perturbation does limit the size of the search space explored by each instance of `FedEx`, as shown in Figure 3 the difference in performance between different configurations in the same ball is still substantial.

Limitations of `FedEx` While `FedEx` is applicable to many important FL algorithms, those that cannot be decomposed into local fine-tuning and aggregation should instead be tuned by one of our baselines, e.g. SHA. `FedEx` is also limited in that it is forced to rely on such algorithms as wrappers for tuning its own hyperparameters and certain FL hyperparameters such as server learning rate.

4 Theoretical Analysis for Tuning the Step-Size in an Online Setting

As noted in Section 3, `FedEx` can be viewed as alternating minimization, with a gradient step on a surrogate personalization loss and an exponentiated gradient update of the configuration distribution θ. We make this formal and prove guarantees for a simple variant of `FedEx` in the setting where the server has one client per round, to which the server sends an initialization to solve an online convex optimization (OCO) problem using online gradient descent (OGD) on a sequence of m adversarial convex losses (i.e. one SGD epoch in the stochastic case). Note we use "client" and "task" interchangeably, as the goal is a meta-learning (personalization) result. The performance measure here is *task-averaged regret*, which takes the average over τ clients of the regret they incur on its loss:

$$\bar{\mathbf{R}}_\tau = \frac{1}{\tau}\sum_{t=1}^{\tau}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{w}_{t,i}) - \ell_{t,i}(\mathbf{w}_t^*) \tag{8}$$

Here $\ell_{t,i}$ is the ith loss of client t, $\mathbf{w}_{t,i}$ the parameter chosen on its ith round from a compact parameter space $\mathcal{W}$, and $\mathbf{w}_t^* \in \arg\min_{\mathbf{w}\in\mathcal{W}}\sum_{i=1}^{m}\ell_{t,i}(\mathbf{w})$ the task optimum. In this setting, the Average Regret-Upper-Bound Analysis (ARUBA) framework [20] can be used to show guarantees for a `Reptile` (i.e. `FedEx` with a server step-size) variant in which at each round the initialization is updated as $\mathbf{w}_{t+1} \leftarrow (1-\alpha_t)\mathbf{w}_t + \alpha_t\mathbf{w}_t^*$ for server step-size $\alpha_t = 1/t$. Observe that the only difference between this update and `FedEx`'s is that the task-t optimum $\mathbf{w}_t^*$ is used rather than the last iterate of OGD on that task. Specifically they bound task-averaged regret by

$$\bar{\mathbf{R}}_\tau \le \tilde{\mathcal{O}}\left(\frac{1}{\sqrt[4]{\tau}} + V\right)\sqrt{m} \quad \text{for} \quad V^2 = \min_{\mathbf{w}\in\mathcal{W}}\frac{1}{\tau}\sum_{t=1}^{\tau}\|\mathbf{w}-\mathbf{w}_t^*\|_2^2 \tag{9}$$

Here V—the average deviation of the optimal actions $\mathbf{w}_t^*$ across tasks—is a measure of *task-similarity*: V is small when the tasks (clients) have similar data and thus can be solved by similar parameters in $\mathcal{W}$ but large when their data is different and so the optimum parameters to use are very different. Thus the bound in (9) shows that as the server (meta-learning) sees more and more clients (tasks), their regret on each decays with rate $1/\sqrt[4]{\tau}$ to depend only on the task-similarity, which is hopefully small if the client data is similar enough that transfer learning makes sense, in particular if $V \ll \operatorname{diam}(\mathcal{W})$. Since single-task regret has lower bound $\Omega(D\sqrt{m})$, achieving asymptotic regret $V\sqrt{m}$ thus demonstrates successful learning of a useful initialization in $\mathcal{W}$ that can be used for personalization. Note that such bounds can also be converted to obtain guarantees in the statistical meta-learning setting as well [20].

A drawback of past results using the ARUBA framework is that they either assume the task-similarity V is known in order to set the client step-size [25] or they employ an OCO method to learn the local step-size that cannot be applied to other potential algorithmic hyperparameters [20]. In contrast, we prove results for using bandit exponentiated gradient to tune the client step-size, which is precisely the `FedEx` update. In particular, Theorem 4.1 shows that by using a discretization of potential client step-sizes as the configurations in Algorithms 2 we can obtain the following task-averaged regret:

