FEDHPO-BENCH: A BENCHMARK SUITE FOR FEDERATED HYPERPARAMETER OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter optimization (HPO) is crucial for machine learning algorithms to achieve satisfactory performance. Its research progress has been boosted by existing HPO benchmarks. Nonetheless, existing efforts in benchmarking all focus on HPO for traditional centralized learning while ignoring federated learning (FL), a promising paradigm for collaboratively learning models from dispersed data. In this paper, we first identify some uniqueness of HPO for FL algorithms from various aspects. Due to this uniqueness, existing HPO benchmarks no longer satisfy the need to compare HPO methods in the FL setting. To facilitate the research of HPO in the FL setting, we propose and implement a benchmark suite FEDHPO-BENCH that incorporates comprehensive FedHPO problems, enables flexible customization of the function evaluations, and eases continuing extensions. We also conduct extensive experiments based on FEDHPO-BENCH to provide the community with more insights into FedHPO. We open-sourced FEDHPO-BENCH at https://github.com/FedHPO-Bench/FedHPO-Bench-ICLR23

1 INTRODUCTION

Most machine learning (ML) algorithms expose many design choices, which can drastically impact the ultimate performance. Hyperparameter optimization (HPO) (Feurer & Hutter, 2019) aims at making the right choices without human intervention. Formally, HPO can be described as the problem $\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$, where each Λ_k corresponds to the candidate choices of a specific hyperparameter, e.g., taking the learning rate from $\Lambda_1 = [0.01, 1.0]$ and the batch size from $\Lambda_2 = \{16, 32, 64\}$. For each specified λ, $f(\lambda)$ is the output result (e.g., validation loss) of executing the considered algorithm configured by λ.

Research in this line has been facilitated by HPO benchmarks (Gijsbers et al., 2019; Eggensperger et al., 2021; Pineda-Arango et al., 2021), which encourage reproducible and fair comparisons between different HPO methods. To this end, their primary efforts are two-fold: One is to keep the results of the same function evaluation consistent across different runtime environments, e.g., by containerizing its execution; The other is to simplify the evaluations, e.g., by evaluating a function via querying a readily available lookup table or a fitted surrogate model.

However, existing HPO benchmarks all focus on traditional learning paradigms, where the functions to be optimized correspond to centralized learning tasks. Federated learning (FL) (McMahan et al., 2017; Li et al., 2020a), as a privacy-preserving paradigm for collaboratively learning a model from distributed data, has not been considered. Actually, along with the increasing privacy concerns from the whole society, FL has been gaining more attention from academia and industry. Meanwhile, HPO for FL algorithms (denoted by FedHPO from now on) is identified as a critical and promising open problem in FL (Kairouz et al., 2019).

As an emerging topic, the community lacks a thorough understanding of how traditional HPO methods perform in the FL setting. Meanwhile, the recently proposed FedHPO methods have not been well benchmarked. Before attempting to fill this gap, it is helpful to pinpoint the difference between FedHPO and traditional HPO. In Section 2.2, we elaborate on such differences from various aspects, which essentially come from the distributed nature of FL and the heterogeneity among FL's participants. In summary, the domain of the objective function in FedHPO includes hyperparameter and fidelity dimensions that have not been considered in traditional non-FL scenarios, with the intricate correlations among these dimensions; The objective function in FedHPO changes depending

on the level of privacy and fairness concerns, while the cost of evaluating its value depends on the specific system conditions; The FL setting brings in challenges, such as a tighter budget constraint, as well as opportunities for FL participants to concurrently explore the search space.

Motivated by FedHPO's uniqueness and the successes of existing HPO benchmarks, we summarize the desiderata of FedHPO benchmarks as follows.

Comprehensiveness. FL tasks are diverse in terms of domain, model architecture, the level of heterogeneity among participants, etc. As their corresponding functions to be optimized by HPO methods are thus likely to be diverse, including a comprehensive collection of FL tasks is necessary for drawing an unbiased conclusion from comparisons.

Flexibility. Practitioners may have different levels of privacy and fairness concerns, which correspond to different objective functions and different fairness-related regularizers in FedHPO. Meanwhile, the execution time for function evaluation depends on the system condition. Thus, users should be allowed to flexibly tailor the FedHPO problems to their privacy budgets, fairness demands, and system conditions.

Extensibility. As a developing field, new FedHPO problems and novel FedHPO methods constantly emerge, and FL's best practice continuously evolves. Thus, what the community desired is more of a benchmarking tool that can effortlessly incorporate novel ingredients.

Towards these desiderata, we propose and implement FEDHPO-BENCH, a dedicated benchmark suite, to facilitate the research and application of FedHPO. FEDHPO-BENCH incorporates rich FL tasks from various domains with respective model architectures, providing realistic and, more importantly, comprehensive FedHPO problems for studying the related methods. Moreover, we provide users with the flexibility to customize each FedHPO problem by the specified privacy budget as well as by the specified type and strength of fairness regularization. We also provide a configurable system model for estimating the cost of function evaluation. Last but not least, we build FEDHPO-BENCH upon the recently open-sourced FL platform, FederatedScope (FS), which provides solid infrastructure and many off-the-shelf FL-related functionalities. Thus, it is easy for developers to extend FEDHPO-BENCH with more and more FedHPO problems and methods.

2 BACKGROUND AND MOTIVATIONS

We first give a brief introduction to the settings of HPO and its related benchmarks. Then we present and explain the uniqueness of FedHPO to show the demand for dedicated FedHPO benchmarks. Due to the space limitation, more discussions about works related to HPO, benchmarking HPO, FL, and FedHPO are deferred to Appendix B.

2.1 PROBLEM SETTINGS

HPO aims at solving $\min_{\lambda \in \Lambda_1 \times \cdots \times \Lambda_K} f(\lambda)$, where each Λ_k corresponds to candidate choices of a specific hyperparameter, and their Cartesian product (denoted by $\times$) constitute the search space. In practice, such Λ_k is often bounded and can be continuous (e.g., an interval of real numbers) or discrete (e.g., a set of categories/integers). Each function evaluation at a specified hyperparameter configuration λ means to execute the corresponding algorithm accordingly and return the value of considered metric (e.g., validation loss) as the result $f(\lambda)$. HPO methods generally solve such a problem with a series of function evaluations. As a full-fidelity function evaluation is extremely costly, multi-fidelity methods exploit low-fidelity function evaluation, e.g., training for fewer epochs (Swersky et al., 2014; Domhan et al., 2015) or on a subset of data (Klein et al., 2017; Petrak, 2000; Swersky et al., 2013), to approximate the exact result. Thus, it would be convenient to treat f as $f(\lambda, b), \lambda \in \Lambda_1 \times \cdots \times \Lambda_K, b \in \mathcal{B}_1 \times \cdots \times \mathcal{B}_L$, where each $\mathcal{B}_l$ corresponds to the possible choices of a specific fidelity dimension, e.g., taking *#epoch* from $\{10, \ldots, 50\}$.

HPO benchmarks (Gijsbers et al., 2019; Eggensperger et al., 2021; Pineda-Arango et al., 2021) have prepared many HPO problems, i.e., various kinds of objective functions, for comparing HPO methods. To evaluate these functions, HPO benchmarks, e.g., HPOBench (Eggensperger et al., 2021), often provide three modes: (1) "Raw" means truly executing the corresponding algorithm; (2) "Tabular" means querying a lookup table, where each entry corresponds to a specific $f(\lambda, b)$; (3) "Surrogate" means querying a surrogate model that might be trained on the tabular data.

2.2 UNIQUENESS OF FEDERATED HYPERPARAMETER OPTIMIZATION

Function evaluation in FL. Despite the various scenarios in FL literature, we restrict our discussion about FedHPO to one of the most general FL settings that have also been adopted in existing FedHPO works (Khodak et al., 2021; Zhang et al., 2021). Conceptually, there are N clients, each of which has its specific data, and a server coordinates them to learn a model θ collaboratively. Most FL algorithms, such as FedAvg (McMahan et al., 2017) and FedOpt (Asad et al., 2020), are designed to optimize θ under this setting. Such FL algorithms are iterative. In the t-th round, the server broadcasts the global model $\theta^{(t)}$; then the clients make local updates and send the updates back; finally, the server aggregates the updates to produce $\theta^{(t+1)}$. Obviously, this procedure consists of two subroutines—local updates and aggregation. Thus, λ can be divided into client-side and server-side hyperparameters according to which subroutine each hyperparameter influences. After executing an algorithm configured by λ for, e.g., T rounds according to the specified fidelity, the best performance, e.g., validation loss, ever achieved during the FL course is regarded as $f(\lambda)$.

The execution of an FL algorithm is essentially a distributed machine learning procedure while distinguishing from general non-FL cases by the heterogeneity among clients (Xie et al., 2022). These characteristics make FedHPO unique against HPO for traditional machine learning algorithms. We summarize the uniqueness from the following perspectives:

Hyperparameter dimensions. The FL setting introduces new hyperparameters such as server-side *learning_rate* for FedOpt (Asad et al., 2020) and client-side *#local_update_step*, which correlate with hyperparameters related to general ML procedures. In Appendix H.3.1, we compare the landscape on ML-related hyperparameters (*batch_size* and client-side *learning_rate*) under the FL setting to that under the non-FL setting. Their differences suggest the uniqueness of FedHPO's objective functions.

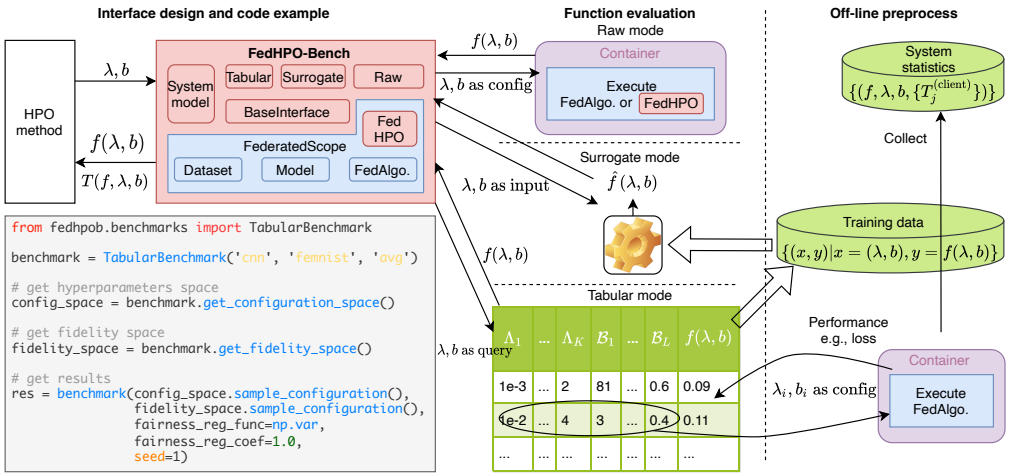
Fidelity dimensions. FedHPO introduces a new fidelity dimension—*client_sample_rate*, which determines the fraction of clients sampled in each round. When considering a *client_sample_rate* less than that of a full-fidelity function evaluation, each round of the corresponding FL course would take less time because there is less likely to be a straggler. However, *client_sample_rate* correlates with another fidelity dimension—*#round*, where a lower *client_sample_rate* often leads to federated aggregation with larger variance, which is believed to need more rounds for convergence. In Appendix G, we balance these two fidelity dimensions w.r.t. the system condition, e.g., choosing large *#round* but small *client_sample_rate* when the straggler issue is severe, to achieve more economical accuracy-efficiency trade-offs.

Privacy. In practice, FL is often combined with privacy protection techniques such as differential privacy (DP) to improve privacy protection (Kairouz et al., 2019). When considering DP with different privacy budgets, the results $f(\lambda, b)$ for the same λ and b are likely to differ. FEDHPO-BENCH enables users to consider the objective function f under different choices of privacy budgets. As practitioners must be interested in the trends of optimal configurations under different privacy budgets, such trends in FEDHPO-BENCH are analyzed in Appendix H.3.2.

Fairness. Despite the average performance, fairness (Li et al., 2021a; Wang et al., 2021b) is also critical in FL, which concerns the client-wise performances. Thus, a FedHPO benchmark is expected to either expose a vector-valued objective function that corresponds to client-wise performances or, as usual, a scalar-valued function where the original objective is regularized by a fairness-related metric, which is often calculated from the client-wise performance. In Section 3.2, we will discuss how FEDHPO-BENCH allows users to flexibly tailor a benchmark to their fairness concerns.

Concurrent exploration. Unlike centralized learning, where each execution can only try a specific λ, some FedHPO methods, such as FedEx (Khodak et al., 2021), try out different client-side configurations in each round and update a policy according to the feedback from all these clients. However, the heterogeneity among clients is likely to make them have different optimal configurations (Koskela & Honkela, 2020), where making decisions by the same policy would become unsatisfactory. How we incorporate several recent FedHPO methods into FEDHPO-BENCH and make it extensible is discussed in Section 3.3, and whether such a concurrent exploration strategy is useful is empirically answered in Section 4.2.

Due to the uniqueness mentioned above, existing HPO benchmarks are inappropriate for studying FedHPO. FedHPO calls for dedicated benchmarks that incorporate functions corresponding to FL algorithms and respect realistic FL settings.




```
from fedhpob.benchmarks import TabularBenchmark

benchmark = TabularBenchmark('cnn', 'femnist', 'avg')

# get hyperparameters space
config_space = benchmark.get_configuration_space()

# get fidelity space
fidelity_space = benchmark.get_fidelity_space()

# get results
res = benchmark(config_space.sample_configuration(),
                fidelity_space.sample_configuration(),
                fairness_reg_func=np.var,
                fairness_reg_coef=1.0,
                seed=1)
```

Figure 1: Overview of FEDHPO-BENCH.

3 OUR PROPOSED BENCHMARK SUITE: FEDHPO-BENCH

We present an overview of FEDHPO-BENCH in Figure 1. Conceptually, FEDHPO-BENCH encapsulates function evaluation and provides a unified interface for HPO methods to interplay with it. Following the design of HPOBench (Eggensperger et al., 2021), function evaluations can be conducted in either of the three modes: *raw*, *tabular*, or *surrogate*. For the raw mode, we chose to build FEDHPO-BENCH upon the well-known FL platform FederatedScope (FS) (Xie et al., 2022), which has provided its docker images so that we can containerize FEDHPO-BENCH effortlessly by executing each FL algorithm in an FS docker container. To generate the lookup table for tabular mode, we truly execute the corresponding FL algorithms with the grids of search space as their configurations. These lookup tables are adopted as training data for the surrogate models, which are expected to approximate the objective functions (more details about this approximation are discussed in Appendix H.3.4). It's important to note that the distributed nature of FL makes it very expensive to run an FL course, so, in FedHPO, the tabular and surrogate modes are much in demand to meet the efficiency requirement. For the convenience of users, we keep FEDHPO-BENCH's interface basically the same as HPOBench's. Meanwhile, we expose extra arguments for users to customize the instantiation of a benchmark. We defer the discussion of the relationship between FEDHPO-BENCH and HPOBench to Appendix B.1.

In this section, we elaborate on three highlights of FEDHPO-BENCH. In addition to the off-the-shelf FL-related components that FS already provides, we contribute new datasets, models, and algorithms to FS to prepare a comprehensive collection of FedHPO problems (see Section 3.1). Moreover, users are allowed to flexibly tailor these problems to their specific cases (see Section 3.2). Also, FS's event-driven framework allows us to easily extend FEDHPO-BENCH by incorporating more FedHPO problems and methods, which is valuable for this nascent research direction (see Section 3.3).

3.1 COMPREHENSIVENESS

There is no universally best HPO method (Gijsbers et al., 2019). Therefore, for the purpose of fairly comparing HPO methods, it is necessary to compare them on a variety of HPO problems that correspond to diverse objective functions and thus can comprehensively assess their performances.

