FLASH: Automating Federated Learning using CASH

Md Ibrahim Ibne Alam[1] **Koushik Kar**[1] **Theodoros Salonidis**[2] **Horst Samulowitz**[2]

[1]Department of ECSE, Rensselaer Polytechnic Institute , Troy, NY, USA - 12180
[2]IBM T.J. Watson Research Center , Yorktown Heights, NY, USA - 10598

Abstract

In this paper, we present FLASH, a framework which addresses for the first time the central AutoML problem of Combined Algorithm Selection and HyperParameter (HP) Optimization (CASH) in the context of Federated Learning (FL).
To limit training cost, FLASH incrementally adapts the set of algorithms to train based on their projected loss rates, while supporting decentralized (federated) implementation of the embedded hyperparameter optimization (HPO), model selection and loss calculation problems. We provide a theoretical analysis of the training and validation loss under FLASH, and their tradeoff with the training cost measured as the data wasted in training sub-optimal algorithms. The bounds depend on the degree of dissimilarity between the datasets of the clients, a result of FL restriction that client datasets remain private. Through extensive experimental investigation on several datasets, we evaluate three variants of FLASH, and show that FLASH performs close to centralized CASH methods.

1 INTRODUCTION

Motivation. Federated learning (FL) is a distributed learning framework that enables training a model from decentralized data located at client sites, without the data ever leaving the clients. Compared to a centralized model, in which training requires all the data to be transmitted to and stored in a central location (e.g., a data center), FL has the benefits of preserving data privacy while avoiding transmission of large volumes of raw data from the client sites.

FL has two key challenges; first, the data across clients can be highly heterogeneous. Second, the communication overhead can be prohibitive during training as model parameters are exchanged in multiple global rounds between the clients and an aggregation server. Therefore there has been significant research effort on FL techniques that reduce communication overhead during model training. Such techniques typically assume that the clients agree on a common algorithm and hyperparameters (HPs) before training occurs, i.e. do not provide AutoML capabilities.

Recently the problem of HyperParameter Optimization (HPO) in FL has been addressed in several works. HPO in FL is an important problem as the choice of HPs can dramatically affect FL system performance. The FL setting poses unique challenges in addressing HPO, due to non-iid data, limited processing power at clients, and function evaluations for an HP set being much more communication and computation intensive than the centralized setting because they require FL training. Solving the algorithm selection along with HPO (popularly known as CASH) in an FL setting inherits the aforementioned challenges of HPO in FL and adds the additional layer of complexity of algorithm selection, where different algorithms have different performance as well as different HP sets. In this paper, we propose (for the first time) a way to solve the CASH problem for an FL setting without performing any FL in the solution process (i.e., only using FL after solving CASH). In prior literature, the CASH problem has only been addressed in the centralized setting, and most approaches treat it as a more complex HPO problem that merges the HPs of all algorithms and adds the algorithm type as a new HP. Extending the HPO algorithms to use this approach would not be adequate due to the explosion on HP dimensionality and computation complexity; in addition it is not evident how to aggregate these new CASH HPs to an single optimal HP set.

FLASH overview. In this paper, we propose and evaluate FLASH, a framework which solves the CASH problem in an FL setting by viewing it as bi-level optimization problem: the algorithm selection problem being solved at the outer level requires solving the embedded HPO problem at the inner level. FLASH solves the algorithm selection problem using a multi-fidelity approach, where, for each algorithm,

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:45–55.

the inner level HPO method (which we term FL-HPO) runs on increasing subsets of the clients' data, providing data increments to a subset of best performing algorithms according to a projected loss curve and subject to a tolerance threshold. This avoids wasting training resources on poorly performing algorithms. We analyze and evaluate the FLASH framework under three FL-HPO methods: *Local Best Model (LBM)*, *Local K-Best Model (LKBM)*, and *Regression based Model (RM)*. These FL-HPO approaches allow the clients to run HPO separately on their private data, but differ in how the results from the individual clients are aggregated at the central server and further re-validated at the clients, before the final HP choice is determined for the algorithm choice made at the outer level algorithm selection problem. Instead of expensive FL training, each HP configuration is evaluated using an approximation metric modeled as a linear combination of the clients' local loss functions. This in turn enables FLASH to reduce communication and computation overhead by first performing CASH search for the best algorithm-hyperparameter (Alg-HP) configuration in only a few rounds of communication between the clients and the central server, and then performing a single FL training to reach the final model for this configuration.

We provide a theoretical analysis of the worst-case loss performance and the wasted training cost measured as the data allocated for training sub-optimal algorithms. The performance bounds are expressed in terms of the dissimilarity between the client dataset distributions and other key parameters. Our extensive experimental study investigates these trade-offs and shows that FLASH can achieve a performance that is close to that of centralized CASH.

Summary of contributions. To summarize, the key novel contributions of this work are as follows.

- We present FLASH, a framework that solves for the first time the CASH problem in a FL setting by decomposing it into algorithm selection (outer level) and FL-HPO (inner level) problems. FLASH minimizes communication and communication overhead during CASH search using a multi-fidelity incremental data approach at the algorithm selection level and by avoiding expensive FL training-based evaluations at the FL-HPO level. Only a single FL training is needed for the Alg-HP configuration found during CASH search.
- We provide a theoretical analysis of the convergence and worst-case loss performance of all three FLASH variants, and the wasted training cost measured as the data allocated for training sub-optimal algorithms. These performance bounds are expressed in terms of the dissimilarity between the client dataset distributions and other key parameters.
- We provide numerical evaluation of FLASH on eight large data sets with seven algorithm choices, for all three FL-HPO variants and several baseline approaches. We compare the accuracy and training cost for these variants, and the performance effects of some of the parameter choices and options that FLASH provides.

2 RELATED WORK

FL Training. A large number of optimization techniques have been devised to address the communication and computation overhead during FL training Li et al. [2020a]. These techniques assume that algorithm and HPs are known. Since FLASH decouples CASH search from FL training, these techniques can be viewed as complementary and could be applied during the FL training after CASH search.