Theorem 4.1. *Let $\mathcal{W} \subset \mathbb{R}^d$ be convex and compact with diameter $D = \operatorname{diam}(\mathcal{W})$ and let $\ell_{t,i}$ be a sequence of $m\tau$ b-bounded convex losses—m for each of τ tasks—with Lipschitz constant $\le G$. We assume that the adversary is oblivious within-task. Suppose we run Algorithm 2 with $B = 1$, configurations $c_j = \frac{D}{Gj\sqrt{m}}$ for each $j = 1, \dots, k$ determining the local step-size of single-epoch SGD (OGD), $\mathbf{w}_{ti} = \mathbf{w}_t^*$, regret $\sum_{i=1}^{m}\ell_{t,i}(\mathbf{w}_{t,i}) - \ell_{t,i}(\mathbf{w}_t)$ used in place of $L_{V_{ti}}(\mathbf{w}_{ti})$, and $\lambda_t = 0\ \forall\, t \in [\tau]$. Then if $\eta_t = \frac{1}{mb}\sqrt{\frac{\log k}{k\tau}}\ \forall\, t \in [\tau]$, $k^{\frac{3}{2}} = \frac{DG}{b}\sqrt{\frac{\tau}{2m}}$, and $\texttt{Agg}_b(\mathbf{w}, \mathbf{w}_t^*) = (1-\alpha_t)\mathbf{w} + \alpha_t\mathbf{w}_t^*$ for $\alpha_t = 1/t\ \forall\, t \in [\tau]$ we have (taking expectations over sampling from D_{θ_t})*

$$\mathbb{E}\bar{\mathbf{R}}_\tau \le \tilde{\mathcal{O}}\left(\sqrt[3]{m/\tau} + V\right)\sqrt{m} \tag{10}$$

The proof of this result, given in the supplement, follows the ARUBA framework of using meta OCO algorithm to optimize the initialization-dependent upper bound on the regret of OGD; in addition we bound errors to the bandit setting and discretization of the step-sizes. Theorem 4.1 demonstrates that `FedEx` is a sensible algorithm for tuning the step-size in the meta-learning setting where each task is an OCO problem, with the average regret across tasks (clients) converging to depend only on the task-similarity V, which we hope is small in the setting where personalization is useful. As we can see by comparing to the bound in (9), besides holding for a more generally-applicable algorithm our bound also improves the dependence on τ, albeit at the cost of an additional $m^{\frac{1}{3}}$ factor. Note that that the sublinear term can be replaced by $1/\sqrt{\tau}$ in the full-information setting, i.e. where required the client to try SGD with each configuration c_j at each round to obtain regret for all of them.

Table 1: Final test error obtained when tuning using a standard hyperparameter tuning algorithm (SHA or RS) alone, or when using it for server (aggregation) hyperparameters while FedEx tunes client (on-device training) hyperparameters. The target model is the one used to compute on-device validation error by the wrapper method, as well as the one used to compute test error after tuning. Note that this table reports the final error results corresponding to the online evaluations reported in Figure 4, which measure performance as more of the computational budget is expended.

Wrapper method	Target model	Tuning method	Shakespeare i.i.d.	Shakespeare non-i.i.d.	FEMNIST i.i.d.	FEMNIST non-i.i.d.	CIFAR-10 i.i.d.
Random Search (RS)	global	RS (server & client)	60.32 ± 10.03	64.36 ± 14.19	22.81 ± 4.56	22.98 ± 3.41	30.46 ± 9.44
		+ FedEx (client)	53.94 ± 9.13	57.70 ± 17.57	20.96 ± 4.77	22.30 ± 3.66	34.83 ± 14.74
	personalized	RS (server & client)	61.10 ± 9.32	61.71 ± 9.08	17.45 ± 2.82	17.77 ± 2.63	34.89 ± 10.56
		+ FedEx (client)	54.90 ± 9.97	56.48 ± 13.60	16.31 ± 3.77	15.93 ± 3.06	39.13 ± 15.13
Successive Halving (SHA)	global	SHA (server & client)	47.38 ± 3.40	46.79 ± 3.51	18.64 ± 1.68	20.30 ± 1.66	21.62 ± 2.51
		+ FedEx (client)	$\mathbf{44.52} \pm 1.68$	$\mathbf{45.24} \pm 3.31$	19.22 ± 2.05	19.43 ± 1.45	$\mathbf{20.82} \pm 1.37$
	personalized	SHA (server & client)	46.77 ± 3.61	48.04 ± 3.72	$\mathbf{14.79} \pm 1.55$	14.78 ± 1.31	24.81 ± 6.13
		+ FedEx (client)	46.08 ± 2.57	45.89 ± 3.76	14.97 ± 1.31	$\mathbf{14.76} \pm 1.70$	21.77 ± 2.83

5 Empirical Results

In our experiments, we instantiate FedEx on the problem of tuning FedAvg, FedProx, and Reptile; the first is the most popular algorithm for federated training, the second is an extension designed for heterogeneous devices, and the last is a compatible meta-learning method used for learning initializations for personalization. At communication round t these algorithms use the aggregation

$$\texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_i\}_{i=1}^B) = (1-\alpha_t)\mathbf{w} + \frac{\alpha_t}{\sum_{i=1}^B |T_{ti}|} \sum_{i=1}^B |T_{ti}| \mathbf{w}_i \tag{11}$$

for some learning rate $\alpha_t > 0$ that can vary through time; in the case of FedAvg we have $\alpha_t = 1 \; \forall \, t$. The local training sub-routine $\texttt{Loc}_c$ is SGD with hyperparameters c over some objective defined by the training data T_{ti}, which can also depend on c. For example, to include FedProx we include in c an additional local hyperparameter for the proximal term compared with that of FedAvg.