To satisfy this need, we leverage FS to prepare various FL tasks, where their considered datasets and model architectures are quite different. Specifically, the data can be images, sentences, graphs, or tabular data. Some datasets are provided by existing FL benchmarks, including FEMNIST (from LEAF (Caldas et al., 2018)) and split Cora (from FS-GNN (Wang et al., 2022)), which are readily distributed and thus conform to the FL setting. Some are centralized initially (e.g., those from OpenML (Joaquin Vanschoren & Torgo, 2013; Gijsbers et al., 2019) and Hugging Face (Socher et al., 2013)), which we partition by FS's splitters to construct their FL version with various kinds of

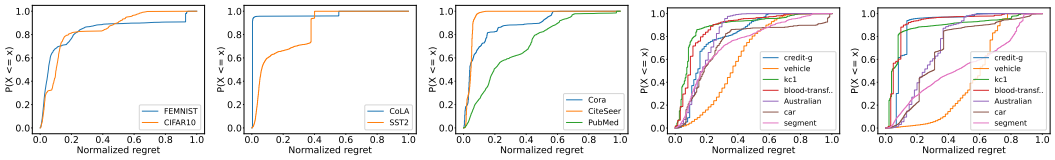
Table 1: Summary of benchmarks in current FEDHPO-BENCH: MF refers to matrix factorization. Rec. and Algo. are short for recommendation and algorithm, respectively. #Cont. and #Disc. denote the number of hyperparameter dimensions corresponding to continuous and discrete candidate choices, respectively. The unit of the budget is either day (d) or second (s).

Scenario	Model	#Dataset	Domain	#Client	#Algo.	#Cont.	#Disc.	Budget
Cross-Silo	CNN	2	CV	200	2	4	2	20d
	BERT	2	NLP	5	2	4	2	20d
	GNN	3	Graph	5	2	4	1	1d
	LR	7	Tabular	5	2	3	1	21,600s
	MLP	7	Tabular	5	2	4	3	43,200s
Cross-Device	MF	1	Rec.	480,189	2	3	1	-
	LR	1	NLP	$\sim$3300	2	3	1	1d

Non-IIDness among clients. All these datasets are publicly available and can be downloaded and preprocessed by our prepared scripts. More details of these datasets can be found in Appendix D. Then the corresponding suitable neural network model is applied to handle each dataset, involving fully-connected networks, convolutional networks, the latest attention-based model, etc. It is worth noticing that our prepared FL tasks cover both cross-silo and cross-device scenarios. In cross-device scenario, there are a lot more clients and a much lower *client_sample_rate* than in cross-silo scenario.

For each such FL task, we employ two FL algorithms, FedAvg and FedOpt, to learn the model from this dataset, respectively. Then the FedHPO problem is defined as optimizing the design choices of the FL algorithm on each specific FL task. So, without further explanation, we use the triple <dataset, model, algorithm> to index a particular benchmark in the remainder of this paper. We summarize our currently provided FedHPO problems in Table 1, and more details can be found in Appendix H. It is worth mentioning that, for each benchmark, in addition to *#round*, *client_sample_rate*, which is unique to FL, is adopted.

We study the empirical cumulative distribution function (ECDF) for each model type in the cross-silo benchmarks. Specifically, in creating the lookup table for tabular mode, we have conducted function evaluations for the hyperparameter configurations located on a very dense grid over the search space, resulting in a finite set $\{(\lambda, f(\lambda))\}$ for each benchmark. Then we normalize the performances (i.e., $f(\lambda)$) and show their ECDF in Figure 2, where these curves exhibit different shapes. For example, the amounts of top-tier configurations for GNN on PubMed are remarkably less than on other graph datasets, which might imply a less smoothed landscape and difficulty in seeking the optimal configuration. As the varying shapes of ECDF curves have been regarded as an indicator of the diversity of benchmarks (Eggensperger et al., 2021), we can conclude from Figure 2 that FEDHPO-BENCH enables evaluating HPO methods comprehensively.



(a) CNN **(b)** BERT **(c)** GNN **(d)** LR **(e)** MLP

Figure 2: Empirical Cumulative Distribution Functions: The normalized regret is calculated for all evaluated configurations of the respective model on the respective FL task with FedAvg.

3.2 FLEXIBILITY

We allow users to instantiate each benchmark with arguments other than the <dataset, model, algorithm> triple. So, both the underlying objective function and the amount of time it takes to calculate its value can be customized according to the cases specified by the users.

One perspective to customize a FedHPO benchmark is the privacy budget, which shows up when differential privacy (DP) is combined with FL to avoid privacy leakage (Kairouz et al., 2019). A

smaller budget usually means a stronger privacy protection requirement, which corresponds to more noise applied to gradients. As a result, the learning dynamics of FL change, and thus different privacy budgets often correspond to different HPO objective functions. Hence, we employ a representative FL+DP algorithm NbAFL (Wei et al., 2020) provided by FS and prepare the tabular benchmarks with several different choices of privacy budgets. Meanwhile, in the raw mode, specifying any valid value for the privacy budget will lead to the execution of NbAFL accordingly.

Another perspective to customize a FedHPO benchmark is the fairness concern. In addition to the mean performance (averaged over all clients), users may also care about fairness. To satisfy this need, in creating the tabular benchmarks, we have recorded the client-wise performances. Thus, the execution of an FL course can be regarded as a vector-valued function rather than a scalar-valued one. Then, we allow users to specify the type and strength of the fairness-related regularizer so that the ultimate function to be optimized by HPO methods becomes, for example, a linear interpolation of mean validation loss and the variance of client-wise validation losses.

In addition to customizing the objective function, it is also much helpful to customize the execution time of function evaluation because the execution time of the same FL course can vary a lot when deployed in environments with different system conditions. Many existing HPO benchmarks record the execution time of the ML courses, which cannot be adapted regarding users' system conditions. Despite a recorded execution time, we provide a system model to estimate the time consumed by evaluating $f(\lambda, b)$ in realistic scenarios, which is configurable so that users with different system conditions can calibrate the model to their cases. Based on the analysis of such a system model and a basic instance (Wang et al., 2021a), we propose and implement our system model, where the execution time $T(f, \lambda, b)$ for each round in evaluating $f(\lambda, b)$ is estimated as follow:

$$
\begin{aligned}
T(f, \lambda, b) &= T_{\text{comm}}(f, \lambda, b) + T_{\text{comp}}(f, \lambda, b), \\
T_{\text{comm}}(f, \lambda, b) &= \frac{S_{\text{down}}(f, \lambda)}{B_{\text{down}}} + \frac{S_{\text{up}}(f, \lambda)}{B_{\text{up}}} + \alpha(N), \\
T_{\text{comp}}(f, \lambda, b) &= \mathbb{E}_{T_i^{(\text{client})} \sim \text{Exp}(\cdot \mid \frac{1}{c(f,\lambda,b)}), i=1,\ldots,N}[\max(\{T_i^{(\text{client})}\})] + T^{(\text{server})}(f, \lambda, b),
\end{aligned}
\tag{1}
$$

where N denotes the number of clients sampled in this round, $\alpha(N)$ denotes the latency, which is an increasing function of N but is independent of the message size (contains the time needed to establish the transmission between the server and the clients), $S(f, \lambda)$ denotes the download/upload size, B denotes the download/upload bandwidth of client, $T^{(\text{server})}$ is the time consumed by server-side computation, and $T_i^{(\text{client})}$ denotes the computation time consumed by i-th client, which is sampled from an exponential distribution with $c(f, \lambda, b)$ as its mean. This design intends to simulate the heterogeneity among clients' computational capacity, where the assumed exponential distribution has been widely adopted in system designs (Wang et al., 2021a) and is consistent with real-world applications (Huba et al., 2022). We defer the details about our system model to Appendix E.

3.3 EXTENSIBILITY

As FedHPO is springing up, we must reduce the effort of introducing more FedHPO problems and novel FedHPO methods to FEDHPO-BENCH.

Recall that a FedHPO problem is characterized by the <dataset, model, algorithm> triple. With FS, we can apply the off-the-shelf data splitters to transform an arbitrary centralized dataset into an FL dataset, reuse any open-sourced model implementation by registering it in FS, and develop a novel FL algorithm via plugging in the hook function that expresses its unique step(s).

Traditional HPO methods are decoupled from the procedure of function evaluation, with a conceptually standard interface for interaction (see Figure 1). Thus, any such method is readily applicable to solve the FedHPO problems. However, FedHPO methods, such as FedEx (Khodak et al., 2021), are coupled with the FL course to make concurrent exploration, as the red color "FedHPO" module in Figure 1 shows. We need to implement such methods in FS if we want to benchmark them on FEDHPO-BENCH.

We present a general algorithmic view in Figure 3, which unifies several such methods as well as their personalized counterparts. At a high level, a policy π for determining the hyperparameters is

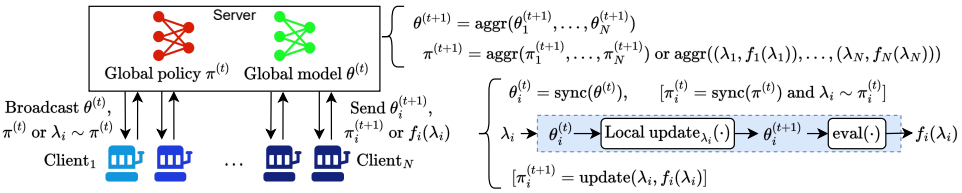


Figure 3: A general algorithmic view for FedHPO methods: They are allowed to concurrently explore different client-side configurations in the same round of FL, but the clients are heterogeneous, i.e., corresponding to different functions $f_i(\cdot)$. Operators in brackets are optional.

to be federally learned, along with the FL course itself. In the t-th communication round: (1) In addition to the model $\theta^{(t)}$, either the policy $\pi^{(t)}$ or its decisions $\lambda_i \sim \pi^{(t)}$ is also broadcasted. (2) For the i-th client, if $\pi^{(t)}$ is received, it needs to synchronous its local policy $\pi_i^{(t)}$ with this global one and then sample a hyperparameter configuration λ_i from its local policy. (3) Either received or locally sampled, λ_i is used to specify the local update procedure of FL, which results in updated local model $\theta_i^{(t+1)}$. (4) Then $\theta_i^{(t+1)}$ is evaluated to provide the result of (client-specific) function evaluation $f_i(\lambda_i)$. (5) For personalized FedHPO methods that maintain a local policy π_i, it is updated w.r.t. $(\lambda_i, f_i(\lambda_i))$ to produce $\pi_i^{(t+1)}$. (6) In addition to the local model $\theta_i^{(t+1)}$, either the local policy $\pi_i^{(t+1)}$ or the feedback $(\lambda_i, f_i(\lambda_i))$ is sent to the server. (7) Finally, the server aggregates $\theta_i^{(t+1)}$s into $\theta^{(t+1)}$ and $\pi_i^{(t+1)}$s/$(\lambda_i, f_i(\lambda_i))$s into $\pi^{(t+1)}$, respectively.

We have implemented FTS, FedEx, and a personalized FedEx in FS with such a view, where the differences between them lie in just the definition of the *sync*, *update*, and *aggr* operations. Benefiting from the event-driven framework of FS, all we need to develop are augmenting the messages exchanged by FL participants and plugging those HPO-related operations in the event-handlers. We defer more implementation details about FTS and FedEx to Appendix F.

4 Experiments

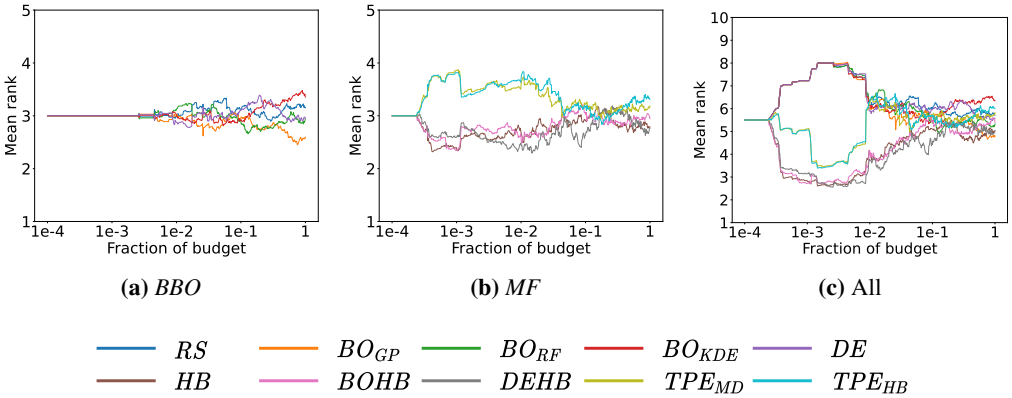
We conduct extensive empirical studies with our proposed FedHPO-Bench. Basically, we exemplify the use of FedHPO-Bench in comparing HPO methods, which, in the meantime, can somewhat validate the correctness of FedHPO-Bench. Moreover, we aim to gain more insights into FedHPO, answering three research questions: **(RQ1)** How do traditional HPO methods behave in solving FedHPO problems? **(RQ2)** Do recently proposed FedHPO methods that exploit "concurrent exploration" (see Section 2.2) significantly improve traditional methods? **(RQ3)** How can we leverage the new fidelity dimension of FedHPO? We conduct empirical studies in Section 4.1 and Section 4.2 to answer RQ1 and RQ2, respectively. Due to the limited space, discussions about RQ3 are deferred to Appendix G. All scripts concerning the studies here have been committed to FedHPO-Bench so that the community can quickly reproduce our established benchmarks.

4.1 Studies about Applying Traditional HPO Methods in the FL Setting

To answer RQ1, we largely follow the experiment conducted in HPOBench (Eggensperger et al., 2021) but focus on the FedHPO problems FedHPO-Bench provided.

Protocol. We employ up to ten optimizers (i.e., HPO methods) from widely adopted libraries (see Table 6 for more details). For black-box optimizers (*BBO*), we consider random search (*RS*), the evolutionary search approach of differential evolution (*DE* (Storn & Price, 1997; Awad et al., 2020)), and bayesian optimization with: a GP model (BO_{GP}), a random forest model (BO_{RF} (Hutter et al., 2011b)), and a kernel density estimator (BO_{KDE} (Falkner et al., 2018b)), respectively. For multi-fidelity optimizers (*MF*), we consider Hyperband (*HB* (Li et al., 2017)), its model-based extensions with KDE-based model (*BOHB* (Falkner et al., 2018a)), and differential evolution (*DEHB* (Awad et al., 2021)), and Optuna's implementations of TPE with median stopping (TPE_{MD}) and TPE with Hyperband (TPE_{HB}) (Akiba et al., 2019). We apply these optimizers to solve the cross-silo FedHPO problems summarized in Table 1, where the time budget is relaxed for these traditional HPO methods to satisfy multiple full-fidelity function evaluations rather than a one-shot setting. For the sake of efficiency, we conduct this experiment with the *tabular* mode of FedHPO-Bench, and consider *#round* as

the fidelity dimension for ML optimizers to controll (while keeping *client_sample_rate*=1.0). To compare the optimizers uniformly and fairly, we repeat each setting five times in the same runtime environment but with different random seeds. The best-seen validation loss is monitored for each optimizer (for multi-fidelity optimizers, higher fidelity results are preferred over lower ones). We sort the optimizers by their best-seen results and compare their mean ranks on all the considered FedHPO problems. Following HPOBench, we use sign tests to judge (1) whether advanced methods outperform their baselines and (2) whether multi-fidelity methods outperform their single-fidelity counterparts. We refer our readers to Appendix C for more details.