Centralized CASH approaches. A popular approach is to view the CASH problem as an extended HPO problem by merging the HPs of all algorithms and introducing the algorithm type as a new HP Komer et al. [2014], Kotthoff et al. [2017] . Then, iterative Bayesian Optimization methods (BO) for HPO are used to solve this HPO problem Shahriari et al. [2015]. BO avoids expensive model training/validation evaluations by estimating the shape of the loss landscape with a surrogate model and suggesting the HP configuration to be evaluated in the next iteration. A major challenge is that the explosion on the HP space introduced by CASH limits the efficiency of BO. Existing solutions include using different surrogate models (random forests Lindauer et al. [2022], trees Olson and Moore [2016]), combining BO with Hyperband Li et al. [2017](a bandit strategy that dynamically allocates resources to a set of random configurations and uses successive halving Jamieson and Talwalkar [2016] to stop poorly performing configurations) Falkner et al. [2018], and multi-fidelity optimization which uses subsets of the data to perform and project on faster training-based evaluations Klein et al. [2017]. Apart from treatment as HPO problem some recent approaches have used reinforcement learning Efimova et al. [2017], adaptive allocation of HPO iterations to algorithms Li et al. [2020b], and alternating direction method of multipliers Liu et al. [2020]. Implementing these approaches to an FL setting directly can be computation and communication exhaustive.

HPO approaches for FL. Most HPO approaches for FL focuses on finding local client HPs such as learning rates [Koskela and Honkela, 2019, Mostafa, 2019, Reddi et al., 2020], number of local SGD iterations [Wang et al., 2019] or global HPs common to all clients such as network architectures [He et al., 2020, Garg et al., 2020, Xu et al., 2020], for SGD training algorithms and deep neural networks (DNNs). Fedex [Khodak et al., 2020, 2021] uses the NAS technique of weight sharing combined with successive halving to tune local HPs at clients to build personalized models. FLoRA Zhou et al. [2022] extends the above works beyond SGD/DNNs to any training algorithm and provides a framework for tuning global HPs. HPO algorithms for FL cannot apply the approach of viewing CASH as an HPO problem: the exploded HP search space would be too vast for

each client (which is more processing limited than server) to execute CASH. Furthermore, each client would result in a different Alg-HP configuration and it is not evident how to aggregate this information to a single Alg-HP configuration. In contrast, FLASH addresses this problem using an outer algorithm selection layer and an inner HPO layer, which are solved in a decentralized manner.

3 CASH FORMULATION IN FL

We first define the CASH problem in an FL setting. Similar to the standard CASH problem considered in a centralized setting Thornton et al. [2013], Zöller and Huber [2021], we are given a set of algorithms $\mathcal{A} = (A^{(1)}, \cdots, A^{(J)})$, where each algorithm $A^{(j)}$ is associated with hyperparameters (HPs) that belong to domain $\Lambda^{(j)}$. Each algorithm choice $A^{(j)}$ and HP setting $\boldsymbol{\lambda} \in \Lambda^{(j)}$, compactly written as $A^{(j)}_{\boldsymbol{\lambda}}$, is associated with a model class $\mathcal{W}^{(j)}_{\boldsymbol{\lambda}}$, from which a model (parameter vector) $\boldsymbol{w} \in \mathcal{W}^{(j)}_{\boldsymbol{\lambda}}$ must be chosen so as to minimize a predictive loss function $\mathcal{L}(\boldsymbol{w}, \mathcal{D}')$ over a validation dataset $\mathcal{D}'$.

In an FL setting McMahan et al. [2017], Yang et al. [2019], the training dataset $\mathcal{D}$ is partitioned into several subsets $\mathcal{D}_i, i \in \mathcal{C}$ that are owned individually by a set of $N = |\mathcal{C}|$ clients. Thus $\mathcal{D} = \cup_{i \in \mathcal{C}} \mathcal{D}_i$. We assume that $\mathcal{D}_i$ is private to client i, and cannot be shared or aggregated due to privacy or complexity reasons. Given an algorithm and HP choice, $A^{(j)}_{\boldsymbol{\lambda}}$, an FL algorithm $\mathcal{F}$ aims to determine a model $\boldsymbol{w}$ using the training dataset, $\mathcal{F}(A^{(j)}_{\boldsymbol{\lambda}}, \cup_i \mathcal{D}_i) \longrightarrow \boldsymbol{w} \in \mathcal{W}^{(j)}_{\boldsymbol{\lambda}}$, where the training dataset $\mathcal{D}$ is written as $\cup_i \mathcal{D}_i$ to emphasize its distributed (partitioned) nature. Usually, $\boldsymbol{w}$ is chosen to minimize the training error, modeled with the given loss function $\mathcal{L}$ but computed over the training dataset $\mathcal{D}$. That is, the FL algorithm $\mathcal{F}$ typically aims to minimize $\mathcal{L}(\boldsymbol{w}, \cup_i \mathcal{D}_i)$ over $\boldsymbol{w} \in \mathcal{W}^{(j)}_{\boldsymbol{\lambda}}$, using iterative methods that involve local model training at the individual clients (using their private datasets) and sharing information on models and their accuracies (but not data) with a central aggregator.

Although not necessary for the validity of our analysis or results, for ease of exposition we assume that the validation dataset $\mathcal{D}'$ is partitioned across the clients as well. Thus $\mathcal{D}' = \cup_i \mathcal{D}'_i$, where $\mathcal{D}'_i$ is the validation dataset of client i. Then given the underlying FL algorithm $\mathcal{F}$ for finding the model (for any Alg-HP setting), the CASH problem for FL involves finding $A^{\star}_{\boldsymbol{\lambda}^{\star}}$ that minimizes a global loss function, computed as the aggregation of loss functions at the clients (over their validation datasets)

$$A^{\star}_{\boldsymbol{\lambda}^{\star}} = \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda^{(j)}}{\arg\min} \sum_i \alpha_i \, \mathcal{L}(\mathcal{F}(A^{(j)}_{\boldsymbol{\lambda}}, \cup_i \mathcal{D}_i), \mathcal{D}'_i), \quad (1)$$

where α_i are appropriately defined client weights, such as $\alpha_i = \frac{1}{N}$ or $\alpha_i = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$, and the FL function ($\mathcal{F}$) also uses these weights for computing the loss in the training process.

While solving this CASH problem, we seek to: (1) adhere to the core FL requirement that the datasets $\mathcal{D}_i$ remain private; (2) minimize the number of communication rounds between the server and the clients, including the rounds needed by the federated learning (model training) algorithm $\mathcal{F}$. The development of the FLASH framework, described next, is guided by these practical requirements.

4 FLASH FRAMEWORK

Even in a centralized setting, solving the CASH problem of finding the best Alg-HP pair $A^{\star}_{\boldsymbol{\lambda}^{\star}}$ is computationally expensive due the large number (set) of Alg-HP combinations over which the loss function must be minimized. This is more complex (i.e., communication intensive) in an FL setting, as the loss evaluation for any specific $A^{(j)}_{\boldsymbol{\lambda}}$ requires solving the underlying FL (model training) problem that may take multiple (possibly many) rounds of communication between the clients and the central server. To address this complexity issue, FLASH adopts three broad principles or approximations, as listed below. These approximations introduce a degree of sub-optimality in the solving the CASH problem in FL, which is quantified through our theoretical analysis in the next section.