We tune several hyperparameters of both aggregation and local training; for the former we tune the server learning rate schedule and momentum, found to be helpful for personalization [17]; for the latter we tune the learning rate, momentum, weight-decay, the number of local epochs, the batch-size, dropout, and proximal regularization. Please see the supplementary material for the exact hyperparameter space considered. While we mainly evaluate FedEx in cross-device federated settings, which is generally more difficult than cross-silo in terms of hyperparameter optimization, FedEx can be naturally applied to cross-silo settings, where the challenges of heterogeneity, privacy requirements, and personalization remain.

Because our baseline is running Algorithm 1, a standard hyperparameter tuning algorithm, to tune all hyperparameters, and because we need to also wrap FedEx in such an algorithm for the reasons described in Section 3, our empirical results will test the following question: does FedEx, wrapped by random search (RS) or a successive halving algorithm (SHA), do better than RS or SHA run with the same settings directly? Here "better" will mean both the final test accuracy obtained and the online evaluation setting, which tests how well hyperparameter optimization is doing at intermediate phases. Furthermore, we also investigate whether FedEx can improve upon the wrapper alone even when targeting a good *global* and not personalized model, i.e. when elimination decisions are made using the average global validation loss. We run Algorithm 1 on the personalized objective and use RS and SHA with elimination rate $\eta = 3$, the latter following Hyperband [26]. To both wrappers we allocate the same (problem-dependent) tuning budget. To obtain the elimination rounds in Algorithm 1 for SHA, we set the number of eliminations to $R = 3$, fix a total communication round budget, and fix a maximum number of rounds to be allocated to any configuration a; as detailed in the Appendix, this allows us to determine $T_1, \ldots, T_R$ so as to use up as much of the budget as possible.

We evaluate the performance of FedEx on three datasets (Shakespeare, FEMNIST, and CIFAR-10) on both vision and language tasks. We consider the following two different partitions of data:

1. Each device holds i.i.d. data. While overall data across the entire network can be non-i.i.d., we randomly shuffle local data *within* each device before splitting into train, validation, and test sets.

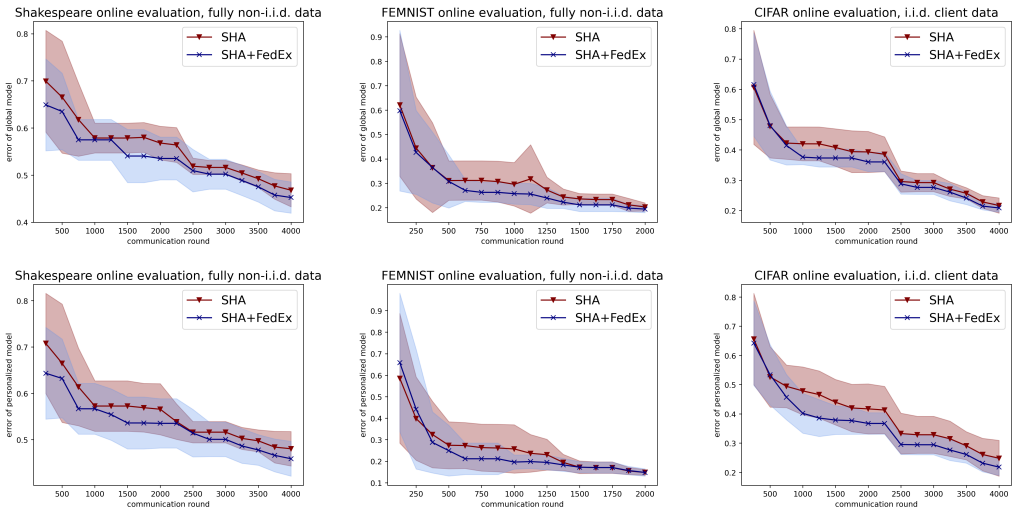

Figure 4: Online evaluation of FedEx on the Shakespeare next-character prediction dataset (left), the FEMNIST image classification dataset (middle), and the CIFAR-10 image classification dataset (right) in the fully non-i.i.d. setting (except CIFAR-10). We report global model performance on the top and personalized performance on the bottom. All evaluations are run for three trials.

2. Each device holds non-i.i.d. data. In Shakespeare, each device is an actor and the local data is split according to the temporal position in the play; in FEMNIST, each device is the digit writer and the local data is split randomly; in CIFAR-10, we do not consider a non-i.i.d. setting.

For Shakespeare and FEMNIST we use 80% of the data for training and 10% each for validation and testing. In CIFAR-10 we hold out 10K examples from the usual training/testing split for validation. The backbone models used for Shakespeare and CIFAR-10 follow from the FedAvg evaluation [36] and use 4K communications rounds (at most 800 round for each arm), while that of FEMNIST follows from LEAF [5] and uses 2K communication rounds (at most 200 for each arm).

Table 1 presents our main results, displaying the final test error of the target model after tuning using either a wrapper algorithm alone or its combination with FedEx. The evaluation shows that using FedEx on the client parameters is either equally or more effective in most cases; in particular, a FedEx-modified method performs best everywhere except i.i.d. FEMNIST, where it is very close. Furthermore, FedEx frequently improves upon the wrapper algorithm by 2 or more percentage points.