(a) *BBO* **(b)** *MF* **(c)** All

Figure 4: Mean rank over time on all FedHPO problems (with FedAvg).

Results and Analysis. We show the overall results in Figure 4, and we defer detailed results to Appendix I. Overall, their eventual mean ranks do not deviate remarkably. For *BBO*, the performances of optimizers are close at the beginning but become more distinguishable along with their exploration. Ultimately, BO_{GP} has successfully sought better configurations than other optimizers. In contrast to *BBO*, *MF* optimizers perform pretty differently in the early stage, which might be rooted in the vast variance of low-fidelity function evaluations. Eventually, *HB* and *BOHB* become superior to others while achieving a very close mean rank. We consider optimizers' final performances on all the considered FedHPO problems, where, for each pair of optimizers, one may win, tie, or lose against the other. Then we can conduct sign tests to compare pairs of optimizers, where results are presented in Table 2 and Table 3. (1) Comparing these advanced optimizers with their baselines, only BO_{GP}, BO_{RF}, and *DE* win on more than half of the problems but have no significant improvement, which is inconsistent with the non-FL setting. (2) Meanwhile, no *MF* optimizers show any advantage in exploiting experience, which differs from non-FL cases. We presume the reason lies in the distribution of configurations' performances (see Figure 2). From Table 3, we draw part of the answer to RQ1 as that *MF* optimizers always outperform their corresponding single-fidelity version, which is consistent with the non-FL settings.

Table 2: P-value of a sign test for the hypothesis—these advanced methods surpass the baselines.

	BO_{GP}	BO_{RF}	BO_{KDE}	*DE*
p-value agains *RS*	0.0637	0.2161	0.1649	0.7561
win-tie-loss	13 / 0 / 7	12 / 0/ 8	7 / 0 / 13	11 / 0 / 9
	BOHB	*DEHB*	TPE_{MD}	TPE_{HB}
p-value against *HB*	0.4523	0.9854	0.2942	0.2454
win-tie-loss	7 / 0 / 13	9 / 0 / 11	9 / 0 / 11	9 / 0 / 11

Table 3: P-value of a sign test for the hypothesis—*MF* methods surpass corresponding *BBO* methods.

	HB vs. *RS*	*DEHB* vs. *DE*	*BOHB* vs. BO_{KDE}
p-value	0.1139	0.2942	0.0106
win-tie-loss	13 / 0 / 7	13 / 0 / 7	16 / 0 / 4

4.2 STUDIES ABOUT CONCURRENT EXPLORATION

To answer RQ2, we select the superior optimizers from Section 4.1 to compare with FedEx (Khodak et al., 2021). As mentioned in Section 2.2, FL allows HPO methods to take advantage of concurrent exploration, which somewhat compensates for the number of function evaluations. We are interested in methods designed regarding these characteristics of FedHPO and design this experiment to see how much concurrent exploration contributes.

Protocol. We consider the FedHPO problem <FEMNIST, CNN, FedAvg), i.e., FedAvg is applied to learn a 2-layer CNN on FEMNIST. As a full-fidelity function evaluation consumes 500 rounds on this dataset, we specify *RS*, BO_{GP}, BO_{RF}, BO_{KDE}, *HB*, and *BOHB* to limit their total budget to 2,500 (i.e., 5 times budget of a full-fidelity evaluation) in terms of *#round*. Precisely, each BBO method consists of 50 trials, each of which runs for 50 rounds. For MF optimizers, we set the η of Successive Halving Algorithm (SHA) (Jamieson & Talwalkar, 2016) to 3, the minimal budget to 9 rounds, and the max budget to 81 rounds. Then we adopt these optimizers and FedEx wrapped by them (*X+FedEx*) to optimize the design choices of FedAvg, respectively. The wrapper is responsible for determining the arms for each execution of FedEx. We consider validation loss the metric of interest, and function evaluations are conducted in the *raw* mode. We repeat each method three times and report the averaged best-seen value at the end of each trial. Meanwhile, for each considered method, we entirely run the FL course with the optimal configuration it seeks. Their averaged test accuracies are compared.

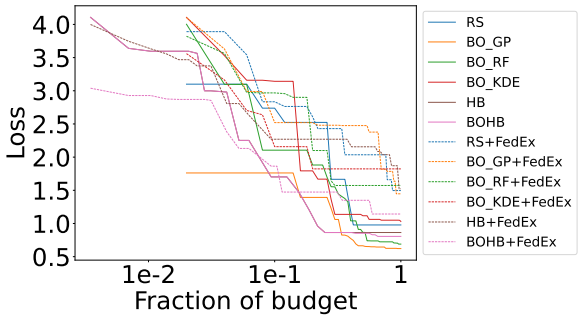


Figure 5: Mean validation loss over time.

Methods	W/O FedEx	W/ FedEx
RS	79.93 ± 2.45	82.03 ± 2.08
BO_{GP}	82.18 ± 0.94	83.20 ± 1.24
BO_{RF}	81.86 ± 1.10	82.20 ± 0.54
BO_{KDE}	81.34 ± 1.75	82.11 ± 0.46
HB	80.26 ± 2.02	82.47 ± 0.04
BOHB	79.59 ± 2.09	**84.02 ± 0.50**

Table 5: Compare the searched configurations: Mean test accuracy (%) ± standard deviation. Underline indicates improvements.

Results and Analysis. We present the results in Figure 5 and Table 5. For FedAvg, the best-seen mean validation losses of all wrapped FedEx decrease slower than their corresponding wrapper. However, their searched configurations' generalization performances are significantly better than their wrappers, which strongly confirms the effectiveness of concurrent exploration. Thus, we have a clear answer to RQ2: concurrent exploration methods significantly improve traditional methods.

5 CONCLUSION AND FUTURE WORK

In this paper, we first identify the uniqueness of FedHPO, which we ascribe to the distributed nature of FL and its heterogeneous clients. This uniqueness prevents FedHPO research from leveraging existing HPO benchmarks, which has led to inconsistent comparisons between some recently proposed methods. Hence, we suggest and implement a comprehensive, flexible, and extensible benchmark suite, FEDHPO-BENCH. We further conduct extensive HPO experiments on FEDHPO-BENCH, validating its correctness and applicability to comparing traditional and FedHPO methods. We have open-sourced FEDHPO-BENCH with an Apache-2.0 license and will actively maintain it in the future (Maintenance of FEDHPO-BENCH is discussed in Appendix A). We believe FEDHPO-BENCH can serve as the stepping stone to developing reproducible FedHPO works.

In our next step, tasks other than federated supervised learning will be incorporated. At the same time, we aim to extend FEDHPO-BENCH to include different FL settings, e.g., HPO for vertical FL (Zhou et al., 2021). Another issue the current version has not touched on is the risk of privacy leakage caused by HPO methods (Koskela & Honkela, 2020), which we should provide related metrics and testbeds in the future.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10, 2020.

N. Awad, N. Mallik, and F. Hutter. Differential evolution for neural architecture search. In *Proceedings of the 1st workshop on neural architecture search(@ICLR'20)*, 2020.

Noor Awad, Neeratyoy Mallik, and Frank Hutter. Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2147–2153, 2021.

James Bennett and Stan Lanning. The netflix prize. 2007.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, pp. 281–305, 2012.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillet. Federated bayesian optimization via thompson sampling. In *NeurIPS*, 2020.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking nas algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, Rene Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

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

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

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated machine learning*, pp. 3–33. 2019.

P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909 [cs.LG]*, 2019.

Pengfei Guo, Dong Yang, Ali Hatamizadeh, An Xu, Ziyue Xu, Wenqi Li, Can Zhao, Daguang Xu, Stephanie A. Harmon, Evrim B Turkbey, Baris I Turkbey, Bradford J. Wood, F. Patella, Elvira Stellato, Gianpaolo Carrafiello, Vishal M. Patel, and Holger R. Roth. Auto-fedrl: Federated hyperparameter optimization for multi-institutional medical image segmentation. *ArXiv*, abs/2203.06338, 2022.

Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. Coco: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, pp. 114 – 144, 2021.

Florian Hase, Matteo Aldeghi, Riley J. Hickman, Loïc M. Roch, Melodie Christensen, Elena Liles, Jason E. Hein, and Alán Aspuru-Guzik. Olympus: a benchmarking framework for noisy optimization and experiment planning. *Machine Learning: Science and Technology*, 2021.

Stephan Holly, Thomas Hiessl, Safoura Rezapour Lakani, Daniel Schall, Clemens Heitzinger, and Jana Kemnitz. Evaluation of hyperparameter-optimization approaches in an industrial federated learning system. *ArXiv*, abs/2110.08202, 2021.

Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems*, 2022.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011a.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, 2011b.

Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Thomas Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Aclib: A benchmark library for algorithm configuration. In *LION*, 2014.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Bernd Bischl Joaquin Vanschoren, Jan N. van Rijin and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, pp. 49–60, 2013.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Nina Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pp. 528–536, 2017.

Antti Koskela and Antti Honkela. Learning rate adaptation for differentially private learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 2465–2475, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Fan Lai, Yinwei Dai, Sanjay S. Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. FedScale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning (ICML)*, 2022.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, pp. 6765–6816, 2017.

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

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, pp. 429–450, 2020b.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *ICML*, pp. 6357–6368, 2021a.

Xiaoxiao Li, Meirui JIANG, Xiaofei Zhang, Michael Kamp, and Qi Dou. FedBN: Federated learning on non-IID features via local batch normalization. In *International Conference on Learning Representations*, 2021b.

Zitao Li, Bolin Ding, Ce Zhang, Ninghui Li, and Jingren Zhou. Federated matrix factorization with privacy guarantee. *Proc. VLDB Endow.*, 15:900–913, 2021c.

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. *J. Mach. Learn. Res.*, pp. 54–1, 2022.

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 Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Hesham Mostafa. Robust federated learning through representation matching and adaptive hyper-parameters, 2020.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, pp. 2825–2830, 2011.

Johann Petrak. Fast subsampling performance estimates for classification algorithm selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pp. 3–14, 2000.

Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. Yahpo gym-an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *First Conference on Automated Machine Learning (Main Track)*, 2022.

Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, pp. 93–93, 2008.

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*, pp. 148–175, 2016.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, 2013.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, pp. 341–359, 1997.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in neural information processing systems*, 2013.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

R. Turner and D. Eriksson. Bayesmark: Benchmark framework to easily compare bayesian optimization methods on real machine learning tasks, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021a.

Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4110–4120, 2022.

Zheng Wang, Xiaoliang Fan, Jianzhong Qi, Chenglu Wen, Cheng Wang, and Rongshan Yu. Federated learning with fair averaging. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 1615–1623, 2021b.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *Trans. Info. For. Sec.*, pp. 3454–3469, 2020.

Yuexiang Xie, Zhen Wang, Daoyuan Chen, Dawei Gao, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope: A flexible federated learning platform for heterogeneity. *arXiv preprint arXiv:2204.05011*, 2022.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48, 2016.

Huanle Zhang, Mi Zhang, Xin Liu, Prasant Mohapatra, and Michael DeLucia. Automatic tuning of federated learning hyper-parameters from system perspective. *arXiv preprint arXiv:2110.03061*, 2021.

Yi Zhou, Parikshit Ram, Theodoros Salonidis, Nathalie Baracaldo, Horst Samulowitz, and Heiko Ludwig. Flora: Single-shot hyper-parameter optimization for federated learning. *arXiv preprint arXiv:2112.08524*, 2021.

A MAINTENANCE OF FEDHPO-BENCH

In this section, we present our plan for maintaining FEDHPO-BENCH following Eggensperger et al. (2021).

- **Who is maintaining the benchmarking library?** FEDHPO-BENCH is developed and maintained by FedHPO-Bench team (to be de-anonymized later).
- **How can the maintainer of the dataset be contacted (e.g., email address)?** Users can reach out to the maintainer via creating issues on the Github repository with FEDHPO-BENCH label.
- **Is there an erratum?** No.
- **Will the benchmarking library be updated?** Yes, as we discussed in Section 5, we will add more FedHPO problems and introduce more FL tasks to the existing benchmark. We will track updates and Github release on the README. In addition, we will fix potential issues regularly.
- **Will older versions of the benchmarking library continue to be supported/hosted/maintained?** All older versions are available and maintained by the Github release, but limited support will be provided for older versions. Containers will be versioned and available via AliyunOSS.
- **If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?** Any contribution is welcome, and all commits to FEDHPO-BENCH must follow the guidance and regulations at `https://github.com/FedHPO-Bench/FedHPO-Bench-ICLR23/blob/main/README.md`.

B RELATED WORK

Hyperparameter Optimization (HPO). Generally, HPO is an optimization problem where the objective function is non-analytic, non-convex, and even non-differentiable. Therefore, most HPO methods solve such an optimization problem in a trial-and-error manner, with different strategies for balancing exploitation and exploration. Model-free methods such as random search (RS) (Bergstra & Bengio, 2012) and grid search query a set of initially determined hyperparameter configurations without any exploitation. Model-based methods such as Bayesian Optimization (BO) (Shahriari et al., 2016) employ a surrogate model to approximate the objective function. Methods in this line (Hutter et al., 2011a; Lindauer et al., 2022; Hutter et al., 2011b; Falkner et al., 2018b) mainly differ from each other in their surrogate model and how they determine the next query. There are also Evolutionary Algorithms (EAs) that iteratively maintain a population. We consider differential EAs (Storn & Price, 1997; Awad et al., 2020) in our experiments.

As training a deep neural network on a large-scale dataset is very costly, the full-fidelity function evaluations made by BO methods are often unaffordable in practice. Naturally, researchers consider trading off the precision of a function evaluation for its efficiency by, e.g., training fewer epochs and training on a subset of the data. Hyperband (Li et al., 2017) is a representative multi-fidelity method that calls the Successive Halving Algorithm (SHA) (Jamieson & Talwalkar, 2016) again and again with a different number of initial candidates. However, in each execution of SHA, the initial candidates are randomly sampled without any exploitation. To exploit the experience of previous SHA executions, researchers combine BO methods with Hyperband (Falkner et al., 2018a; Awad et al., 2021).

Benchmarking HPO. AutoML-related optimization benchmarks have been proved helpful for promoting fair comparisons of related methods and reproducible research works. There have been many successful examples (Hutter et al., 2014; Hansen et al., 2021; Hase et al., 2021; Turner & Eriksson, 2019; Dong & Yang, 2020; Dong et al., 2021; Gijsbers et al., 2019). Noticeably, HPO-B (Pineda-Arango et al., 2021) is highlighted by its support for benchmarking transfer-HPO methods, and HPOBench (Eggensperger et al., 2021) fills the gap of missing multi-fidelity HPO benchmarks.

However, existing HPO benchmarks mainly focus on centralized ML, yet FL, as a promising learning paradigm, has been ignored. In this paper, we identify the uniqueness of FedHPO (see Section 2.2) and implement FEDHPO-BENCH to satisfy the demand for a FedHPO benchmark suite.

Federated Learning (FL). In this paper, we restrict our discussion of FL to the "standard" scenario introduced in Section 2.2, where FedAvg (McMahan et al., 2017) is widely adopted. Fancy FL optimization algorithms, including FedProx (Li et al., 2020b) and FedOpt (Asad et al., 2020), are mainly designed to improve the convergence rate and/or better handle the non-IIDness among clients (Wang et al., 2021a). Despite these synchronous optimization algorithms, asynchronous ones (Huba et al., 2022; Xie et al., 2022) are proposed to keep a high concurrency utility.

Sometimes, learning one global model is insufficient to handle the non-IIDness, which calls for personalized FL (Kairouz et al., 2019; Wang et al., 2021a). Many popular pFL algorithms, such as FedBN (Li et al., 2021b) and Ditto (Li et al., 2021a), have been incorporated into FS (Xie et al., 2022), with their unique hyperparameters exposed. Thus, we can further extend FEDHPO-BENCH by considering FedHPO tasks of optimizing the hyperparameters of such algorithms.