Firstly, in FLASH the global loss for any $A^{(j)}_{\boldsymbol{\lambda}}$ setting is computed by aggregating the losses computed at the clients on their individual datasets. In other words, in comparing the Alg-HP settings in FLASH, the loss function in FLASH (compare with (1)) is calculated as

$$\sum_i \alpha_i \, \mathcal{L}(\mathcal{F}(A^{(j)}_{\boldsymbol{\lambda}}, \mathcal{D}_i), \mathcal{D}'_i). \quad (2)$$

We note that $\cup_i \mathcal{D}_i$ within the model training function $\mathcal{F}$ in (1) is replaced by $\mathcal{D}_i$ in (2). This implies that in FLASH, model training (and therefore loss computation) happens locally in each client, avoiding the communication-intensive procedure of computing the global model through FL. This allows FLASH to compute the global loss function (albeit approximately), for a given $A^{(j)}_{\lambda}$ and training dataset, in a single round of communication.

Secondly, FLASH divides up the CASH problem in FL into two levels (see Algorithm 1): the outer level ('for' loop in Step 2) which requires finding the optimal algorithm $A^{(j)} \in \mathcal{A}$ (for their best HP setting), and the inner level problem that requires finding the best HP $\boldsymbol{\lambda} \in \Lambda^{(j)}$ for any given $A^{(j)}$. However, finding the best (global) HP $\boldsymbol{\lambda}$ for a given $A^{(j)}$, even for the separable loss function in (2), can be computation and communicative intensive in an FL setting. For this reason, FLASH approximates it using decentralized FL-HPO approaches (Step 9 of Algorithm 1, described later in this section) that work by aggregating the HPs (and their corresponding loss values) computed/validated separately by the clients on their individual datasets.

Finally, since training all algorithms on entire client datasets could be wasteful (particularly when the client datasets are large, or there are a large number of algorithm choices), FLASH allocates training data to the algorithms incrementally, focusing only the best performing algorithms at any time. More specifically, FLASH works in rounds, i.e., $0, 1, 2, ..., M$ (Step 2 in Algorithm 1), and keeps a running set of best performing algorithms $\tilde{\mathcal{A}} \subseteq \mathcal{A}$ that it updates after each round. In round m, FLASH evaluates the training loss on fraction a_m of the data (randomly chosen) at each client (by calling Algorithm 2 in Step 9), where $0 < a_0 < a_1 < \ldots < a_M = 1$, and projects the loss curve to the entire data set, in a manner similar to that described in some of the prior work on centralized algorithm selection Sabharwal et al. [2016], Li et al. [2020b]. More precisely, denoting $\ell(A^{(j)}, a_m)$ as the loss rate for algorithm $A^{(j)}$ calculated in round m by FLASH, the loss projection (LP) for $A^{(j)}$ is computed by linearly extrapolating $\ell(A^{(j)}, a_m)$ from a_m to $a_M = 1$, i.e. $LP(A^{(j)}) = \ell(A^{(j)}, a_m) + (1 - a_m) \cdot \ell'(A^{(j)}, a_m)$ (Step 11). Here, $\ell'(A^{(j)}, a_m)$ is an estimate of the derivative of the loss rate curve based on the loss rates calculated by FLASH so far, $\ell(A^{(j)}, a'_m), m' \leq m$. Also, $LP^*(a_m) = \min_j LP(A^{(j)}, a_m)$ is the minimum projected loss computed at step m for all $A^{(j)}$ (Step 6). Then for a chosen *tolerance factor* Δ (an input parameter), in any round FLASH selects all algorithms (for training in the next round) whose projected loss is within Δ of the best projected loss in that round (Step 6). Note that to calculate the loss projection (LP) for each algorithm at least two values ($m = 2$) are needed. However, due to the small (fractional) datasets used in the initial steps, $m = 2$ may lead to a very noisy loss projection. Hence, we chose to go up to 3 iterations ($m = 3$) to estimate the initial LP of all algorithms (Steps 3 to 4) and start choosing algorithms (to allocate training data to) from $m = 4$.

Computing the loss function on fraction a_m of the training dataset for any algorithm $A^{(j)}$ requires finding the optimum HP $\boldsymbol{\lambda} \in \Lambda^{(j)}$ for that dataset. Since this dataset is spread across the clients, this optimization in done in a federated manner, by aggregating (at the central server) the HPs and corresponding losses by running per-client HPOs. This is referred to as FL-HPO in Algorithm 1 (Step 9), and is described below.

Algorithm 1 FLASH

1: **Input:** Set of all algorithms $\mathcal{A}$, tolerance parameter Δ.
2: **for** $m = 0, 1, \ldots, M$ **do**
3: **if** $m < 4$ **then**
4: $\tilde{\mathcal{A}}$= $\mathcal{A}$.
5: **else**
6: $\tilde{\mathcal{A}}$ = [Algorithms for which $LP(A^{(j)}, a_{m-1}) - LP^*(a_{m-1}) \leq \Delta$].
7: **end if**
8: **for** each algorithm $A^{(j)} \in \tilde{\mathcal{A}}$ **do**
9: Call *FL-HPO*($A^{(j)}, a_m$) to get best HP $\lambda(A^{(j)}, a_m)$ and its loss $\ell(A^{(j)}, a_m)$.
10: Store the best (HP, loss) pair.
11: Update LP of the algorithm: $LP(A^{(j)}, a_m) = \ell(A^{(j)}, a_m) + (1 - a_m) \cdot \frac{\ell(A^{(j)}, a_m) - \ell(A^{(j)}, a_{m-1})}{a_m - a_{m-1}}$.
12: **end for**
13: $LP^*(a_m) = \min_j LP(A^{(j)}, a_m)$
14: **end for**
15: Set $A^\dagger = argmin_{A^{(j)}} \ell(A^{(j)}, a_M)$ algorithm with minimum loss in the last iteration, and $\lambda^\dagger = \lambda(A^\dagger, a_M)$, its best HP.
16: **Output:** $A^\dagger$, $\lambda^\dagger$.