We further present online evaluation results in Figure 4, where we display the test error of FedEx wrapped with SHA compared to SHA alone as a function of communication rounds. Here we see that for most of training FedEx is either around the same or better then the alternative, except at the beginning; the former is to be expected since the randomness of FedEx leads to less certain updates at initialization. Nevertheless FedEx is usually better than the SHA baseline by the halfway point.

6 Conclusion

In this paper we study the problem of hyperparameter optimization in FL, starting with identifying the key challenges and proposing reasonable baselines that adapts standard approaches to the federated setting. We further make a novel connection to the weight-sharing paradigm from NAS—to our knowledge the first instance of this being used for regular (non-architectural) hyperparameters—and use it to introduce FedEx. This simple, low-overhead algorithm for accelerating the tuning of hyperparameters in federated learning can be theoretically shown to successfully tune the step-size for multi-task OCO problems and effectively tunes FedAvg, FedProx, and Reptile on standard benchmarks. The scope of application of FedEx is very broad, including tuning actual architectural hyperparameters rather than just settings of local SGD, i.e. doing federated NAS, and tuning initialization-based meta-learning algorithms such as Reptile and MAML. Lastly, any work on FL comes with privacy and fairness risks due its frequent use of sensitive data; thus any application of our work must consider tools being developed by the community for mitigating such issues [32, 37].

Acknowledgments

This material is based on work supported by the National Science Foundation under grants CCF-1535967, CCF-1910321, IIS-1618714, IIS-1901403, SES-1919453, IIS-1705121, IIS-1838017, IIS-2046613 and IIS-2112471; the Defense Advanced Research Projects Agency under cooperative agreements HR00112020003 and FA875017C0141; an AWS Machine Learning Research Award; an Amazon Research Award; a Bloomberg Research Grant; a Microsoft Research Faculty Fellowship; an Amazon Web Services Award; a Facebook Faculty Research Award; funding from Booz Allen Hamilton Inc.; a Block Center Grant; and a Two Sigma Fellowship Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of these funding agencies.

References

[1] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[2] Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. Federated learning via posterior averaging: A new perspective and practical algorithms. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[5] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. arXiv, 2018.

[6] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. arXiv, 2018.

[7] Zhongxiang Dai, Kian Hsiang Low, and Patrick Jaillet. Federated bayesian optimization via thompson sampling. In *Advances in Neural Information Processing Systems*, 2020.

[8] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[9] Hubert Eichner, Tomer Koren, H. Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[10] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. arXiv, 2019.

[11] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. In *Advances in Neural Information Processing Systems*, 2020.

[12] Anubhav Garg, Amit Kumar Saha, and Debo Dutta. Direct federated neural architecture search. arXiv, 2020.

[13] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. arXiv, 2020.

[14] Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[15] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Towards non-i.i.d. and invisible data with FedNAS: Federated deep learning via neural architecture search. arXiv, 2020.

[16] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, 2011.

[17] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. arXiv, 2019.

[18] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. arXiv, 2019.

[19] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[20] Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Adaptive gradient-based meta-learning methods. In *Advances in Neural Information Processing Systems*, 2019.

[21] Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Provable guarantees for gradient-based meta-learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[22] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132:1–63, 1997.

[23] Antti Koskela and Antti Honkela. Learning rate adaptation for federated and differentially private learning. arXiv, 2018.

[24] Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, and Tong Zhang. StacNAS: Towards stable and consistent differentiable neural architecture search. arXiv, 2019.

[25] Jeffrey Li, Mikhail Khodak, Sebastian Caldas, and Ameet Talwalkar. Differentially private meta-learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[26] Liam Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[27] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[28] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2019.

[29] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. arXiv, 2020.

[30] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37, 2020.

[31] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of the Conference on Machine Learning and Systems*, 2020.

[32] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *Proceedings of the 6th International Conference on Learning Representations*, 2020.

[33] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta-learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[35] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. arXiv, 2020.

[36] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artifical Intelligence and Statistics*, 2017.

[37] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[38] Hesham Mostafa. Robust federated learning through representation matching and adaptive hyper-parameters. arXiv, 2019.

[39] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. arXiv, 2018.

[40] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[41] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[42] Reuven Y. Rubinstein and Alexander Shapiro. *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*. John Wiley & Sons, Inc., 1993.

[43] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, 2017.

[44] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

[45] Xiang Wang, Shuai Yuan, Chenwei Wu, and Rong Ge. Guarantees for tuning the step size using a learning-to-learn approach. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[46] Mengwei Xu, Yuxin Zhao, Kaigui Bian, Gang Huang, Qiaozhu Mei, and Xuanzhe Liu. Federated neural architecture search. arXiv, 2020.

[47] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. arXiv, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See end of Section 3.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See end of Section 6.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4 and the Appendix.