FedHPO. When we consider HPO in the FL setting, as mentioned in Section 2.2, there is some uniqueness that brings in challenges while, at the same time, it can be leveraged by deliberately designed FedHPO methods. For example, in contrast to traditional HPO methods that query one configuration in each trial, FedEx (Khodak et al., 2021) maintains one policy for determining the client-side hyperparameters and independently samples each client's configuration in each communication round. Different configurations may be evaluated with the same model parameters, which is in analogy to the weight-sharing idea in neural architecture search (NAS), as summarized by the authors of FedEx. However, due to the non-IIDness among clients, clients' HPO objective functions tend to be different, where determining their configurations by only one policy might be unsatisfactory. Regarding this issue, FTS (Dai et al., 2020) can be treated as a personalized FedHPO method, where each client maintains its own policy. During the learning procedure, the clients benefit each other by sharing the policies in a privacy-preserving manner and conducting Thompson sampling.

As an emerging research topic, existing works relating to FedHPO include Fed-Tuning (Zhang et al., 2021) concerning about system-related performance, learning rate adaptation (Koskela & Honkela, 2020), FLoRA for Gradient Boosted Decision Trees (GBDT), online adaptation scheme-based method (Mostafa, 2020), Auto-FedRL (Guo et al., 2022) for RL-based hyperparameter adaptation, and an insightful comparison between local and global HPO (Holly et al., 2021). These methods can also be easily incorporated into FS, enabling FEDHPO-BENCH to benchmark them.

B.1 RELATION TO HPOBENCH

HPOBench (Eggensperger et al., 2021) is a collection of multi-fidelity HPO benchmarks, highlighted by their efficiency, reproducibility, and flexibility. These benchmarks can be accessed in either tabular, surrogate, or raw mode. On the one hand, the tabular and surrogate modes enable function evaluation without truly executing the corresponding ML algorithm and thus are efficient. On the other hand, the raw mode means execution in a docker container, which ensures reproducibility. HPOBench provides twelve families of benchmarks that correspond to different data domains, model types, fidelity spaces, etc., and thus flexible usages to validate HPO methods. This collection of HPO benchmarks can promote fair comparisons of related works and reproducible research work, so HPOBench has gained more and more attention from the community.

As pointed out in Section 2.2 that evaluating the objective function that corresponds to an FL algorithm is extremely expensive, FEDHPO-BENCH also prepares tabular and surrogate modes for users to avoid truly executing FL courses. Meanwhile, we provide the raw mode to truly execute an FL course in the docker container of FederatedScope (FS) (Xie et al., 2022).

Sharing the same modes, a question naturally arises—**is it possible to reuse HPOBench's interface for FEDHPO-BENCH?** We answer this question by discussing their commonality and the unique ingredients of FEDHPO-BENCH:

Commonality. As the code snippet in Figure 1 shows, the instantiation of a benchmark class, the "ConfigSpace" package based specification of search space, and the protocol for the interaction between an HPO method and a benchmark object are roughly consistent with HPOBench.

Uniqueness. In addition to a collection of benchmarks, FEDHPO-BENCH is flexible in terms of enabling users to tailor one benchmark to their scenarios (see Section 3.2). To this end, users are allowed to instantiate a specific benchmark object with extra optional arguments:

- *Privacy budget* with which function evaluation corresponds to the execution of NbAFL (Wei et al., 2020) instead of vanilla FedAvg. Taking the tabular mode, for example, this means looking up a privacy budget-specific table.
- *The type of fairness metric and its strength* with which FEDHPO-BENCH will consider a vector-valued objective function (i.e., client-wise results) rather than a scalar-valued objective function. Besides, the return value of calling the function evaluation will be the mean performance regularized by the specified fairness regularizer.
- *The parameter(s) for our system model* with which the execution time is estimated regarding the user's system condition. Without using a system model, FEDHPO-BENCH can provide the recorded execution time in the creation of this benchmark.

Currently, we implement the interfaces of FEDHPO-BENCH by ourselves, where the style of our interfaces is kept similar to HPOBench for the convenience of users who are familiar with HPOBench. We also provide several examples (`https://github.com/FedHPO-Bench/FedHPO-Bench-ICLR23/tree/main/demo`) to access our tabular, surrogate, and raw benchmarks by implementing HPOBench's abstract base class. As a first step, we are going to contribute more such subclasses to the repository of HPOBench so that users can access our benchmarks via HPOBench's interfaces, where flexible customization cannot be provided temporarily. In our next step, we plan to extend the interfaces of HPOBench such that the benchmarks of FEDHPO-BENCH can be accessed with our proposed flexible customization.

C HPO METHODS

As shown in Table 6, we provide an overview of the optimizers (i.e., HPO methods) we use in this paper.

Table 6: Overview of the optimizers from widely adopted libraries.

Name	Model	Packages	version
RS (Bergstra & Bengio, 2012)	-	*HPBandster*	0.7.4
BO_{GP} (Hutter et al., 2011a; Lindauer et al., 2022)	*GP*	*SMAC3*	1.3.3
BO_{RF} (Hutter et al., 2011b)	*RF*	*SMAC3*	1.3.3
BO_{KDE} (Falkner et al., 2018b)	*KDE*	*HPBandster*	0.7.4
DE (Storn & Price, 1997; Awad et al., 2020)	-	*DEHB*	git commit
HB (Li et al., 2017)	-	*HPBandster*	0.7.4
BOHB (Falkner et al., 2018a)	*KDE*	*HPBandster*	0.7.4
DEHB (Awad et al., 2021)	-	*DEHB*	git commit
TPE_{MD} (Akiba et al., 2019)	*TPE*	*Optuna*	2.10.0
TPE_{HB} (Akiba et al., 2019)	*TPE*	*Optuna*	2.10.0

C.1 BLACK-BOX OPTIMIZERS

RS (*Random search*) is a priori-free HPO method, i.e., each step of the search does not exploit the already explored configuration. The random search outperforms the grid search within a small fraction of the computation time.

$\boldsymbol{BO_{GP}}$ is a Bayesian optimization with a Gaussian process model. BO_{GP} uses a Matérn kernel for continuous hyperparameters, and a hamming kernel for categorical hyperparameters. In addition, the acquisition function is expected improvement (EI).

$\boldsymbol{BO_{RF}}$ is a Bayesian optimization with a random forest model. We set the hyperparameters of the random forest as follows: the number of trees is 10, the max depth of each tree is 20, and we use the default setting of the minimal samples split, which is 3.

$\boldsymbol{BO_{KDE}}$ is a Bayesian optimization with kernel density estimators (KDE), which is used in *BOHB* (Falkner et al., 2018a). It models objective function as $\Pr(x \mid y_{\text{good}})$ and $\Pr(x \mid y_{\text{bad}})$. We set the hyperparameters for BO_{KDE} as follows: the number of samples to optimize EI is 64, and

$1/3$ of purely random configurations are sampled from the prior without the model; the bandwidth factor is 3 to encourage diversity, and the minimum bandwidth is 1e-3 to keep diversity.

DE uses the evolutionary search approach of Differential Evolution. We set the mutation strategy to *rand1* and the binomial crossover strategy to *bin* [1]. In addition, we use the default settings for the other hyperparameters of *DE*, where the mutation factor is 0.5, crossover probability is 0.5, and the population size is 20.

C.2 MULTI-FIDELITY OPTIMIZERS

HB (*Hyperband*) is an extension on top of successive halving algorithms for the pure-exploration nonstochastic infinite-armed bandit problem. Hyperband makes a trade-off between the number of hyperparameter configurations and the budget allocated to each hyperparameter configuration. We set η to 3, which means only a fraction of $1/\eta$ of hyperparameter configurations goes to the next round.

BOHB combines *HB* with the guidance and guarantees of convergence of Bayesian optimization with kernel density estimators. We set the hyperparameter of the *BO* components and the *HB* components of *BOHB* to be the same as BO_{KDE} and *HB* described above, respectively.

DEHB combines the advantages of the bandit-based method *HB* and the evolutionary search approach of *DE*. The hyperparameter of *DE* components and *BO* components are set to be exactly the same as *DE* and *HB* described above, respectively.

$\boldsymbol{TPE_{MD}}$ is implemented in *Optuna* and uses Tree-structured Parzen Estimator (*TPE*) as a sampling algorithm, where on each trial, TPE fits two Gaussian Mixture models for each hyperparameter. One is to the set of hyperparameters with the best performance, and the other is to the remaining hyperparameters. In addition, it uses the median stopping rule as a pruner, which means that it will prune if the trial's best intermediate result is worse than the median (*MD*) of intermediate results of previous trials at the same step. We use the default settings for both *TPE* and *MD*.

$\boldsymbol{TPE_{HB}}$ is similar to TPE_{MD} described above, which uses *TPE* as a sampling algorithm and *HB* as pruner. We set the reduction factor to 3 for *HB* pruner, and all other settings use the default ones.

D DATASETS

As shown in Table 7, we provide a detailed description of the datasets we use in current FEDHPO-BENCH. For comprehensiveness, we use 16 FL datasets from 5 domains, including CV, NLP, graph, tabular, and recommendation (Xie et al., 2022; Wang et al., 2022; Eggensperger et al., 2021). Some of them are inherently real-world FL datasets, while others are simulated FL datasets split by the splitter modules of FS. Notably, the name of datasets from OpenML is the ID of the corresponding task.

FEMNIST is an FL image dataset from LEAF (Caldas et al., 2018), whose task is image classification. Following (Caldas et al., 2018), we use a subsample of FEMNIST with 200 clients, which is round 5%. And we use the default train/valid/test splits for each client, where the ratio is $60\% : 20\% : 20\%$.

CIFAR-10 (Krizhevsky et al., 2009) is from Tiny Images dataset and consists of $60{,}000$ 32×32 color images, whose task is image classification. We split images into 5 clients by latent dirichlet allocation (LDA) to produce statistical heterogeneity among these clients. We split the raw training set to training and validation sets with a ratio $4 : 1$, so that ratio of final train/valid/test splits is 66.7%:16.67%:16.67%.

SST-2 is a dataset from GLUE (Wang et al., 2018) benchmark, whose task is binary sentiment classification for sentences. We also split these sentences into 5 clients by LDA. In addition, we use the official train/valid/test splits for SST-2.

CoLA is also a dataset from GLUE benchmark, whose task is binary classification for sentences—whether it is a grammatical English sentence. We exactly follow the experimental setup in SST-2.

Cora & CiteSeer & PubMed (Sen et al., 2008; Yang et al., 2016) are three widely adopted graph datasets, whose tasks are node classification. Following FS-G (Wang et al., 2022), a community

[1] Please refer to `https://github.com/automl/DEHB/blob/master/README.md` for details.

Table 7: Statistics of the datasets used in current FEDHPO-BENCH.

Name	#Client	Subsample	#Instance	#Class	Split by
FMNIST	3,550	5%	805,263	62	Writer
CIFAR-10	5	100%	60,000	10	LDA
CoLA	5	100%	10,657	2	LDA
SST-2	5	100%	70,042	2	LDA
Cora	5	100%	2,708	7	Community
CiteSeer	5	100%	4,230	6	Community
PubMed	5	100%	19,717	3	Community
credit-g_{31}	5	100%	1,000	2	LDA
vehicle_{53}	5	100%	846	4	LDA
kc1_{3917}	5	100%	2,109	2	LDA
blood-transf..$_{10101}$	5	100%	748	2	LDA
$\text{Australian}_{146818}$	5	100%	690	2	LDA
car_{146821}	5	100%	1,728	4	LDA
segment_{146822}	5	100%	2,310	7	LDA
FedNetflix	480,189	100%	≈100,000,000	5	User
Twitter	660,120	0.5%	1,600,498	2	User

splitter is applied to each graph to generate five subgraphs for each client. We also split the nodes into train/valid/test sets, where the ratio is 60%:20%:20%.

Tabular datasets are consist of 7 tabular datasets from OpenML (Bischl et al., 2017), whose task ids (name of source data) are 31 (**credit-g**), 53 (**vehicle**), 3917 (**kc1**), 10101 (**blood-transfusion-service-center**), 146818 (**Australian**), 146821 (**car**) and 146822 (**segment**). We split each dataset into 5 clients by LDA, respectively. In addition, we set the ratio of train/valid/test splits to 80%:10%:10%.

FedNetflix is a recommendation dataset from The Netflix Prize (Bennett & Lanning, 2007), whose task is to predict the ratings between users and movies. Netflix consists of around 100 million ratings between 480,189 users and 171,770 movies. We split the Netflix dataset into 480,189 clients by users. In addition, we set the ratio of train/valid/test splits to 80%:10%:10%.

Twitter is a sentiment analysis dataset from LEAF (Caldas et al., 2018), whose task is to determine sentiment of sentences. We use a subsample of Twitter with around 3300 clients. Moreover, we use the train/valid/test splits for each client, where the ratio is $80\% : 10\% : 10\%$. It is worth noting that the average number of samples is only 1.94, which means some clients do not have valid split or test split, and we evaluate the performance on a shared test split merged by all clients.

E SYSTEM MODEL

We provide default parameters of our system model, including c, B, and $T^{(\text{server})}$, based on observations collected from the executions in Section 3.2. Users are allowed to specify these parameters according to their scenarios or other system statistic providers, e.g., estimating the computation time of stragglers by sampling from FedScale (Lai et al., 2022). The estimated parameters are $B_{\text{down}} \sim 0.75\text{MB/secs}, B_{\text{up}} \sim 0.25\text{MB/secs}$ (Lai et al., 2022; Wang et al., 2021a). The default value of $c(f, \lambda, b)$ is obtained by averaging the recorded client-wise time costs in trials of tabular mode benchmarks. And we set the default value of the maximum number of connections in calculating $\alpha(N)$ to 65535.

To implement our system model, we use the following proposition to calculate Eq. equation 1 analytically.

Proposition 1. *When the computation time of clients are identically independently distributed, following an exponential distribution Exp$(\cdot|\frac{1}{c})$, then the expected time for the straggler of N uniformly sampled clients is $\sum_{i=1}^{N} \frac{c}{i}$.*

What we need to calculate is the expected maximum of i.i.d. exponential random variable. Proposition 1 states that, for N exponential variables independently drawn from Exp$(\cdot|\frac{1}{c})$, the expectation is

$\sum_{i=1}^{N} \frac{c}{i}$. There are many ways to prove this useful proposition, and we provide a proof starting from studying the minimum of the exponential random variables.

Proof. According to the Wikipedia page about exponential distribution, the minimum of them obeys $\text{Exp}(\cdot|\frac{N}{c})$, where their λs correspond to $\frac{1}{c}$ here. Denoting the i-th minimum of them by T_i, $T_1 \sim \text{Exp}(\cdot|\frac{N}{c})$ and T_N is what we are interested in. Meanwhile, it is well known that exponential distribution is memoryless, namely, $\Pr(X > s + t|X > s) = \Pr(X > t)$. Thus, $T_2 - T_1$ obeys the same distribution as the minimum of $N - 1$ such random variables, that is to say, $T_2 - T_1$ is a random variable drawn from $\text{Exp}(\cdot|\frac{N-1}{c})$. Similarly, $T_{i+1} - T_i$ $\text{Exp}(\cdot|\frac{N-i}{c})$, $i = 1, \ldots, N-1$. Thus, we have:

$$\mathbb{E}[T_N] = \mathbb{E}[T_1 + \sum_{i=1}^{N-1}(T_{i+1} - T_i)] = \frac{c}{N} + \sum_{i=1}^{N-1} \frac{c}{N-i} = \sum_{i=1}^{N} \frac{c}{i}, \tag{2}$$

which concludes this proof. □

It is worth noting that we provide several optional system models. For example, for point-to-point transport protocols, T_{comm} should contain the time the server sends the model to each client.