FLASH FL-HPO ALGORITHM

Our FL-HPO method is summarized by Algorithm 2. It provides a way to compute the best HP for a given algorithm $A^{(j)}$ and data size fraction a in a decentralized manner and can be implemented easily in any FL platform. There are three variants of our FL-HPO method called $LBM, LKBM, RM$; the main difference among them is how they aggregate local HPs computed by the clients to find a globally optimal HP.

Initially, when the FL-HPO method (Algorithm 2) is called, each client i creates a subset of its dataset by sampling a fraction a of its rows (Step 1). Then it runs locally an HPO algorithm (e.g. HyperOpt) on this subset for a given number of iterations (HPO_{iter}). Each iteration evaluates an HP on the subset using k-fold cross-validation and yields a loss value L_{iter}. A set of HPs and their loss values explored by HPO is then communicated to the server by the client as (HP, loss) pairs. Then the server aggregates the HPs and yields a set of candidate global HPs using one of the three variants described as follows (Step 2):

Local Best Model (LBM): In LBM, each client sends its best (HP, loss) pair to the server. The aggregator computes the global HP set by performing max-voting to categorical HP coordinates (ties broken randomly) and averaging the numerical HP coordinates across the clients' HP sets.

Local K-Best Model (LKBM): In $LKBM$, each client sends its K-best (HP,loss) pairs explored by its HPO to the server. Then the server sends these $K \times N$ HP pairs as candidate global HPs to all clients for evaluation (each client will evaluate the K-best HPs of the others).

Regression based Model (RM): In RM, each client sends all (HP,loss) pairs explored by its HPO to the server. Then the server uses all these pairs to train a regressor model (we used Random Forest). After training, the regressor model, is used to compute the predicted losses for a large number of

Algorithm 2 FL-HPO $(A^{(j)}, a)$

1: **Run Local HPO:** At each client i, run HPO on a fraction of its dataset, and send the best HP(s) and corresponding loss(es) (L_{iter}) to the aggregator.
2: **Aggregate Results:** Aggregate the results using one of the following methods:
3:
- *LBM:* Calculate HP by max-voting or averaging the best HP setting of each client.
- *LKBM:* Take the union of the top K HP settings of each client for re-evaluation.
- *RM:* Perform regression using κ HPs (and their losses) collected per client to generate top K HP settings for re-evaluation.

4: **Re-Evaluation** (*LKBM* and *RM* only): Send the HP setting candidates back to clients for re-evaluation.
5: **Final Aggregation** (*LKBM* and *RM* only): Average the re-validated HP settings and corresponding losses.
6: **Output:** HP $\boldsymbol{\lambda} \in \Lambda^{(j)}$ and corresponding loss value.

HP settings (generated randomly), and the top-10 performing ones are kept. These HPs form the global HP set are sent to the clients for re-evaluation. RM is similar to FLoRA, the FL-HPO approach presented in Zhou et al. [2022], but with the additional re-evaluation step.

After the candidate set of global HP sets are determined (from Step 2), these sets are sent to the clients for re-evaluation. The clients evaluate them using k-fold cross-validation on their data subsets and send back to the server the global HP sets and their corresponding loss values (Step 4). Finally, the server averages the losses of each global HP set sent by the clients and selects the global HP set with the minimum average loss (Step 5). It is to be noted that Step 4 and 5 are only executed for $LKBM$ and RM variants, whereas the global best HP is found at step 3 for the LBM variant. In other words, in LBM the HPs are computed by just combining the best HPs provided by the clients, i.e., the server does not send any HP(s) back to the clients for re-validation. This makes LBM simpler, but as we will see later in Section 6, it results in a slightly worse performance then the other two variants.

5 THEORETICAL ANALYSIS

In this section, we provide a theoretical analysis of the loss optimality and training cost of FLASH; proofs of the results are included in the Supplementary Material (Alam et al. [2023]).

Preliminaries. Let $\mathcal{D}^a$ represent a dataset comprising of a fraction of the training data, and $\mathcal{D}_i^a$ the corresponding per-client datasets; from Algorithm 1, recall that a varies as $a_0, a_1, \cdots, a_M$. For a given algorithm $A^{(j)}$, let $\underline{\ell}(A^{(j)}, a)$ represent the *true training loss* for algorithm $A^{(j)}$ when using a fraction of the data. Since this training loss depends on how the dataset $\mathcal{D}^a$ is chosen, the true training loss can be estimated by averaging over the losses computed over all possible datasets $\mathcal{D}^a$, denoted by $\boldsymbol{\mathcal{D}}^a = \{\mathcal{D}^a \subseteq \mathcal{D}, |\mathcal{D}^a| = a|\mathcal{D}|\}$. Therefore, from (1), $\underline{\ell}(A^{(j)}, a)$ can be expressed as,

$$\underline{\ell}(A^{(j)}, a) = \mathbb{E}_{\mathcal{D}^a \in \boldsymbol{\mathcal{D}}^a} \min_{\boldsymbol{\lambda} \in \Lambda^{(j)}} \sum_i \alpha_i \, \mathcal{L}(\mathcal{F}(A^{(j)}_{\boldsymbol{\lambda}}, \cup_i \mathcal{D}_i^a), \mathcal{D}_i^a).$$

From (1), note that $\mathcal{D}_i'$ is replaced by $\mathcal{D}_i^a$, since $\underline{\ell}(\cdot, a)$ denotes the training cost on a fraction of the dataset.

Assuming that all dataset sizes $|\mathcal{D}_i^a|$ are sufficiently large, and cross-validation is considered, it is reasonable to assume that the true loss function is smooth and convex in a (since loss functions are usually convex with respect to training data). We further assume that $\underline{\ell}$ has bounded second derivatives, i.e., $\underline{\ell}''(A^{(j)}, a) \leq B, \forall a, \forall A^{(j)}$.

Let $\ell(A^{(j)}, a)$ represent the training loss computed for algorithm $A^{(j)}$ under FLASH, when using a fraction of the data. Note that ℓ will in general differ from the true training loss $\underline{\ell}$ for several reasons: (i) The FL-HPO algorithm may calculate the HP sub-optimally; (ii) $\ell(A^{(j)}, a)$ may be calculated over one (or a few) datasets $\mathcal{D}^a \in \boldsymbol{\mathcal{D}}^a$, instead of averaging over all possible datasets in $\boldsymbol{\mathcal{D}}^a$. Let σ represent the maximum difference between $\ell(A^{(j)}, a)$ and $\underline{\ell}(A^{(j)}, a)$, i.e., $|\ell(A^{(j)}, a) - \underline{\ell}(A^{(j)}, a)| \leq \sigma, \forall a, \forall A^{(j)}$. The value of σ depends on which of the three FL-HPO variants is used, and is provided later in this section. Finally, let δ be the minimum difference between the a_m, i.e., $\delta = \min\{a_0, \min_{m \in \{1, \cdots, M\}}(a_m - a_{m-1})\}$.