(b) Did you include complete proofs of all theoretical results? [Yes] See the Appendix.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See link in the Appendix.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the Appendix.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 1 and Figure 4.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] The main unit of cost in our setting is communication round, which we do report in e.g. Figure 4.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] see link in the Appendix.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 4.1

Proof. Let $\gamma_t \sim \mathcal{D}_{\theta_t}$ be the step-size chosen at time t. Then we have that

$$\begin{aligned}
\tau \mathbb{E}\bar{\mathbf{R}}_\tau &= \sum_{t=1}^{\tau} \mathbb{E}_{\gamma_t} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,\gamma_t)}\right) - \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&= \sum_{t=1}^{\tau} \sum_{j=1}^{k} \theta_{t[j]} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,c_j)}\right) - \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&\leq \frac{\log k}{\eta} + \eta k \tau m^2 b^2 \\
&\quad + \min_{j\in[k]} \sum_{t=1}^{\tau} \sum_{i=1}^{m} \ell_{t,i}\left(\mathbf{w}_{t,i}^{(\mathbf{w}_t,c_j)}\right) - \min_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{m} \ell_{t,i}(\mathbf{w}_t^*) \\
&\leq 2mb\sqrt{\tau k \log k} + \min_{j\in[k]} \sum_{t=1}^{\tau} \frac{1}{2c_j}\|\mathbf{w}_t - \mathbf{w}_t^*\|_2^2 + c_j m G^2 \\
&\leq 2mb\sqrt{\tau k \log k} + \min_{j\in[k]} \frac{D^2(1+\log\tau)}{2c_j} + \left(\frac{V^2}{2c_j} + c_j m G^2\right)\tau \\
&\leq 2mb\sqrt{\tau k \log k} + \frac{D^2(1+\log\tau)+V^2\tau}{2\gamma^*} + \gamma^* m G^2 \tau \\
&\quad + \min_{j\in[k]} \left(\frac{1}{c_j} - \frac{1}{\gamma^*}\right)\frac{D^2(1+\log\tau)+V^2\tau}{2} + (c_j - \gamma^*) m G^2 \tau \\
&\leq 2mb\sqrt{\tau k \log k} + 4D\sqrt{\frac{\tau+\tau\log\tau}{2}} + \left(2V + \frac{D}{k}\right) G\tau\sqrt{\frac{m}{2}} \\
&= mb\sqrt{2\tau\log\tau} + 4D\sqrt{\frac{\tau+\tau\log\tau}{2}} + (DG + 2GV\tau)\sqrt{\frac{m}{2}}
\end{aligned}$$

where the second line uses linearity of expectations over $\gamma_t \sim \mathcal{D}_{\theta_t}$, the third substitutes the bandit regret of EG [**?** , Corollary 4.2], the fourth substitutes $\eta = \frac{1}{mb}\sqrt{\frac{\log k}{\tau k}}$ and the regret of OGD [**?** , Corollary 2.7], the fifth substitutes the regret guarantee of Adaptive OGD over functions $\frac{1}{2}\|\mathbf{w}_t - \mathbf{w}_t^*\|_2^2$ [**?** , Theorem 2.1] with step-size $\alpha_t = 1/t$ and the definition of V, the sixth substitutes the best discretized step-size c_j for the optimal $\gamma^* \in \left(0, \frac{D}{G\sqrt{2m}}\right]$, and the seventh substitutes $\frac{V}{2G\sqrt{2m}} + \frac{D}{2G}\sqrt{\frac{1+\log\tau}{2m\tau}}$ for γ^* and $\arg\min_{j:c_j\geq\gamma^*}$ for $\arg\min_j c_j$. Setting $k^{\frac{3}{2}} = \frac{DG}{b}\sqrt{\frac{\tau}{2m}}$ and dividing both sides by τ yields the result. □

B Decomposing Federated Optimization Methods

As detailed in Section 2 our analysis and use of FedEx to tune local training hyperparameters depends on a formulation that decomposes FL methods into two subroutines: a local training routine $\texttt{Loc}_c(S, \mathbf{w})$ with hyperparameters c over data S and starting from initialization $\mathbf{w}$ and an aggregation routine $\texttt{Agg}_b$ with hyperparameters b. In this section we discuss how a variety of federated optimization methods, including several of the best-known, can be decomposed in this manner. This enables the application of FedEx to tune their hyperparameters.

B.1 FedAvg [36]

The best-known FL method, FedAvg runs SGD on each client in a batch starting from a shared initialization and then updates to the average of the last iterate of the clients, often weighted by the number of data points each client has. The decomposition here is:

- $\texttt{Loc}_c$ Local SGD (or another gradient-based algorithm, e.g. Adam [**?**]), with c being the standard hyperparameters such as step-size, momentum, weight-decay, etc.
- $\texttt{Agg}_b$ Weighted averaging, with no hyperparameters in b.

B.2 FedProx [31]

FedProx has the same decomposition as FedAvg except local SGD is replaced by a proximal version that regularizes the routine to be closer to the initialization, adding another hyperparameter to c governing the strength of this regularization.

B.3 Reptile [39]

A well-known meta-learning algorithm, Reptile has the same decomposition as FedAvg except the averaged aggregation is replaced by a convex combination of the initialization and the average of the last iterates, as in Equation 11. This adds another hyperparameter to b governing the tradeoff between the two.