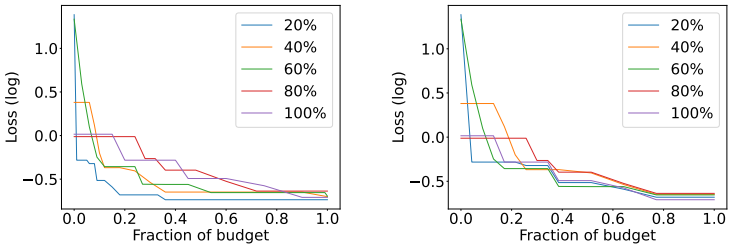
F Details of the Implementations of FedEx and FTS

We implemented these two FedHPO methods with the general algorithmic view shown in Figure 3.

In FedEx (Khodak et al., 2021), λ_is are independently sampled from π, and the aggregation operator "aggr_p" is exponential gradient descent. In FTS (Dai et al., 2020), the broadcasted policy $\pi^{(t)}$ is the samples drawn from all clients' posterior beliefs. The synchornoous operator "sync_p" can be regarded as mixing Gaussian process (GP) models. The update operator "update_p" corresponds to updating local GP model. Then a sample drawn from local GP posterior belief is regarded as $\pi_i^{(t+1)}$ and uploaded. Finally, the aggregation operator "aggr_p" is packing received samples together.

G Studies about the New Fidelity

In FL, a larger *client_sample_rate* leads to a minor variance of the aggregated model in each round, which is believed to need less *#round* for convergence and to perform better. Therefore, we tend to set the *client_sample_rate* as close to 1 as possible. However, according to our system model in Section 3.2, a large *client_sample_rate* leads to an increase in latency ($\alpha(N)$), which makes the communication cost higher. To answer RQ3, we use tabular mode and study the trade-off between these two fidelity dimensions: *client_sample_rate* and *#round*. We simulate two distinct system conditions by specifying different parameters for our system model.



(a) With bad network status **(b)** With good network status

Figure 6: Performances of different *client_sample_rate* under different system conditions.

Protocol. We compare the performance of *HB* with different *client_sample_rate*s to learn a 2-layer CNN with 2,048 hidden units on FEMNIST. To simulate a system condition with bad network status, we set the upload bandwidth B_{up} to 0.25MB/second and the download bandwidth B_{down} to 0.75MB/second (Wang et al., 2021a). As for good network status, we set the upload bandwidth B_{up} to 0.25GB/second and the download bandwidth $B_{(\text{down})}$ to 0.75GB/second. In both cases, we consider

different computation overhead so that it is negligible and significant, respectively. As for the rest settings, we largely follow that in Section 4.1.

Results and Analysis. We have an answer to RQ3: with the same time budget, the FL procedure with a lower *client_sample_rate* achieves a better result than higher *client_sample_rate* with the bad network status. In comparison, that with a higher *client_sample_rate* achieves a better result than lower *client_sample_rate* in the good network status. In conclusion, this study suggests a best practice of pursuing a more economic accuracy-efficiency trade-off by balancing *client_sample_rate* with *#round*, w.r.t. the system condition. Better choices tend to achieve more economical accuracy-efficiency trade-offs for FedHPO.

H DETAILS ON FEDHPO-BENCH BENCHMARKS

FEDHPO-BENCH consists of serveral categories of benchmarks on the different datasets (see Appendix D) with three modes. If not specified, we use the model as the name of the benchmark in cross-silo scenario. In this part, we provide more details about how we construct the FedHPO problems provided by current FEDHPO-BENCH and the three modes to interact with them.

H.1 CATEGORY

We categorize our benchmarks by model types. Each benchmark is designed to solve specific FL HPO problems on its data domain, wherein CNN benchmark on CV, BERT benchmark on NLP, GNN benchmark on the graphs, and LR & MLP benchmark on tabular data. All benchmarks have several hyperparameters on configuration space and two on fidelity space, namely sample rate of FL and FL round. And the benchmarks support several FL algorithms, such as FedAvg and FedOpt.

CNN benchmark learns a two-layer CNN with 2048 hidden units on FEMNIST and 128 hidden units on CIFAR-10 with five hyperparameters on configuration space that tune the batch size of the dataloader, the weight decay, the learning rate, the dropout of the CNN models, and the step size of local training round in client each FL communication round. The tabular and surrogate mode of the CNN benchmark only supports FedAvg due to our limitations in computing resources for now, but we will update FEDHPO-BENCH with more results as soon as possible.

BERT benchmark fine-tunes a pre-trained language model, BERT-Tiny, which has two layers and 128 hidden units, on CoLA and SST-2. The BERT benchmark also has five hyperparameters on configuration space which is the same as CNN benchmark. In addition, the BERT benchmark support FedAvg and FedOpt with all three mode.

GNN benchmark learns a two-layer GCN with 64 hidden units on Cora, CiteSeer and PubMed. The GNN benchmark has four on hyperparameters configuration space that tune the weight decay, the learning rate, the dropout of the GNN models, and the step size of local training round in client each FL communication round. The GNN benchmark support FedAvg and FedOpt with all three mode.

LR benchmark learns an lr on seven tasks from OpenML, see Appendix D for details. The LR benchmark has four on hyperparameters configuration space that tune the batch size of the dataloader, the weight decay, the learning rate, and the step size of local training round in client each FL communication round. The LR benchmark support FedAvg and FedOpt with all three mode.

MLP benchmark's the vast majority of settings are the same as LR benchmark. But in particular, we add depth and width of the MLP to search space in terms of model architecture. The MLP benchmark also support FedAvg and FedOpt with all three mode.

Cross-device. In cross-device scenarios, there can be large number of clients in total, but only a few participate in each communication round. This benchmark contains two datasets, Twitter and FedNetflix. We use a bag of words model with LR and tune the *learning_rate*, *weight_decay*, and *step_size* of local training round in Twitter. As for FedNetflix, we tune an HMFNet (Li et al., 2021c) in the *learning_rate*, *batch_size*, and *step_size* of local training round. Due to the time limit, the results FedNetflix is incomplete, and we present the ECDF of Twitter in Figure 8.

Table 8: The search space of our benchmarks, where continuous search spaces are discretized into several bins under the tabular mode.

Benchmark		Name	Type	Log	#Bins	Range
CNN	Client	*batch_size*	int	×	-	{16, 32, 64}
		weight_decay	float	×	4	[0, 0.001]
		dropout	float	×	2	[0, 0.5]
		step_size	int	×	4	[1, 4]
		learning_rate	float	✓	10	[0.01, 1.0]
	Server	*momentum*	float	×	2	[0.0, 0.9]
		learning_rate	float	×	3	[0.1, 1.0]
	Fidelity	*client_sample_rate*	float	×	5	[0.2, 1.0]
		round	int	×	250	[1, 500]
BERT	Client	*batch_size*	int	×	-	{8, 16, 32, 64, 128}
		weight_decay	float	×	4	[0, 0.001]
		dropout	float	×	2	[0, 0.5]
		step_size	int	×	4	[1, 4]
		learning_rate	float	✓	10	[0.01, 1.0]
	Server	*momentum*	float	×	2	[0.0, 0.9]
		learning_rate	float	×	3	[0.1, 1.0]
	Fidelity	*client_sample_rate*	float	×	5	[0.2, 1.0]
		round	int	×	40	[1, 40]
GNN	Client	*weight_decay*	float	×	4	[0, 0.001]
		dropout	float	×	2	[0, 0.5]
		step_size	int	×	8	[1, 8]
		learning_rate	float	✓	10	[0.01, 1.0]
	Server	*momentum*	float	×	2	[0.0, 0.9]
		learning_rate	float	×	3	[0.1, 1.0]
	Fidelity	*client_sample_rate*	float	×	5	[0.2, 1.0]
		round	int	×	500	[1, 500]
LR	Client	*batch_size*	int	✓	7	[4, 256]
		weight_decay	float	×	4	[0, 0.001]
		step_size	int	×	4	[1, 4]
		learning_rate	float	✓	6	[0.00001, 1.0]
	Server	*momentum*	float	×	2	[0.0, 0.9]
		learning_rate	float	×	3	[0.1, 1.0]
	Fidelity	*client_sample_rate*	float	×	5	[0.2, 1.0]
		round	int	×	500	[1, 500]
MLP	Client	*batch_size*	int	✓	7	[4, 256]
		weight_decay	float	×	4	[0, 0.001]
		step_size	int	×	4	[1, 4]
		learning_rate	float	✓	6	[0.00001, 1.0]
		depth	int	×	3	[1, 3]
		width	int	✓	7	[16, 1024]
	Server	*momentum*	float	×	2	[0.0, 0.9]
		learning_rate	float	×	3	[0.1, 1.0]
	Fidelity	*client_sample_rate*	float	×	5	[0.2, 1.0]
		round	int	×	500	[1, 500]

H.2 MODE

Following HPOBench (Eggensperger et al., 2021), FEDHPO-BENCH provides three different modes for function evaluation: the tabular mode, the surrogate mode, and the raw mode. The valid input hyperparameter configurations and the speed of acquiring feedback vary from mode to mode. Users can choose the desired mode according to the purposes of their experiments.

Tabular mode. The idea is to evaluate the performance of many different hyperparameter configurations in advance so that users can acquire their results immediately. For efficient function evaluation, we implement the tabular mode of FEDHPO-BENCH by running the FL algorithms configured by the grid search space in advance from our original search space (see Table 8). For hyperparameters whose original search space is discrete, we just preserve its original one. As for continuous ones, we discretize them into several bins (also see Table 8 for details). To ensure that the results are reproducible, we execute the FL procedure in the Docker container environment. Each specific configuration λ is repeated three times with different random seeds, and the resulted performances, including loss, accuracy and f1-score under train/validation/test splits, are averaged and adopted as the results of $f(\lambda)$. Users can choose the desired metric as the output of the black-box function via FEDHPO-BENCH's APIs. Besides, we provide not only the results of $f(\lambda)$ (i.e., that with full-fidelity) but also results of $f(\lambda, b)$, where b is enumerated across different *#round* and different *client_sample_rate*. Since executing function evaluation is much more costly in FL than traditional centralized learning, such lookup tables are precious. In creating them, we spent about two months of computation time on six machines, each with four Nvidia V100 GPUs. Now we make them publicly accessible via the tabular mode of FEDHPO-BENCH.

Surrogate mode.

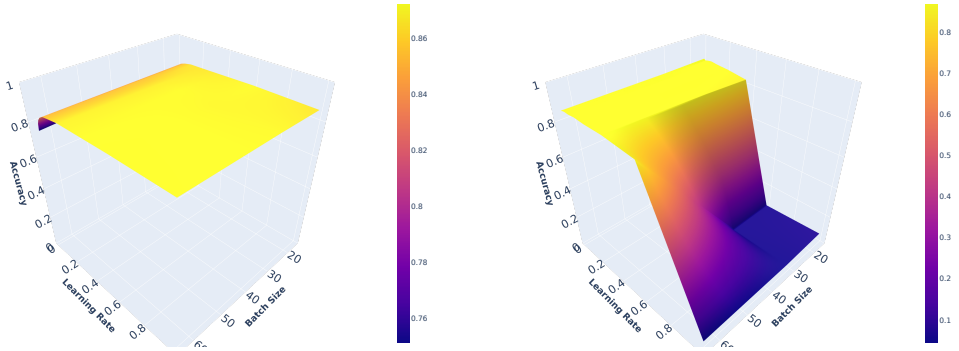
As tabular mode has discretized the original search space and thus cannot respond to queries other than the grids, we train random forest models on these lookup tables, i.e., $\{(\lambda, b), f(\lambda, b))\}$. These models serve as a surrogate of the functions to be optimized and can answer any query λ by simply making an inference. Specifically, we conduct 10-fold cross-validation to train and evaluate random forest models (implemented in scikit-learn (Pedregosa et al., 2011)) on the tabular data. Meanwhile, we search for suitable hyperparameters for the random forest models with the number of trees in $\{10, 20\}$ and the max depth in $\{10, 15, 20\}$. The mean absolute error (MAE) of the surrogate model w.r.t. the true value is within an acceptable threshold. For example, in predicting the true average loss on the CNN benchmark, the surrogate model has a training error of 0.00609 and a testing error of 0.00777. In addition to the off-the-shelf surrogate models we provide, FEDHPO-BENCH offers tools for users to build brand-new surrogate models. Meanwhile, we notice the recent successes of neural network-based surrogate, e.g., YAHPO Gym (Pfisterer et al., 2022), and we will also try it in the next version of FEDHPO-BENCH.

Raw mode. Both of the above modes, although they can respond quickly, are limited to pre-designed search space. Thus, we introduce raw mode to FEDHPO-BENCH, where user-defined search spaces are allowed. Once FEDHPO-BENCH's APIs are called with specific hyperparameters, a containerized and standalone FL procedure (supported by FS) will be launched. It is worth noting that although we use standalone simulation to eliminate the communication cost, raw mode still consumes much more computation cost than tabular and surrogate modes.

H.3 DATA ANALYTICS

H.3.1 LANDSCAPES ON ML-RELATED HYPERPARAMETERS

To study the loss landscape of the federated learning method (FedAvg) and non-federated method (Isolated), we consider *learning_rate* and *batch_size*, the hyperparameters of the ML algorithm, as the coordinate axis to build the loss landscapes. We fix other ML-related hyperparameters *weight_decay* to 0.0, *dropout* to 0.5 for both FedAvg and Isolated, which is the best configuration chosen from the tabular benchmark <CNN, FEMNIST, FedAvg> under 1.0 *client_sample_rate*. As the landscapes shown in Figure 7 with learning rate in [0.01, 1.0] and batch size in 16, 32, 64, we observe that the FedAvg with a higher learning rate achieves better results, while the non-federated method (Isolated) prefer a lower learning rate.



(a) FedAvg **(b)** Isolated

Figure 7: Landscape with the hyperparameters of the ML algorithm on FEMNIST.

H.3.2 Trends in Different Privacy Budgets

We extend the tabular benchmarks with different levels of privacy budgets in FEMNIST and Cora. To explore the trends of optimal configurations under different privacy budgets, we adopt NbAFL (Wei et al., 2020) with $\epsilon \in \{1, 10, 20\}$. We observe that the best configuration varies under different levels of privacy budgets in and Cora, as shown in Table 10. Under different privacy budgets, a large *step_size* all leads to a good performance. However, when the noise is intense, a higher *learning_rate* is preferred, while a lower *learning_rate* will perform better when the noise is weak.

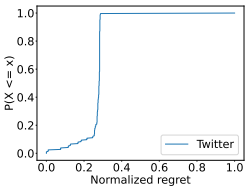


Figure 8: ECDF on Twitter.

ϵ	learning_rate	weight_decay	dropout	step_size	Test Acc. (%)
1	1.0	0.001	0.5	7	63.87 ± 6.38
10	0.59948	0.0	0.5	6	87.02 ± 1.16
20	0.59948	0.001	0.5	6	87.30 ± 0.54

Table 10: Best configuration with different levels of privacy budgets in Cora.

H.3.3 Trends with Different Regularity in FedProx

To extend the tabular benchmarks with more FL algorithms, we adopt FedProx (Li et al., 2020b) to GNN benchmark. Based on Table 8, we tune the server-side hyperparameters mu, the coefficient of the regular term, in {0.1, 1.0, 5.0} to study the trends with different regularity in FedProx. We show the landscape in Figure 9 with learning rate in [0.01, 1.0] and mu in [0.1, 5.0], and we observe that when the learning rate is low, the effect of mu has little impact on the accuracy; however, when the learning rate is large, the increase of mu can seriously damage the accuracy.