Loss Optimality Analysis Let ℓ^* be the minimum training loss achievable, i.e., the minimum value of $\underline{\ell}(A^{(j)}, 1)$ across all algorithms $A^{(j)} \in \mathcal{A}$. In the following lemma, which is key to bounding loss performance and training cost of FLASH, the optimum algorithm refers to the one that attains the minimum training loss ℓ^*.

Lemma 1. *If $\Delta > B + 2\sigma + \frac{4\sigma}{\delta}$, FLASH ensures the training of the optimum algorithm (that attains ℓ^*) in every iteration $m = \{0, \cdots, M\}$.*

Lemma 1 quantifies how large the tolerance parameter Δ needs to be so that the optimum algorithm is allocated data in every round. This leads to the following result, which shows in terms of *training loss*, FLASH can be sub-optimal by at most 2σ.

Theorem 2. *If $\Delta > B + 2\sigma + \frac{4\sigma}{\delta}$, then the Alg-HP pair chosen by FLASH attains a training loss that is within σ (for $LKBM$ and RM) and 2σ (for LBM) of the minimum training loss, ℓ^*.*

Training Cost Analysis Next we try to bound the degree of *wasteful training*, measured by the amount of training data allocated to sub-optimal algorithms. Let ϵ_j denotes how sub-optimal algorithm $A^{(j)}$ is, in terms of the training loss, i.e., $\epsilon_j = \underline{\ell}(A^{(j)}, 1) - \ell^*$.

Theorem 3. *If* $\epsilon_j \leq \frac{B}{2} + 2\sigma + \frac{4\sigma}{\delta} + \Delta$*, then algorithm* $A^{(j)}$ *receives the full dataset for training under FLASH; otherwise, the fraction of the training data allocated in rounds* $m \geq 4$ *by FLASH to any algorithm* $A^{(j)}$ *is no more than* $\max\left\{0, 1 - \sqrt{\frac{\epsilon_j - B/2 - 2\sigma - 4\sigma/\delta - \Delta}{B/2}}\right\}$.

Theorem 3 implies that the algorithms whose true training loss is within $(\frac{B}{2} + 2\sigma + \frac{4\sigma}{\delta} + \Delta)$ of ℓ^* receives full training; algorithms whose true training loss is beyond $(B + 2\sigma + \frac{4\sigma}{\delta} + \Delta)$ of ℓ^* do not receive any training data at all (except in the initial 3 rounds when all algorithms are trained). If their true training loss is within these two limits, then those sub-optimal algorithm incurs a training cost that decreases monotonically with ϵ_j.

Bounding the Loss Calculation Error We now proceed to bound σ, as defined earlier, by computing an upper bound on $|\ell(A^{(j)}, a) - \underline{\ell}(A^{(j)}, a)|$ over all $A^{(j)}, a$ values. To capture how the training loss rate for a given algorithm $A^{(j)}$ varies with the distribution of the training dataset $\hat{\mathcal{D}}$, we define loss function $\hat{l}(A^{(j)}, \hat{\mathcal{D}})$ as

$$\hat{\ell}(A^{(j)}, \hat{\mathcal{D}}) = \min_{\boldsymbol{\lambda} \in \Lambda^{(j)}} \sum_i \alpha_i \, \mathcal{L}(\mathcal{F}(A^{(j)}_{\boldsymbol{\lambda}}, \cup_i \hat{\mathcal{D}}_i), \hat{\mathcal{D}}_i).$$

For any algorithm $A^{(j)}$, and any two training datasets $\hat{\mathcal{D}}_1, \hat{\mathcal{D}}_2$, we assume that $\hat{\ell}$ satisfies $|\hat{\ell}(A^{(j)}, \hat{\mathcal{D}}_1) - \hat{\ell}(A^{(j)}, \tilde{\mathcal{D}}_2)| \leq \beta \cdot \nu(\hat{\mathcal{D}}_1, \hat{\mathcal{D}}_2)$, for some scalar constant β, with ν being 1-Wasserstein distance measure between the distributions of the two training datasets $\hat{\mathcal{D}}_1, \hat{\mathcal{D}}_2$. Further, let $\underline{\mathcal{D}}^a$ denote the expectation of the distributions of all the datasets in $\boldsymbol{\mathcal{D}}^a$. Let $d_j(\cdot, \cdot)$ denote the 1-norm distance metric in $\Lambda^{(j)}$, the hyperparameter space of $A^{(j)}$. Further, let $\boldsymbol{\lambda}^i_k$, $k \in [k] = \{1, \cdots, \kappa\}$ denote the κ HP choices of client i in RM. Define $D_j = \sum_i \alpha_i \min_{k \in [\kappa]} d_j(\boldsymbol{\lambda}, \boldsymbol{\lambda}^i_k)]$, where $\min_{k \in [\kappa]} d_j(\boldsymbol{\lambda}, \boldsymbol{\lambda}^i_k)]$ determines the worst case distance of any HP in the space $\Lambda^{(j)}$ from the closest initial HP chosen by client i. Let $\bar{D}$ be an upper bound on $D_j \forall j$.

The upper bound on σ depends on which FLASH variant is being used, and can be stated as follows.

Theorem 4. *For the training dataset* $\mathcal{D}^a$*, the loss calculation error for any algorithm* $A^{(j)}$ *is upper-bounded by* $\sigma(a)$*, given as* $\sigma(a) = \beta_0 \, \nu(\mathcal{D}^a, \underline{\mathcal{D}}^a) + \hat{\sigma}(a)$*, where*

$$\hat{\sigma}(a) = \begin{cases} \beta_1 \sum_i \alpha_i \, \nu(\mathcal{D}^a_i, \mathcal{D}^a) & (LBM) \\ \beta_2 \max_{i,i'} \nu(\mathcal{D}^a_i, \mathcal{D}^a_{i'}) & (LKBM) \\ \beta_3 \sum_i \alpha_i \, \nu(\mathcal{D}^a_i, \mathcal{D}^a) + \gamma \bar{D} & (RM) \end{cases}$$

for appropriately defined scalar constants β_0*,* β_1*,* β_2*,* β_3*, and* γ*. Then* σ *is given by* $\sigma = \max_{a \in [a_0, \cdots, a_m]} \sigma(a)$.