B.4 SCAFFOLD [19]

SCAFFOLD comes in two variants, both of which compute and aggregate control variates in parallel to the model weights. The decomposition here is:

- $\texttt{Loc}_c$ Local SGD starting from a weight initialization with a control variate, which can be merged to form the local training initialization. The hyperparameters in c are the same as in FedAvg.
- $\texttt{Agg}_b$ Weighted averaging of both the initialization and the control variates, with the same hyperparameters as Reptile.

B.5 FedDyn [1]

In addition to a FedAvg/FedProx/Reptile-like training routine, this algorithm maintains a regularizer on each device that affects the local training routine. While this statefulness cannot strictly be subsumed in our decomposition, since it does not introduce any additional hyperparameters the remaining hyperparameters can be tuned in the same manner as we do for FedAvg/FedProx/Reptile. In order to choose between using FedDyn or not, one can introduce a binary hyperparameter to c specifying whether or not $\texttt{Loc}_c$ uses that term in the objective it optimizes or not, allowing it also to be tuned via FedEx.

B.6 FedPA [2]

This algorithm replaces local SGD in FedAvg by a local Markov-chain Monte Carlo (MCMC) routine starting from the initialization given by aggregating the previous round's MCMC routines. The decomposition is then just a replacement of local SGD and its associated hyperparameters by local MCMC and its hyperparameters, with the aggregation routine remaining the same.

B.7 Ditto [29]

Although it depends on what solver is used for the local solver and aggregation routines, in the simplest formulation, the local optimization of personalized models involves an additional regularization hyperparameter. While the updating rule of Ditto is different from that of FedProx, the hyperparameters can be decomposed and tuned in a similar manner.

B.8 MAML [?]

A well-known meta-learning algorithm, MAML takes one or more full-batch gradient descent (GD) steps locally and updates the global model using a second-order gradient using validation data. The decomposition here is :

$\texttt{Loc}_c$ Local SGD starting from a weight initialization. The hyperparameters in c are the same as in FedAvg. The algorithm also returns second-order information required to compute the meta-gradient.

$\texttt{Agg}_b$ Meta-gradient computation, summation, and updating using a standard descent method like Adam [**?**]. The hyperparameters in b are the hyperparameters of the latter.

C FedEx Details

C.1 Stochastic Gradient used by FedEx

Below is a simple calculation showing that the stochastic gradient used to update the categorical architecture distribution of FedEx is an unbiased approximation of the true gradient w.r.t. its parameters.

$$
\begin{aligned}
&\nabla_{\theta_j} \mathbb{E}_{c_{ij}|\theta} L_{V_{ti}}(\mathbf{w}_i) \\
&\quad = \nabla_{\theta_j} \mathbb{E}_{c_{ij}|\theta} (L_{V_{ti}}(\mathbf{w}_i) - \lambda) \\
&\quad = \mathbb{E}_{c_{ij}|\theta} \left((L_{V_{ti}}(\mathbf{w}_i) - \lambda) \nabla_{\theta_j} \log \mathbb{P}_\theta(c_{ij}) \right) \\
&\quad = \mathbb{E}_{c_{ij}|\theta} \left((L_{V_{ti}}(\hat{w}_k) - \lambda) \nabla_{\theta_j} \log \prod_{i=1}^{n} \mathbb{P}_\theta(c_{ij} = c_j) \right) \\
&\quad = \mathbb{E}_{c_{ij}|\theta} \left((L_{V_{ti}}(\mathbf{w}_i) - \lambda) \sum_{i=1}^{n} \nabla_{\theta_j} \log \mathbb{P}_\theta(c_{ij} = c_j) \right) \\
&\quad = \mathbb{E}_{c_{ij}|\theta} \left(\frac{(L_{V_{ti}}(\mathbf{w}_i) - \lambda) 1_{c_{ij}=c_j}}{\theta_j} \right)
\end{aligned}
$$

Note that this use of the reparameterization trick has some similarity with a recent RL approach to tune the local step-size and number of epochs [38]; however, FedEx can be rigorously formulated as an optimization over the personalization objective, has provable guarantees in a simple setting, uses a different configuration distribution that leads to our exponentiated update, and crucially for practical deployment does not depend on obtaining aggregate reward signal on each round.

C.2 FedEx wrapped with SHA

For completeness, we present the pseudo code of wrapping FedEx with SHA in Algorithm 3 below.