H.3.4 Errors of Surrogate Benchmarks

As we mentioned in Section H.2, we report the regression error of training surrogate model in Table 12. Meanwhile, we present the mean rank over time of optimizers with surrogate modes in Figure 20 and Figure 21. Compared to the results of tabular modes in 11 and 12, BO_{GP} shows good performance in both modes, while Random Search does not. This show the consistent performance of the same optimizer when it interplays with surrogate and tabular benchmarks.

H.3.5 Variance of Different Sample Rate

As we build our tabular benchmark from FL courses executed in docker images provided by FS, we can fully reproduce all the results given the same random seed in raw mode. Other than that, to study

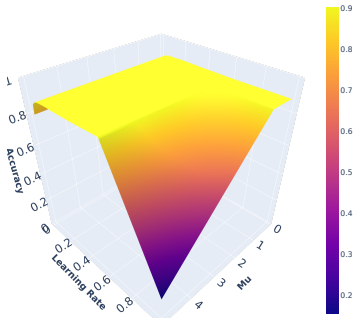


Figure 9: Landscape with the different regularity of FedProx on Cora.

Model	Dataset	Algo.	Train MAE	Test MAE
CNN	FEMNIST	FedAvg	0.00609	0.00777
BERT	CoLA	FedAvg	0.04724	0.05454
		FedOpt	0.02426	0.02959
	SST2	FedAvg	0.02597	0.03227
		FedOpt	0.02802	0.03166
GNN	Cora	FedAvg	0.04702	0.04839
		FedOpt	0.05703	0.05893
	CiteSeer	FedAvg	0.01334	0.01381
		FedOpt	0.01652	0.01717
	PubMed	FedAvg	0.04042	0.04148
		FedOpt	0.04816	0.05699

Table 12: The regression error of surroagte models.

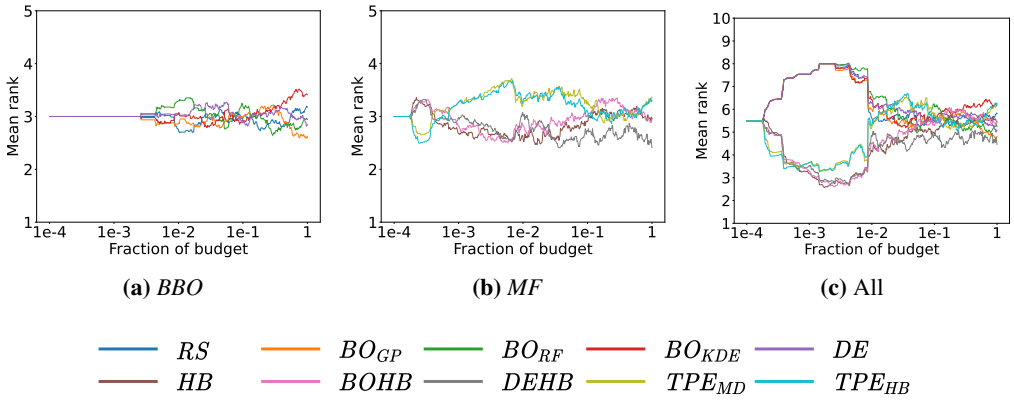
the noise of different federated optimization, we analyze the variance of validation loss with 500 rounds under different sample rates in FEMNIST. And the mean standard deviation validation loss is {1.945e-2, 1.7e-2, 1.728e-2, 1.715e-2, 1.43e-2} with sample rate {0.2, 0.4, 0.6, 0.8, 1.0}, which shows that the higher sample rate tends to have lower variance. The reason is apparent: the lower the sampling rate, the more inconsistent the set of clients sampled during the training process leads to this error.

I MORE RESULTS

In this section, we show the detailed experimental results of the optimizers on FEDHPO-BENCH benchmarks under different modes. We firstly report the averaged best-seen validation loss, from which the mean rank over time for all optimizers can be deduced. Due to time and computing resource constraints, we do not have a complete experimental result of the raw mode, which we will supplement as soon as possible.

I.1 TABULAR MODE

Following Section 4.1, we show the overall mean rank overtime on all FedHPO problems with FedOpt, whose pattern is similar to that of FedAvg in Figure 4. Then, we report the final results with FedAvg and FedOpt in Table 13 and 14, respectively. Finally, we report the mean rank over time in Figure 11-19. Due to time and computing resource constraints, the results on CNN benchmark are incomplete (lacking that with FedOpt), which we will supplement as soon as possible.



(a) *BBO* **(b)** *MF* **(c)** All

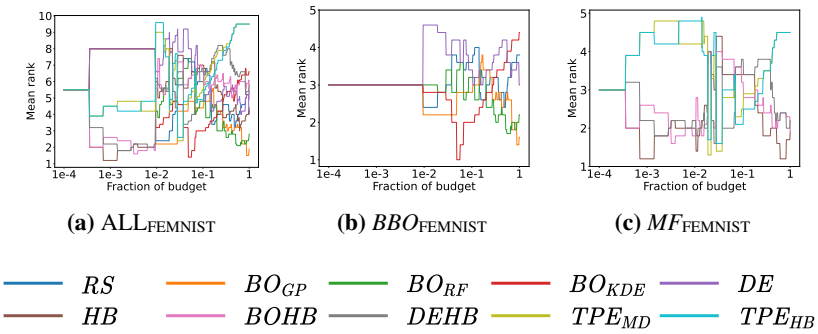
Figure 10: Mean rank over time on all FedHPO problems (with FedOpt).

Table 13: Final results of the optimizers on tabular mode with FedAvg (lower is better).

benchmark	*RS*	BO_{GP}	BO_{RF}	BO_{KDE}	*DE*	*HB*	*BOHB*	*DEHB*	TPE_{MD}	TPE_{HB}
$\text{CNN}_{\text{FEMNIST}}$	0.4969	0.4879	0.4885	0.5004	0.4928	0.4926	0.4945	0.498	0.5163	0.5148
$\text{BERT}_{\text{SST-2}}$	0.435	0.4276	0.4294	0.4334	0.437	0.4311	0.4504	0.4319	0.4341	0.4251
$\text{BERT}_{\text{CoLA}}$	0.6151	0.6148	0.6141	0.6133	0.6143	0.6143	0.6168	0.6178	0.6158	0.6146
GNN_{Cora}	0.3265	0.3258	0.326	0.3347	0.3267	0.3324	0.3288	0.3225	0.3241	0.3249
$\text{GNN}_{\text{CiteSeer}}$	0.6469	0.6442	0.6499	0.6442	0.6453	0.6387	0.6425	0.6452	0.6324	0.6371
$\text{GNN}_{\text{PubMed}}$	0.5262	0.5146	0.5169	0.5311	0.5001	0.5006	0.5194	0.4934	0.506	0.5044
LR_{31}	0.6821	0.6308	0.6382	0.6385	0.667	0.6492	0.6461	0.6145	0.7228	0.758
LR_{53}	1.6297	1.7288	1.6116	1.7142	1.6062	1.5765	1.5634	1.4755	1.5506	1.5506
LR_{3917}	1.8892	1.7561	1.7186	2.4271	1.7519	3.948	1.6384	3.1183	2.1344	2.6576
LR_{10101}	0.548	0.5483	0.5482	0.5487	0.5481	0.5504	0.5505	0.5516	0.5483	0.5487
LR_{146818}	0.5294	0.5291	0.5295	0.5289	0.5291	0.5292	0.529	0.5293	0.5328	0.5387
LR_{146821}	0.4733	0.464	0.4722	0.4843	0.4971	0.4678	0.4747	0.4707	0.4792	0.4688
LR_{146822}	0.4581	0.4481	0.4505	0.4731	0.4587	0.4478	0.4446	0.4304	0.4376	0.4419
MLP_{31}	0.5899	0.5891	0.5808	0.5904	0.5925	0.5921	0.5929	0.593	0.593	0.593
MLP_{53}	0.7795	0.7373	0.7849	0.8215	0.8068	0.769	0.7577	0.8173	0.9491	1.0567
MLP_{3917}	0.3863	0.3937	0.3858	0.3958	0.383	0.3895	0.3911	0.4084	0.3979	0.3988
MLP_{10101}	0.4054	0.4217	0.4361	0.4162	0.418	0.4137	0.4152	0.4102	0.4522	0.4352
MLP_{146818}	0.5089	0.4997	0.5125	0.5112	0.5138	0.5009	0.5199	0.5039	0.5392	0.54
MLP_{146821}	0.184	0.1251	0.155	0.1769	0.1851	0.1561	0.1683	0.1572	0.1654	0.1761
MLP_{146822}	0.2839	0.2892	0.317	0.3586	0.2928	0.2927	0.2823	0.2549	0.2745	0.2755

Table 14: Final results of the optimizers on tabular mode with FedOpt (lower is better).

benchmark	*RS*	BO_{GP}	BO_{RF}	BO_{KDE}	*DE*	*HB*	*BOHB*	*DEHB*	TPE_{MD}	TPE_{HB}
$\text{BERT}_{\text{SST-2}}$	0.441	0.4325	0.4301	0.4463	0.4351	0.4403	0.4295	0.4285	0.4293	0.4332
$\text{BERT}_{\text{CoLA}}$	0.616	0.616	0.6141	0.6137	0.6159	0.6154	0.6157	0.6176	0.6172	0.6168
GNN_{Cora}	0.3264	0.3235	0.3268	0.3322	0.3256	0.3245	0.3347	0.3254	0.3405	0.3361
$\text{GNN}_{\text{CiteSeer}}$	0.6483	0.6517	0.6497	0.6535	0.6458	0.6442	0.6543	0.6463	0.6488	0.6495
$\text{GNN}_{\text{PubMed}}$	0.4777	0.4426	0.4718	0.4943	0.4318	0.4559	0.4699	0.4318	0.4368	0.4402
LR_{31}	0.7358	0.6831	0.6849	0.8152	0.7085	0.6772	0.6877	0.6385	0.8652	0.7044
LR_{53}	1.7838	1.5609	1.5241	1.5116	1.6208	1.6045	1.7236	1.3488	1.6654	1.7978
LR_{3917}	2.254	2.0316	2.3952	1.9788	2.6261	2.3472	2.5452	2.3144	3.2131	2.0291
LR_{10101}	0.5533	0.55	0.5505	0.5509	0.549	0.5504	0.5476	0.5522	0.5612	0.8567
LR_{146818}	0.511	0.506	0.5034	0.5133	0.5007	0.5032	0.5086	0.4974	0.4983	0.5104
LR_{146821}	0.4017	0.3599	0.4121	0.4134	0.4079	0.395	0.398	0.3902	0.4447	0.4625
LR_{146822}	0.3972	0.4211	0.4037	0.4442	0.4075	0.4131	0.4008	0.3916	0.3878	0.3871
MLP_{31}	0.5912	0.5914	0.5912	0.5912	0.5918	0.5923	0.5921	0.5911	0.5921	0.5921
MLP_{53}	0.9096	0.8166	0.8111	0.8872	0.8546	1.0163	0.8565	0.9849	1.1276	1.0952
MLP_{3917}	0.3798	0.3937	0.3862	0.3871	0.3867	0.4109	0.4262	0.3812	0.4003	0.4003
MLP_{10101}	0.4219	0.4141	0.4197	0.4111	0.4303	0.4145	0.4256	0.4215	0.4502	0.4502
MLP_{146818}	0.4943	0.4913	0.5022	0.5023	0.4884	0.4995	0.5046	0.4921	0.4978	0.4861
MLP_{146821}	0.1169	0.0836	0.0915	0.1674	0.1079	0.0891	0.1389	0.0838	0.1051	0.1194
MLP_{146822}	0.2963	0.2914	0.2705	0.3025	0.2779	0.2759	0.2621	0.2549	0.257	0.2518



(a) $\text{ALL}_{\text{FEMNIST}}$ **(b)** BBO_{FEMNIST} **(c)** MF_{FEMNIST}

RS BO_{GP} BO_{RF} BO_{KDE} *DE*
HB *BOHB* *DEHB* TPE_{MD} TPE_{HB}

Figure 11: Mean rank over time on CNN benchmark (FedAvg).

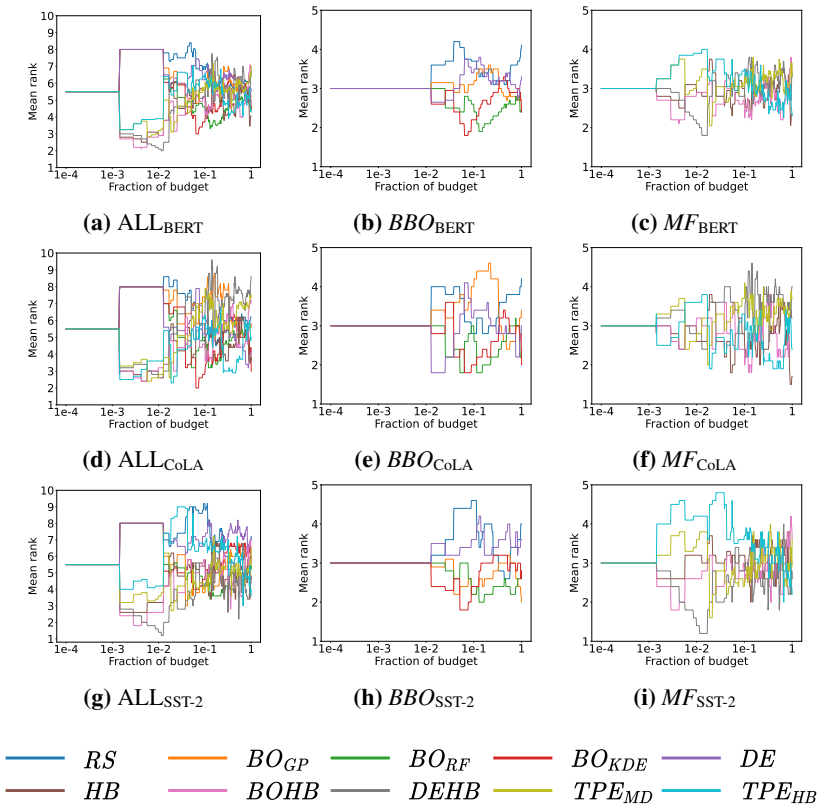


Figure 12: Mean rank over time on BERT benchmark (FedAvg).

I.2 SURROGATE MODE

We report the the final results with FedAvg on FEMNIST and BERT benchmarks in Table 15. Then we present the mean rank over time of the optimizers in Figure 20 and Figure 21.