In the above results, LKBM has been analyzed for the conservative case of $K = 1$. Note that the bound for RM depends on κ (through $\bar{D}$), the number of initial HPs chosen by each client, as it determines the accuracy of the HPO. The bounds are not directly comparable between the three FL-HPO models as the constants β_1, β_2, β_3 can be different. However, the key takeaway from the bounds is that the loss calculation errors (and therefore the overall loss performance bounds as computed by Theorem 2) depend on the dissimilarity between the client datasets. This results from the fact that in these FL-HPO approaches, the HPs (losses) are optimized (calculated) on the individual client datasets and then aggregated, instead of being computed globally.

Table 1: Comparison of *CASH-D*, *CASH-O* and Auto-SKL

DataSet	*CASH-D*	*CASH-O*	Auto-SKL
EEG - Eye	93.10	94.05	97.42
Electricity	91.04	93.51	93.24
Eye Movement	69.87	73.73	75.58
Diabetic Data	53.36	56.79	51.82
Connect - 4	72.97	75.54	76.01
Higgs	72.18	72.56	72.83
Magic Telescope	86.13	86.66	85.47
Default of Credit	73.91	75.51	70.61

6 EMPIRICAL EVALUATION

Dataset Selection: We initially selected 35 datasets (from OpenML Vanschoren et al. [2013]) with more than 10000 data-samples (the dataset sizes ranged from about 11k to about 100k). We split the datasets in a training part and validation part. We trained models using the training part and used accuracy on the validation part as performance metric. We considered seven well performing algorithms: *Random Forest, Decision Tree, Extra Tree, Logistic Regression, XGB, LGBM* and *MLP* with well defined HP space for simulation. We define *CASH-D* and *CASH-O* as the validation accuracies found from the best performing algorithm (in terms of training accuracy) using default HP settings of scikit-learn Pedregosa et al. [2011] and with the optimized HP setting found through HPO, respectively.

As expected, *CASH-O* attained better accuracy than *CASH-D*. However, 23 out of the 35 datasets showed very minor improvement indicating that HPO does not yield much gain over the default HPs. However, we observed that the algorithm choice did have a significant impact on the performance. From the remaining 12 datasets we selected 8 datasets (name and accuracies in Table 1) where CASH-O demonstrated higher gains and are diverse in terms of number of examples and features. In Table 1 we also provide the accuracy values attained by centrally solving the CASH problem with auto-sklearn (Auto-SKL). The higher accuracies attained by Auto-SKL compared to *CASH-O* in some cases is largely due to the fact that Auto-SKL spans a much larger algorithm set and HP space.

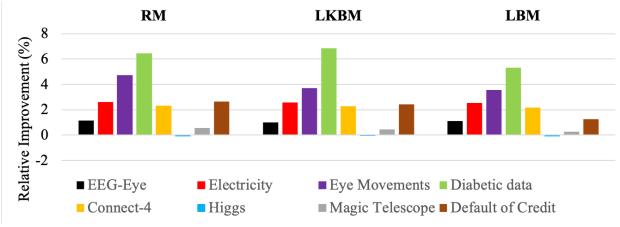

Figure 1: RI of FLASH compared to *CASH-D*

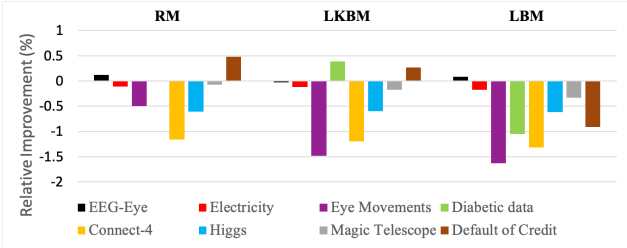

Figure 2: RI of FLASH compared to *CASH-O*

Baselines and Evaluation Metric: *CASH-D* and *CASH-O* are used as the performance evaluation baselines for FLASH for the 8 datasets that we selected. Let us define P_D and P_* as the performance (accuracy) of *CASH-D* and *CASH-O* respectively for some dataset. For that same dataset if FLASH attains accuracy P, we define relative improvement with respect to *CASH-D* (*CASH-O*) as RI_D (RI_*, respectively), calculated as $\frac{P-P_D}{P_D} \times 100\%$ ($\frac{P-P_*}{P_*} \times 100\%$, respectively). The RI reflects how much % improvement is achieved by FLASH compared to the centralized baselines. A negative RI value means that FLASH is performing worse than the baseline.

Implementation: FLASH is implemented with two major loops (Algorithm 1), where the outer loop performs the algorithm selection and the inner loop performs FL-HPO (Algorithm 2). For each of the 8 datasets we selected, the generation of the client training and validation datasets was done as follows: first the dataset was randomly divided in N subsets, one for each client. Then, each client's dataset was divided to training and validation parts with a ratio of 70-30 using stratified sampling on target class. Performance was measured as the average validation accuracy across clients. For training and evaluations, we used Hyperopt Komer et al. [2014], a mainstream and easily configurable HPO algorithm, using 10-fold cross-validation. We also used different data seeds (which change the client data distributions) and different HPO seeds (which change the initial HP used by HyperOpt. Unless otherwise specified, each experiment was ran 25 times with at different data and HPO seeds. Moreover, the value of a_m in our empirical evaluation followed a geometric progression (not necessary for the theoretical analysis), implying $a_m = a_0 \cdot r^m$. For the results in the paper, we used $a_0 = 3.75\%$ and a progression rate of $r = 1.5$. While there can be other ways of choosing $a_0, a_1, ...$, geometric progression was used because we expect the change of the slope to slow down for larger values of m.

Comparison of different versions of FLASH: We first perform experiments on all 3 FL-HPO approaches ($LBM, LKBM, RM$) with $\Delta = 0$, $N = 3$ clients, and $HPO_{iter} = 50$ HPO iterations. We compare the performance of all three variants of FLASH (RM, LKBM, LBM) using their RI with respect to the baselines (i.e., *CASH-D*, *CASH-O*). Fig. 1 shows the value of RI when FLASH is compared to *CASH-D*. The improvement in performance with FLASH is quite prominent and for some datasets RI is as high as 5%. For 'Higgs' dataset the performance was decreased for some versions of FLASH, but that is less than 0.1%. We observe that all three variants of FLASH perform consistently while RM usually yields the best performance. This is an important finding because it shows that FLASH which performs CASH on *distributed* client datasets performs better than an approach using optimal algorithm selection *over default HPs* and assuming all data is available at a central location.