Algorithm 3: FedEx wrapped with SHA

Input: distribution $\mathcal{D}$ over hyperparameters $\mathcal{A}$, elimination rate $\eta \in \mathbb{N}$, elimination rounds $\tau_0 = 0, \tau_1, \ldots, \tau_R$

sample set of η^R hyperparameters $H \sim \mathcal{D}^{[\eta^R]}$

initialize a model $\mathbf{w}_a \in \mathbb{R}^d$ for each $a \in H$

for *elimination round* $r \in [R]$ **do**

  for *setting* $a = (b, c) \in H$ **do**

    $s_a, \mathbf{w}_a, \theta_a \leftarrow$ FedEx $(\mathbf{w}_a, b, c, \theta_a, \tau_{r+1} - \tau_r)$

  $H \leftarrow \{a \in H : s_a \leq \frac{1}{\eta}\text{-quantile}(\{s_a : a \in H\})\}$

Output: remaining $a \in H$ and associated model $\mathbf{w}_a$

FedEx $(\mathbf{w}, b, \{c_1, \ldots, c_k\}, \theta, \tau \geq 1)$:

initialize $\theta_1 \leftarrow \theta$

initialize shared weights $\mathbf{w}_1 \leftarrow \mathbf{w}$

for *comm. round* $t = 1, \ldots, \tau$ **do**

  for *client* $i = 1, \ldots, B$ **do**

    send $\mathbf{w}_t, \theta_t$ to client

    sample $c_{ti} \sim \mathcal{D}_{\theta_t}$

    $\mathbf{w}_{ti} \leftarrow \texttt{Loc}_{c_{ti}}(T_{ti}, \mathbf{w}_t)$

    send $\mathbf{w}_{ti}, c_{ti}, L_{V_{ti}}(\mathbf{w}_{ti})$ to server

  $\mathbf{w}_{t+1} \leftarrow \texttt{Agg}_b(\mathbf{w}, \{\mathbf{w}_{ti}\}_{i=1}^B)$

  set step size η_t and baseline λ_t

  $\tilde{\nabla}_j \leftarrow \frac{\sum_{i=1}^B |V_{ti}|(L_{V_{ti}}(\mathbf{w}_{ti}) - \lambda_t)1_{c_{ti}=c_j}}{\theta_{t[j]} \sum_{i=1}^B |V_{ti}|} \; \forall \; j$

  $\theta_{t+1} \leftarrow \theta_t \odot \exp(-\eta_t \tilde{\nabla})$

  $\theta_{t+1} \leftarrow \theta_{t+1} / \|\theta_{t+1}\|_1$

  $s \leftarrow \sum_{i=1}^B |V_{ti}| L_{V_{ti}} / \sum_{i=1}^B |V_{ti}|$

Return s, model $\mathbf{w}$, hyperparameter distribution θ

C.3 Hyperparameters of FedEx

We tune the computation of the baseline λ_t, which we set to

$$\lambda_t = \frac{1}{\sum_{s<t} \gamma^{t-s}} \sum_{s<t} \frac{\gamma^{t-s}}{\sum_{i=1}^B |V_{ti}|} \sum_{i=1}^B L_{V_{ti}}(\mathbf{w}_i)$$

for discount factor $\gamma \in [0, 1]$. As discussed in Section 3, the local perturbation factor is set to $\varepsilon = 0.1$. 27 configurations are used in each arm for SHA and RS. The number of configuration used per arm of FedEx (i.e. the dimensionality of θ) is the same (27).

D Experimental Details

Code implementing `FedEx` is available at `https://github.com/mkhodak/fedex`. The code automatically downloads CIFAR-10 data, while Shakespeare and FEMNIST data is made available by the LEAF repository: `https://github.com/TalwalkarLab/leaf`.

D.1 Settings of the Baseline/Wrapper Algorithm

We use the same settings of Algorithm 1 for both tuning `FedAvg` and for wrapping `FedEx`. Given an elimination rate η, number of elimination rounds R, resource budget B, and maximum rounds per arm M, we assign $T_1, \dots, T_R$ s.t.

$$T_i - T_{i-1} = \frac{T - M}{\frac{\eta^{n+1}-1}{\eta-1} - n - 1}$$

(recall $T_0 = 0$) and assign any remaining resources to maximize resource use. All remaining details were noted in Section 5.

D.2 Hyperparameters of `FedAvg`/`FedProx`/`Reptile`

Server hyperparameters (learning rate $\alpha_t = \gamma^t$):

$$\begin{aligned} \log_{10}\text{lr}: &\quad \text{Unif}[-1, 1] \\ \text{momentum}: &\quad \text{Unif}[0, 0.9] \\ \log_{10}(1-\gamma): &\quad \text{Unif}[-4, -2] \end{aligned}$$

Local training hyperparameters (note we only use 1 epoch for Shakespeare to conserve computation):

$$\begin{aligned} \log_{10}(\text{lr}): &\quad \text{Unif}[-4, 0] \\ \text{momentum}: &\quad \text{Unif}[0.0, 1.0] \\ \log_{10}(\text{weight-decay}): &\quad \text{Unif}[-5, -1] \\ \text{epoch}: &\quad \text{Unif}\{1, 2, 3, 4, 5\} \\ \log_2(\text{batch}): &\quad \text{Unif}\{3, 4, 5, 6, 7\} \\ \text{dropout}: &\quad \text{Unif}[0, 0.5] \end{aligned}$$

E Confidence Intervals

Table 2: Final test error obtained when tuning using a standard hyperparameter tuning algorithm (SHA or RS) alone, or when using it for server (aggregation) hyperparameters while `FedEx` tunes client (on-device training) hyperparameters. The target model is the one used to compute on-device validation error by the wrapper method, as well as the one used to compute test error after tuning. The confidence intervals displayed are 90% Student-t confidence intervals for the mean estimates from Table 1, with 5 independent trials for Shakespeare, 10 for FEMNIST, 10 for RS on CIFAR, and 6 for SHA on CIFAR.