Table 15: Final results of the optimizers in surrogate mode (lower is better).

benchmark	*RS*	BO_{GP}	BO_{RF}	BO_{KDE}	*DE*	*HB*	*BOHB*	*DEHB*	TPE_{MD}	TPE_{HB}
CNN$_{\text{FEMNIST}}$	0.0508	0.0478	0.0514	0.0492	0.0503	0.0478	0.048	0.0469	0.0471	0.0458
BERT$_{\text{SST-2}}$	0.4909	0.4908	0.4908	0.4908	0.4908	0.4908	0.4908	0.4917	0.4908	0.4908
BERT$_{\text{CoLA}}$	0.5013	0.4371	0.4113	0.487	0.444	0.4621	0.4232	0.4204	0.3687	0.3955

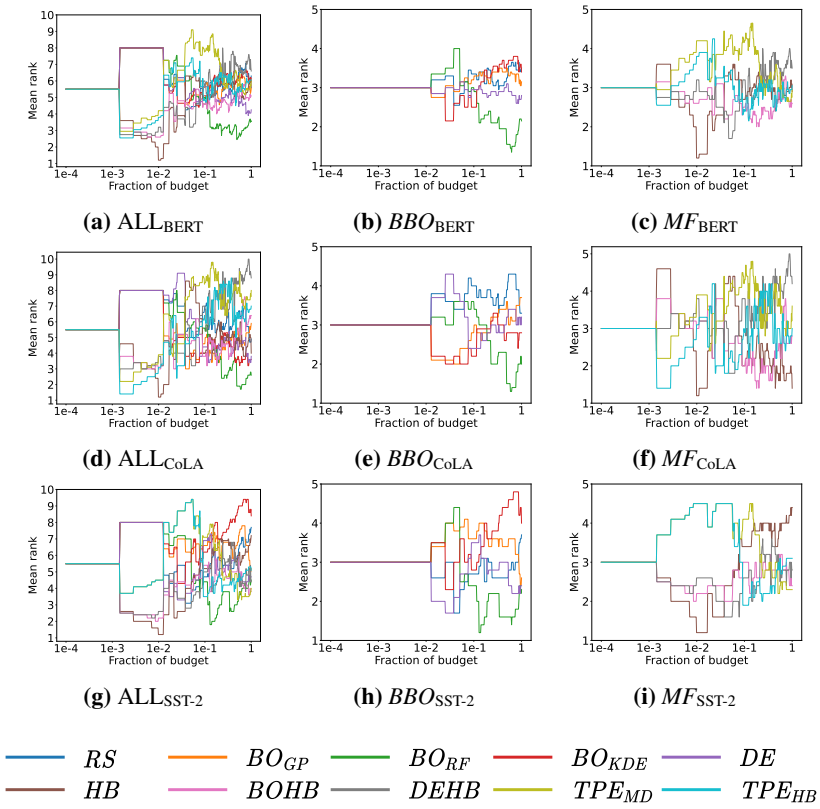


Figure 13: Mean rank over time on BERT benchmark (FedOpt).

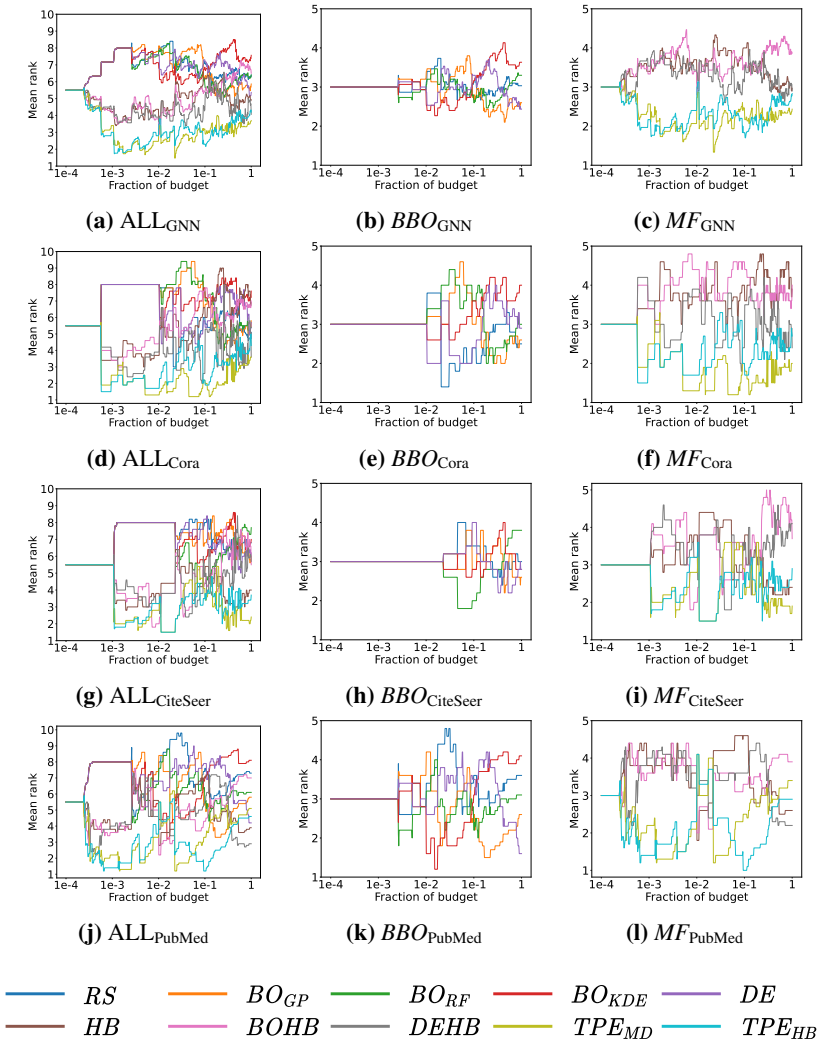


Figure 14: Mean rank over time on GNN benchmark (FedAvg).

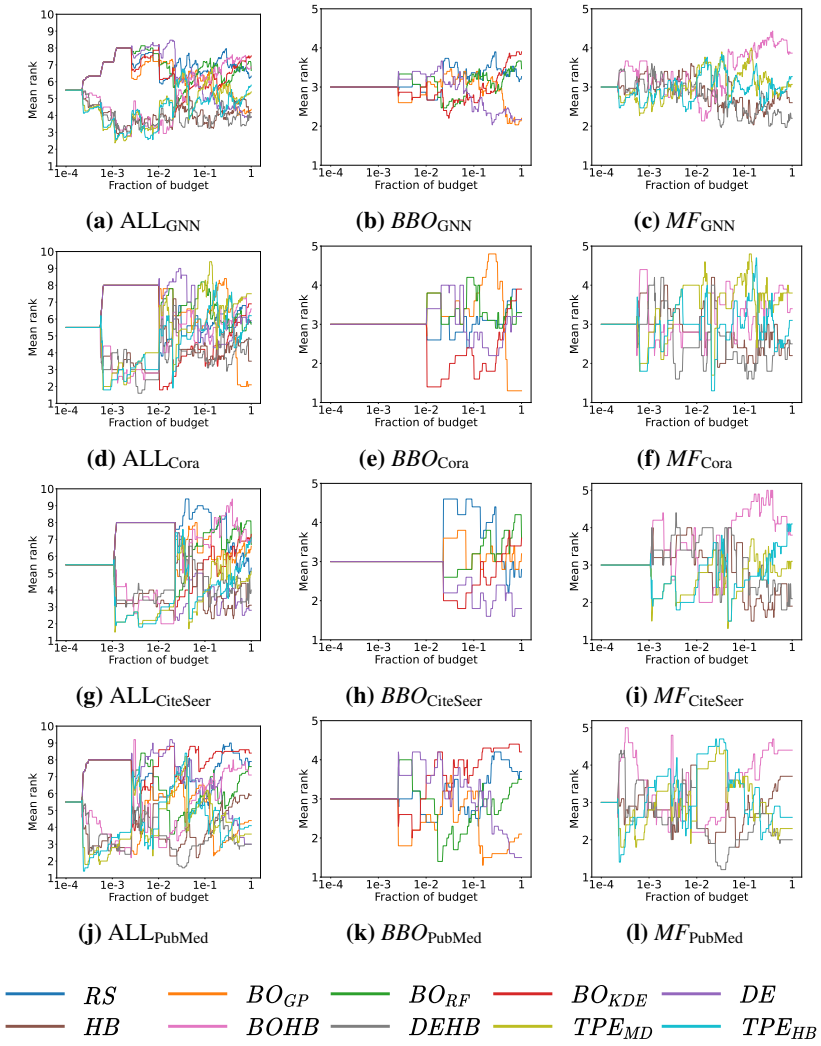
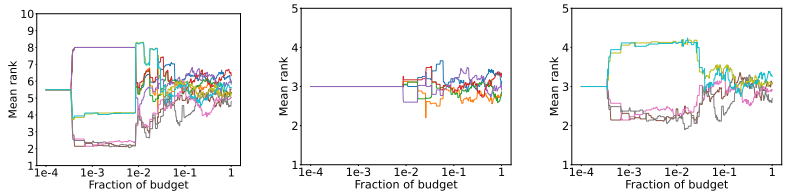
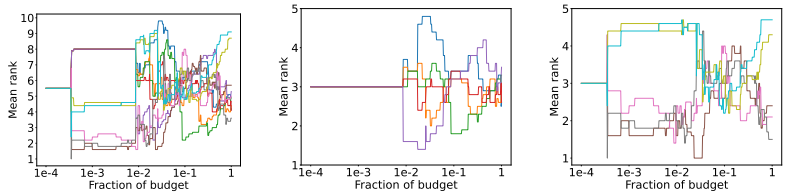


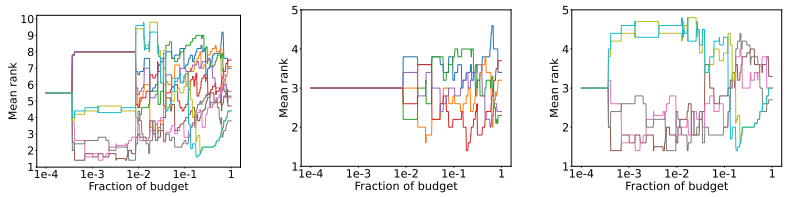
Figure 15: Mean rank over time on GNN benchmark (FedOpt).



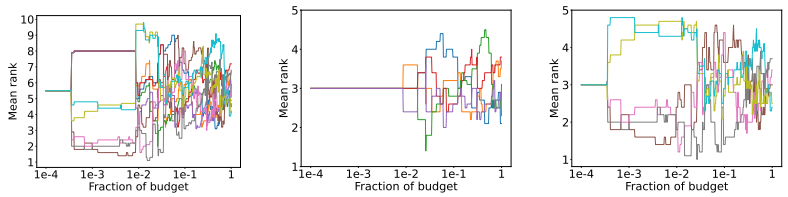
(a) ALL_{LR} **(b)** BBO_{LR} **(c)** MF_{LR}



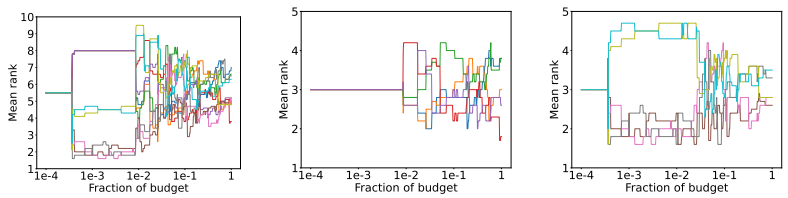
(d) $\text{ALL}_{31\text{OpenML}}$ **(e)** $BBO_{31\text{OpenML}}$ **(f)** $MF_{31\text{OpenML}}$



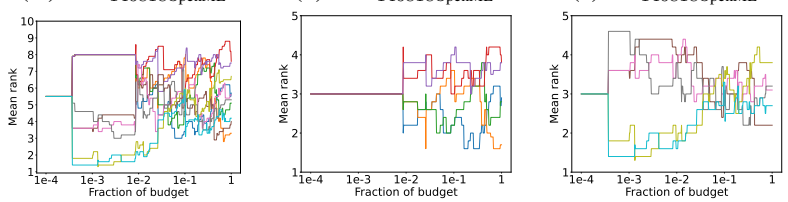
(g) $\text{ALL}_{53\text{OpenML}}$ **(h)** $BBO_{53\text{OpenML}}$ **(i)** $MF_{53\text{OpenML}}$



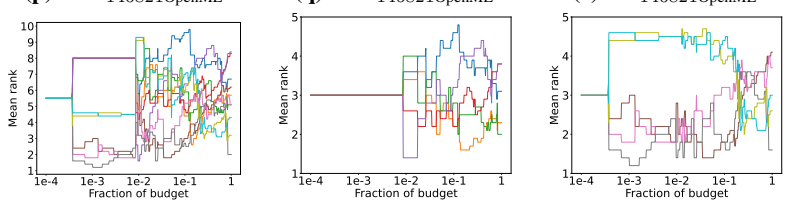
(j) $\text{ALL}_{10101\text{OpenML}}$ **(k)** $BBO_{10101\text{OpenML}}$ **(l)** $MF_{10101\text{OpenML}}$



(m) $\text{ALL}_{146818\text{OpenML}}$ **(n)** $BBO_{146818\text{OpenML}}$ **(o)** $MF_{146818\text{OpenML}}$



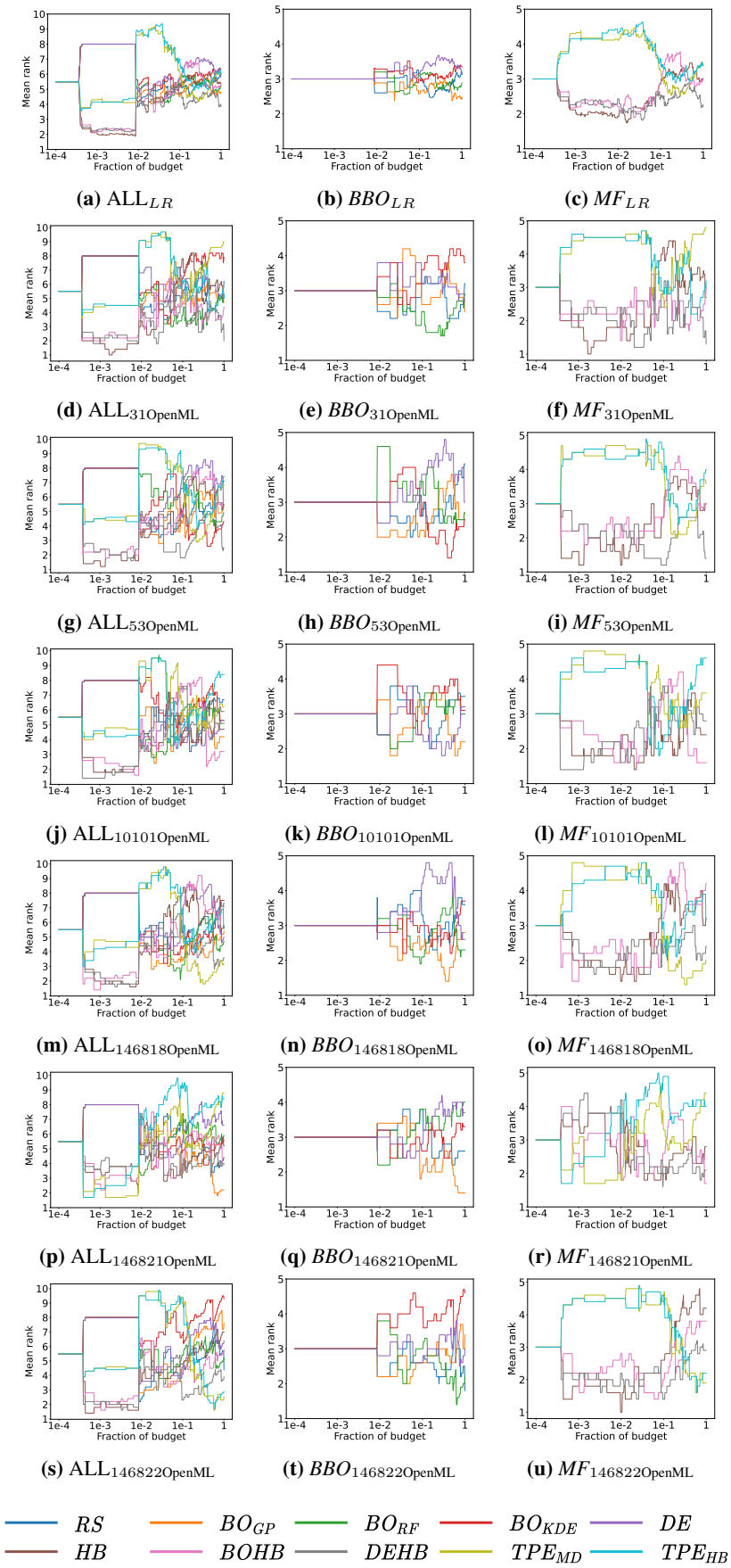
(p) $\text{ALL}_{146821\text{OpenML}}$ **(q)** $BBO_{146821\text{OpenML}}$ **(r)** $MF_{146821\text{OpenML}}$



(s) $\text{ALL}_{146822\text{OpenML}}$ **(t)** $BBO_{146822\text{OpenML}}$ **(u)** $MF_{146822\text{OpenML}}$

RS BO_{GP} BO_{RF} BO_{KDE} DE
HB $BOHB$ $DEHB$ TPE_{MD} TPE_{HB}

Figure 16: Mean rank over time on LR benchmark (FedAvg).