Table 2: Robustness of FLASH (Random HP setting)

Value of Δ	Avg. Error (%)	Training Cost (norm.)
0	0.441	1.00
0.2	0.427	1.0095
0.4	0.358	1.0995
0.6	0.365	1.2096
0.8	0.343	1.2867
1.0	0.336	1.4394

Fig. 2 depicts the RI of FLASH compared to the *CASH-O*. In this case it is expected for FLASH to not yield improvement as *CASH-O* uses optimal algorithm selection and HPO on centralized data. The small negative RI values (up to -1.5%) indicate that FLASH is performing worse but very close to the centralized *CASH-O* solution. A few cases yield very small positive RI values (less than 0.5%) which indicate that FLASH slightly outperforms CASH-O, which sounds counter-intuitive. These cases arise due to over-fitting in the CASH-O model training which cause the depicted marginal decrease in *validation accuracy*. Of course, CASH-O always performs better than FLASH in terms of training accuracy.

Run time for all the 3 variants of FL-HPO, normalized with respect to the average runtime of the slowest variant (RM), are provided in table 3, . Although FLASH RM is usually the better performing FL-HPO technique, however it consistently takes the longest to run due to the additional regression analysis. Also, the communication overhead for FLASH RM and $LKBM$ (not experimentally evaluated in this study) are twice as of LBM due to the re-evaluation. Moreover, the runtime for RM and $LKBM$ tend to increase more than LBM with the increase of clients, possibly

Table 3: Run-time comparison

# of clients	RM	LKBM	LBM
3	1.00	0.981	0.793
5	1.00	0.985	0.778
10	1.00	0.988	0.76
20	1.00	0.992	0.751

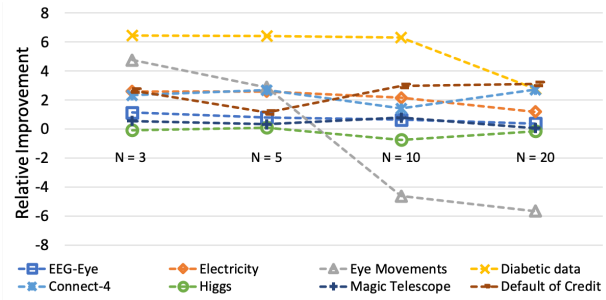

Figure 3: Effect of # of clients (N)

because of the re-evaluation done on the former two.

We now perform an ablation study to evaluate FLASH performance when Δ, N and HPO_{iter} are varied.

Impact of Tolerance Parameter (Δ): We ran FLASH for Δ ranging between 0 and 1. For each Δ, we performed 100 runs that included different data seed (client data distributions), HP seed (different initial Hyperopt HPs) and HPO iterations (HPO_{iter}). Table 2 quantifies FLASH performance in terms of (%) Avg. Error and training cost. For each Δ, the Avg. Error is computed by averaging the difference between the accuracy of each run and the best accuracy over the 100 runs. The training cost is the ratio of the average training time of a Δ over the average training time of $\Delta = 0$ (training time increases with increased Δ), where average is taken over the 100 runs. We see that $\Delta = 0$ yields Avg. Error of 0.44%, which is very low and slightly higher than that of the higher Δs. Also training cost increases abruptly for $\Delta > 0.6$. Thus, $\Delta < 0.6$ seems best for our method and $\Delta = 0$ is a good value for the datasets we considered. It should be noted that increasing Δ ensures a larger set of algorithms (having higher accuracy projection) to be trained at each round (m) and therefore a lower chance of picking a sub-optimal algorithm. Hence with a higher Δ value, the average error decreases while the average training cost increases.

Impact of Number of Clients: Fig. 3 depicts the RI in performance for FLASH RM compared to *CASH-D* for different values of N. Notably, 6 of the 8 datasets showed very consistent results, with the RI values varying over a narrow range with variation in N. The performance decreases monotonically with increasing N for *Eye Movements* and shows a sharp drop at $N = 20$ for *Diabetic data*. However, upon close inspection of Fig. 3, we do see some non-monotonic

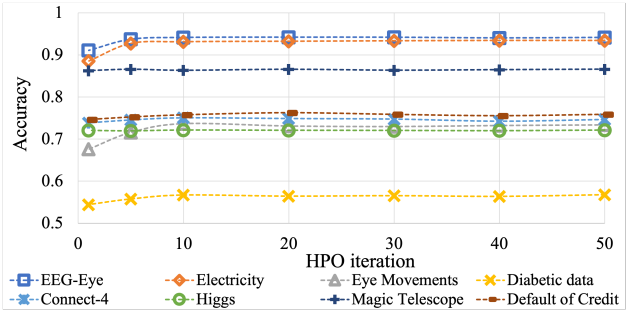

Figure 4: Effect of HPO iterations

behavior in the performance of FLASH RM. The overall performance of FLASH RM is impacted by factors such as: i) the amount of data per client and ii) the number of HP settings reported back to the central server to perform FL-HPO. In our empirical evaluation, we divided the whole dataset with equal data-samples per client, hence with larger N each client had fewer data-samples. Usually, less data per client leads to worse loss estimates, whereas with more HP settings reported (due to a larger number of clients), FLASH–RM is able to come up with better HP settings. Thus, there are two opposing forces in play here as we increase N, and it is difficult to identify under what conditions one factor would dominate the other.

Impact of Number of HPO iterations: The HPO_{iter} was varied from 1 to 50 on each client's dataset, and for FLASH RM the accuracy values are plotted against number of iteration in Figure 4. We observe that accuracy in general increases when HPO_{iter} goes from 1 to 10 for FLASH RM (it takes a few more iterations for LBM and $LKBM$), after which it becomes almost flat (or increases slowly). Reaching the optimal solution in a few iterations justifies the use of the three proposed FL-HPO algorithms, especially in the case where HPO iterations at the clients are compute-intensive.

Accuracy projections with training data: To get more insight on how FLASH works, the projection of accuracy (analogous to the loss projection) for a specific dataset (Electricity) is plotted in Fig. 5. It is worth noting that during the initial stages of FLASH, when the value of m is small (i.e., smaller training data), the Extra Tree and MLP algorithms exhibit remarkably high accuracy projections. However, as the training progresses, their accuracy projections decline, while algorithms like XGBoost and LGBM emerge as the frontrunners. Consequently, if FLASH had not reintroduced these algorithms in subsequent rounds and discarded them solely based on their initial performance, the overall accuracy would have been lower by 10%. These results use $\Delta = 100\%$, so that the projection value of all the algorithms can be observed in each round.