Wrapper method	Target model	Tuning method	Shakespeare i.i.d.	Shakespeare non-i.i.d.	FEMNIST i.i.d.	FEMNIST non-i.i.d.	CIFAR-10 i.i.d.
Random Search (RS)	global	RS (server & client)	60.32 ± 9.56	64.36 ± 13.53	22.81 ± 2.64	22.98 ± 1.98	30.46 ± 5.47
		+ `FedEx` (client)	53.94 ± 8.70	57.70 ± 16.75	20.96 ± 2.77	22.30 ± 2.12	34.83 ± 8.54
	personalized	RS (server & client)	61.10 ± 8.89	61.71 ± 8.66	17.45 ± 1.63	17.77 ± 1.52	34.89 ± 6.12
		+ `FedEx` (client)	54.90 ± 9.50	56.48 ± 12.97	16.31 ± 2.19	15.93 ± 1.77	39.13 ± 8.77
Successive Halving (SHA)	global	SHA (server & client)	47.38 ± 3.24	46.79 ± 3.35	18.64 ± 0.97	20.30 ± 0.96	21.62 ± 1.45
		+ `FedEx` (client)	$\mathbf{44.52} \pm 1.60$	$\mathbf{45.24} \pm 3.16$	19.22 ± 1.19	19.43 ± 0.84	$\mathbf{20.82} \pm 0.79$
	personalized	SHA (server & client)	46.77 ± 3.44	48.04 ± 3.54	$\mathbf{14.79} \pm 0.90$	14.78 ± 0.75	24.81 ± 3.55
		+ `FedEx` (client)	46.08 ± 2.45	45.89 ± 3.58	14.97 ± 0.76	$\mathbf{14.76} \pm 0.99$	21.77 ± 1.64

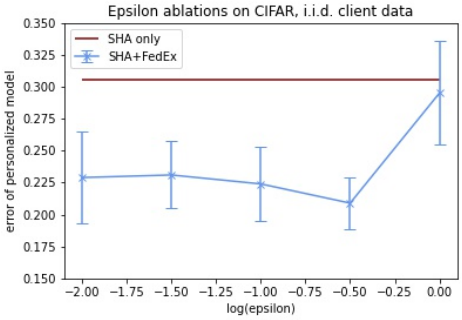

Figure 5: Comparison of different ε settings for the local perturbation component of FedEx from Section 3.

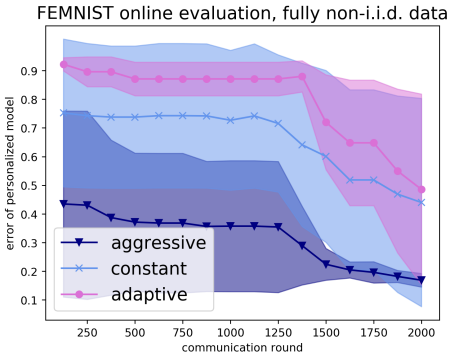

Figure 6: Comparison of step-size schedules for η_t in FedEx. In practice we chose the 'aggressive' schedule, which exhibits faster convergence to favorable configurations.

F Ablation Studies

We now discuss two design choices of FedEx and how they affect performance of the algorithm. First, the choice of the local perturbation $\varepsilon = 0.1$ discussed in Section 3; we choose this setting due to its consistent performance across several settings. In Figure 5 we plot the performance of FedEx on CIFAR-10 between $\varepsilon = 0.0$ (no FedEx, i.e. SHA only) and $\varepsilon = 1.0$ (full FedEx, i.e. client configurations are chosen independently) and show that while the use of a nonzero ε is important, performance at fairly low values of ε is roughly similar.

We further investigated the setting of the step-size η_t for the exponentiated gradient update in FedEx. We examine three different approaches: a constant rate of $\eta_t = \sqrt{2 \log k}$, an 'adaptive' schedule of $\eta_t = \sqrt{2 \log k} / \sqrt{\sum_{s \leq t} \|\tilde{\nabla}_s\|_\infty^2}$, and an 'aggressive' schedule of $\eta_t = \sqrt{2 \log k} / \|\tilde{\nabla}_t\|_\infty$. Here $\tilde{\nabla}_t$ is the stochastic gradient w.r.t. θ computed in Algorithm 2 at step t and the form of the step-size is derived from standard settings for exponentiated gradient in online learning [**?**]. We found that the 'aggressive' schedule works best in practice, as shown in Figure 6. A key issue with using the 'constant' and 'adaptive' approaches is that they continue to assign high probability to several configurations late in the tuning process; this slows down training of the shared weights. One could consider a tradeoff between allowing FedEx to run longer than while keeping the total budget constant, but for simplicity we chose the more effective 'aggressive' schedule.