(a) ALL_{LR} **(b)** BBO_{LR} **(c)** MF_{LR}

(d) $\text{ALL}_{31\text{OpenML}}$ **(e)** $BBO_{31\text{OpenML}}$ **(f)** $MF_{31\text{OpenML}}$

(g) $\text{ALL}_{53\text{OpenML}}$ **(h)** $BBO_{53\text{OpenML}}$ **(i)** $MF_{53\text{OpenML}}$

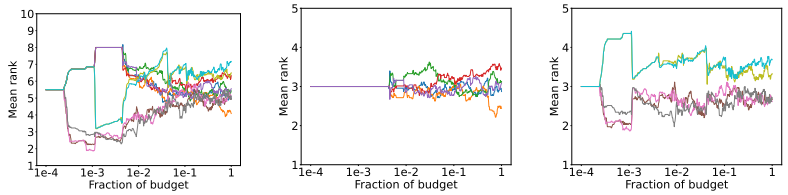
(j) $\text{ALL}_{10101\text{OpenML}}$ **(k)** $BBO_{10101\text{OpenML}}$ **(l)** $MF_{10101\text{OpenML}}$

(m) $\text{ALL}_{146818\text{OpenML}}$ **(n)** $BBO_{146818\text{OpenML}}$ **(o)** $MF_{146818\text{OpenML}}$

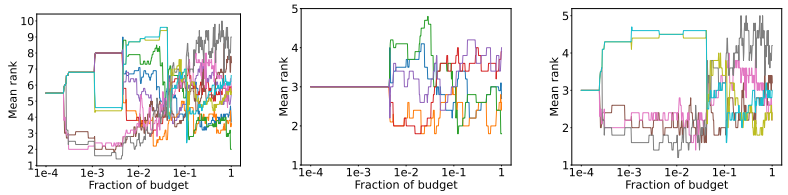
(p) $\text{ALL}_{146821\text{OpenML}}$ **(q)** $BBO_{146821\text{OpenML}}$ **(r)** $MF_{146821\text{OpenML}}$

(s) $\text{ALL}_{146822\text{OpenML}}$ **(t)** $BBO_{146822\text{OpenML}}$ **(u)** $MF_{146822\text{OpenML}}$

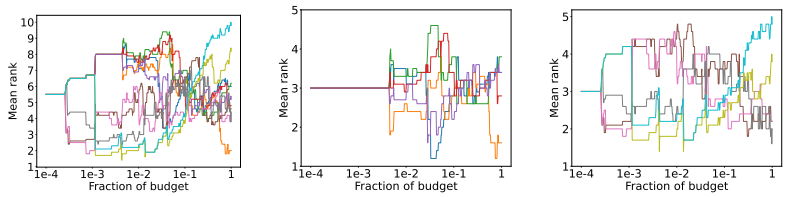
Figure 17: Mean rank over time on LR benchmark (FedOpt).



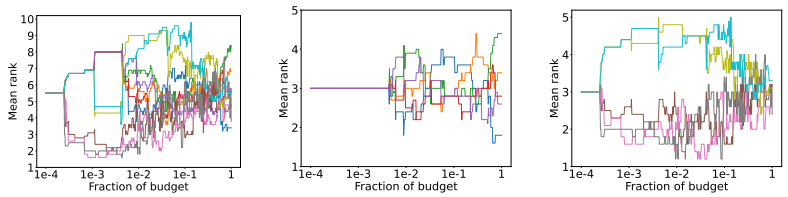
(a) ALL_{MLP} **(b)** BBO_{MLP} **(c)** MF_{MLP}



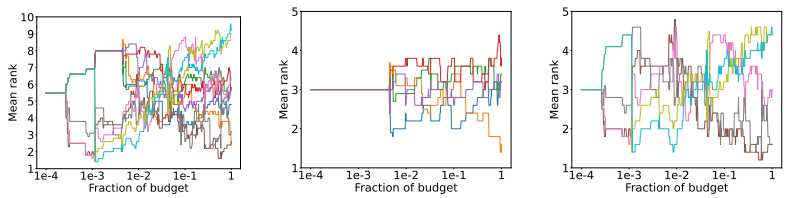
(d) $\text{ALL}_{\text{31OpenML}}$ **(e)** BBO_{31OpenML} **(f)** MF_{31OpenML}



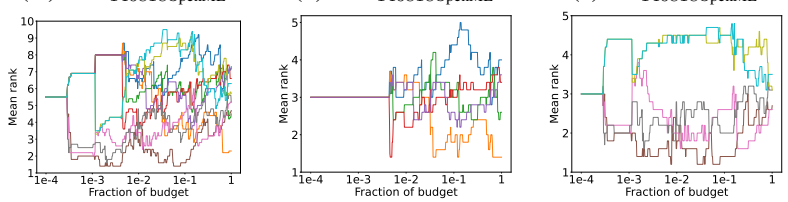
(g) $\text{ALL}_{\text{53OpenML}}$ **(h)** BBO_{53OpenML} **(i)** MF_{53OpenML}



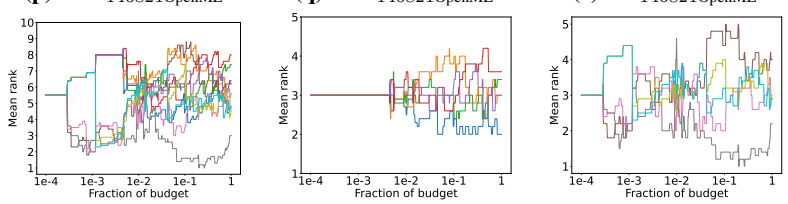
(j) $\text{ALL}_{\text{10101OpenML}}$ **(k)** $BBO_{\text{10101OpenML}}$ **(l)** $MF_{\text{10101OpenML}}$



(m) $\text{ALL}_{\text{146818OpenML}}$ **(n)** $BBO_{\text{146818OpenML}}$ **(o)** $MF_{\text{146818OpenML}}$



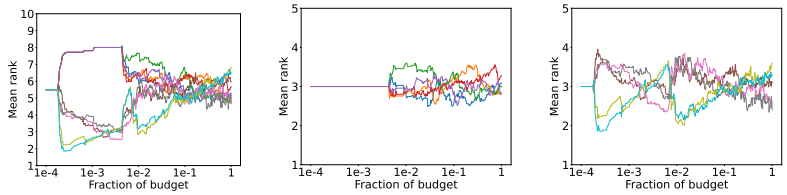
(p) $\text{ALL}_{\text{146821OpenML}}$ **(q)** $BBO_{\text{146821OpenML}}$ **(r)** $MF_{\text{146821OpenML}}$



(s) $\text{ALL}_{\text{146822OpenML}}$ **(t)** $BBO_{\text{146822OpenML}}$ **(u)** $MF_{\text{146822OpenML}}$

RS BO_{GP} BO_{RF} BO_{KDE} DE
HB $BOHB$ $DEHB$ TPE_{MD} TPE_{HB}

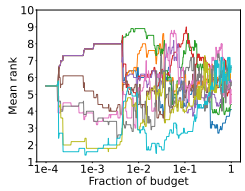
Figure 18: Mean rank over time on MLP benchmark (FedAvg).



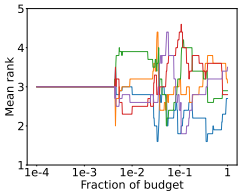
(a) ALL_{MLP}

(b) BBO_{MLP}

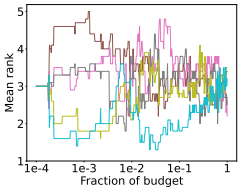
(c) MF_{MLP}



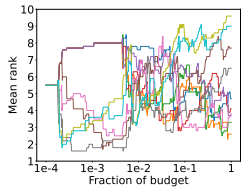
(d) $\text{ALL}_{31\text{OpenML}}$



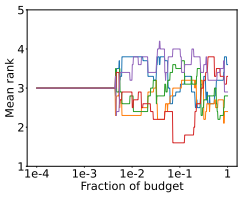
(e) $BBO_{31\text{OpenML}}$



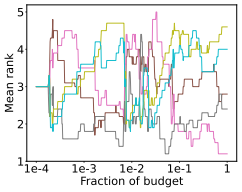
(f) $MF_{31\text{OpenML}}$



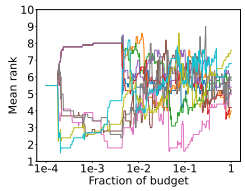
(g) $\text{ALL}_{53\text{OpenML}}$



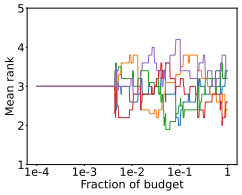
(h) $BBO_{53\text{OpenML}}$



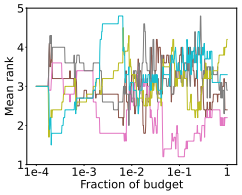
(i) $MF_{53\text{OpenML}}$



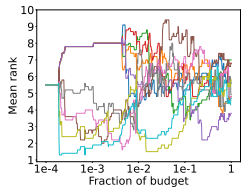
(j) $\text{ALL}_{10101\text{OpenML}}$



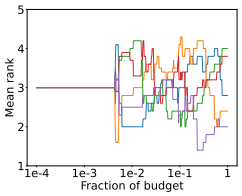
(k) $BBO_{10101\text{OpenML}}$



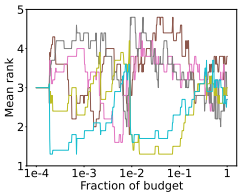
(l) $MF_{10101\text{OpenML}}$



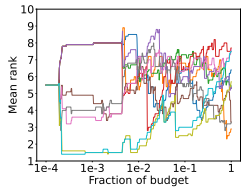
(m) $\text{ALL}_{146818\text{OpenML}}$



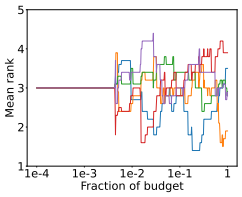
(n) $BBO_{146818\text{OpenML}}$



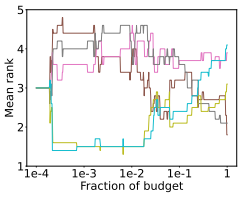
(o) $MF_{146818\text{OpenML}}$



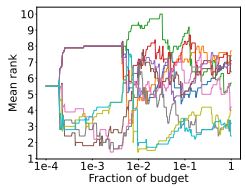
(p) $\text{ALL}_{146821\text{OpenML}}$



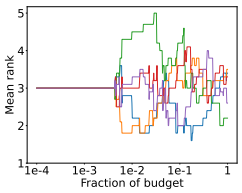
(q) $BBO_{146821\text{OpenML}}$



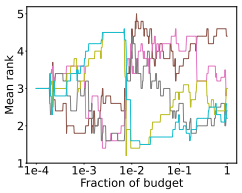
(r) $MF_{146821\text{OpenML}}$



(s) $\text{ALL}_{146822\text{OpenML}}$



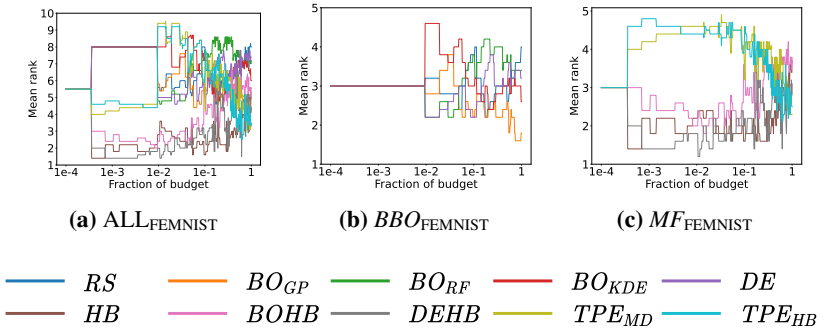
(t) $BBO_{146822\text{OpenML}}$



(u) $MF_{146822\text{OpenML}}$

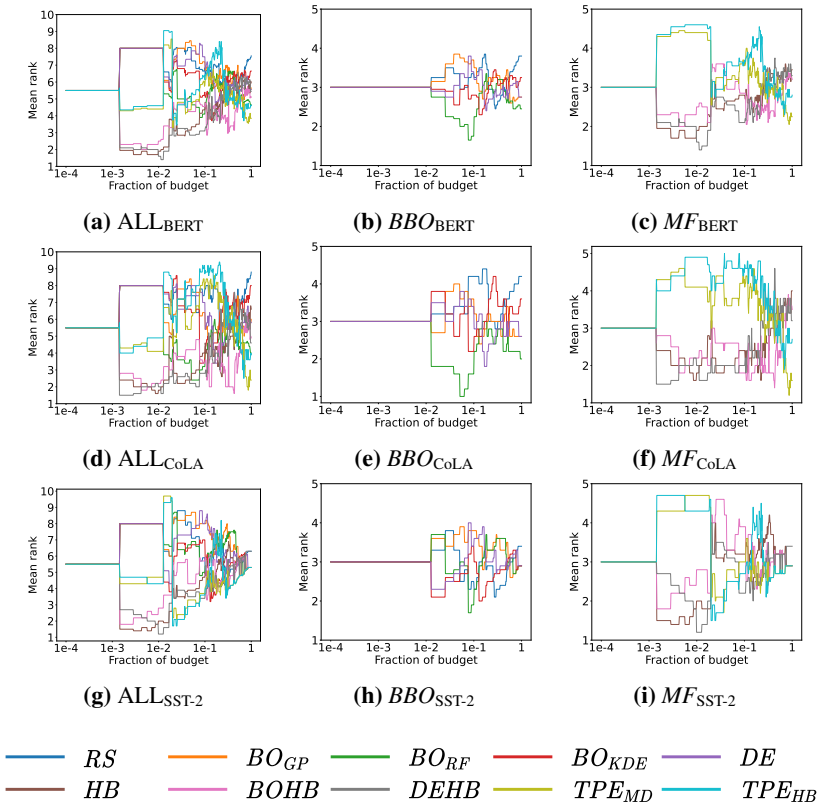
RS BO_{GP} BO_{RF} BO_{KDE} DE
HB $BOHB$ $DEHB$ TPE_{MD} TPE_{HB}

Figure 19: Mean rank over time on MLP benchmark (FedOpt).



(a) $\text{ALL}_{\text{FEMNIST}}$ **(b)** BBO_{FEMNIST} **(c)** MF_{FEMNIST}

Figure 20: Mean rank over time on CNN benchmark under surrogate mode (FedAvg).



(a) ALL_{BERT} **(b)** BBO_{BERT} **(c)** MF_{BERT}

(d) ALL_{CoLA} **(e)** BBO_{CoLA} **(f)** MF_{CoLA}

(g) $\text{ALL}_{\text{SST-2}}$ **(h)** $BBO_{\text{SST-2}}$ **(i)** $MF_{\text{SST-2}}$

Figure 21: Mean rank over time on BERT benchmark under surrogate mode (FedAvg).