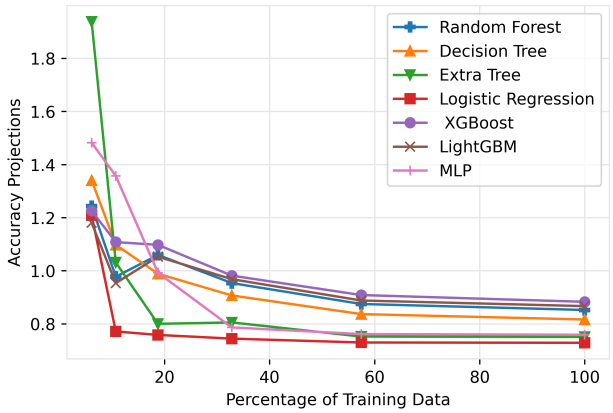

Figure 5: Accuracy projections with training data.

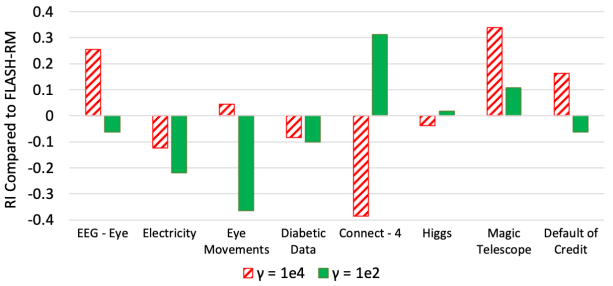

Figure 6: Impact of heterogeneity in client data distributions, depicting RI of FLASH-RM with different Dirichlet constant γ over FLASH with random distribution.

Controlled heterogeneity. We now evaluate the impact of the degree of heterogeneity on FLASH performance. In all of the aforementioned results, the label distributions of the clients were created by randomly dividing the entire dataset. We refer to FLASH with this random data distribution as FLASH-RANDOM. As in Hsu et al. [2019], we create several client data distributions by controlling the heterogeneity of labels of the data points distributed to the clients using Dirichlet constant γ (smaller γ yields more heterogeneous non-iid distribution). Fig. 6 depicts the RI values of FLASH-RM for two values of γ (10^2 and 10^4) over FLASH-RANDOM. RI has a small range (-0.4% to 0.35%) for both values of γ across all datasets, demonstrating that FLASH performs consistently across heterogeneous non-iid distributions.

7 CONCLUSION

We presented and evaluated FLASH, which solves the CASH problem in an FL setting by combining outer-level algorithm selection with inner-level FL-HPO methods, and requires the global FL model training problem to be solved only once, i.e., after the Alg-HP configuration has been selected. FLASH reduces training cost by allocating training data incrementally to only a subset of all algorithms based on their loss performances. Specifically, we theoretically analyzed and evaluated FLASH with three FL-HPO methods which are simple to implement, and compute the global HP and loss function by aggregating those computed individually by the clients on their private data sets. FLASH is able to identify near optimal Alg-HP configuration with a few rounds of communication between the clients and the central server, and easy to implement in an FL environment. Extensive simulations show consistent and competitive performance of FLASH upon comparing it with centralized benchmarks.

Acknowledgements

The work was supported by the Rensselaer-IBM AI Research Collaboration (`http://airc.rpi.edu`), part of the IBM AI Horizons Network (`http://ibm.biz/AIHorizons`).

References

Md Ibrahim Ibne Alam, Koushik Kar, Theodoros Salonidis, and Horst Samulowitz. FLASH: Automating Federated Learning using CASH (Supplementary Material). 2023. https://tinyurl.com/UAI-2023-657.

Valeria Efimova, Andrey Filchenkov, and Viacheslav Shalamov. Fast automated selection of learning algorithm and its hyperparameters by reinforcement learning. In *International Conference on Machine Learning AutoML Workshop*, 2017.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

Anubhav Garg, Amit Kumar Saha, and Debo Dutta. Direct federated neural architecture search. *arxiv.2010.06223*, 2020.

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.2004.08546*, 2020.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

Mikhail Khodak, Tian Li, Liam Li, M Balcan, Virginia Smith, and Ameet Talwalkar. Weight sharing for hyperparameter optimization in federated learning. In *Int. Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2020*, 2020.

Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *arXiv preprint arXiv:2106.04502*, 2021.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, 11 (2):4945–4968, 2017.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, pages 2825–2830. Citeseer, 2014.

A. Koskela and A. Honkela. Learning rate adaptation for federated and differentially private learning. *arXiv preprint arXiv:1809.03832*, 2019.

Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *J. Mach. Learn. Res.*, 18(1):826–830, January 2017. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=3122009.3122034.

Lisha Li, Kevin G Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR (Poster)*, page 53, 2017.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37 (3):50–60, 2020a.

Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. Efficient Automatic CASH via Rising Bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4763–4771, 2020b.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23 (54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. An ADMM based framework for automl pipeline configuration. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1905.00424v5.

B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. International Conference on Artificial Intelligence and Statistics*, pages 1273–1282, Ft. Lauderdale, FL, Apr 2017.

H. Mostafa. Robust federated learning through representation matching and adaptive hyper-parameters. *arXiv preprint arXiv:1912.13075*, 2019.

Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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*, 2020.

Ashish Sabharwal, Horst Samulowitz, and Gerald Tesauro. Selecting near-optimal learners via incremental data allocation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2487629. URL https://doi.org/10.1145/2487575.2487629.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *Journal Selected Areas in Communications (JSAC)*, 2019.

Mengwei Xu, Yuxin Zhao, Kaigui Bian, Gang Huang, Qiaozhu Mei, and Xuanzhe Liu. Federated neural architecture search. *arxiv.2002.06352*, 2020.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019. ISSN 2157-6904. doi: 10.1145/3298981. URL https://doi.org/10.1145/3298981.

Yi Zhou, Parikshit Ram, Theodoros Salonidis, Nathalie Baracaldo, Horst Samulowitz, and Heiko Ludwig. Single-shot hyper-parameter optimization for federated learning: A general algorithm and analysis. *arXiv preprint arXiv:2202.08338*, 2022. doi: 10.48550/ARXIV.2202.08338. URL https://arxiv.org/abs/2202.08338.

Marc-André Zöller and Marco F. Huber. Benchmark and survey of automated machine learning frameworks. *J. Artif. Int. Res.*, 70:409–472, may 2021. ISSN 1076-9757. doi: 10.1613/jair.1.11854. URL https://doi.org/10.1613/jair.1.11854.