HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO

Katharina Eggensperger[1*], **Philipp Müller**[1], **Neeratyoy Mallik**[1], **Matthias Feurer**[1], **René Sass**[2], **Aaron Klein**[3†], **Noor Awad**[1], **Marius Lindauer**[2], **Frank Hutter**[1,4]
[1] Albert-Ludwigs-Universität Freiburg [2] Leibniz Universität Hannover
[3] Amazon [4] Bosch Center for Artificial Intelligence

Abstract

To achieve peak predictive performance, hyperparameter optimization (HPO) is a crucial component of machine learning and its applications. Over the last years, the number of efficient algorithms and tools for HPO grew substantially. At the same time, the community is still lacking realistic, diverse, computationally cheap, and standardized benchmarks. This is especially the case for multi-fidelity HPO methods. To close this gap, we propose *HPOBench*, which includes 7 existing and 5 new benchmark families, with a total of more than 100 multi-fidelity benchmark problems. *HPOBench* allows to run this extendable set of multi-fidelity HPO benchmarks in a reproducible way by isolating and packaging the individual benchmarks in containers. It also provides surrogate and tabular benchmarks for computationally affordable yet statistically sound evaluations. To demonstrate *HPOBench*'s broad compatibility with various optimization tools, as well as its usefulness, we conduct an exemplary large-scale study evaluating 13 optimizers from 6 optimization tools. We provide *HPOBench* here: `https://github.com/automl/HPOBench`.

1 Introduction

The plethora of design choices in modern machine learning (ML) makes research on practical and effective methods for hyperparameter optimization (HPO) ever more important. In particular, ever-growing models and datasets create a demand for new HPO methods that are more efficient and powerful than existing black-box optimization (BBO) methods. Especially if it is only feasible to evaluate very few models fully, multi-fidelity optimization methods have been shown to yield impressive results by trading off cheap-to-evaluate proxies and expensive evaluations on the real target [1–5]. They showed tremendous speedups, such as accelerating the search process in low-dimensional ML hyperparameter spaces by a factor of 10 to 1000 [2, 5]. However, the development of such methods often happens in isolation, which potentially prevents HPO research from reaching its full potential. Prior publications on new HPO methods (i) often relied on artificial test functions and low-dimensional toy problems, (ii) sometimes introduced a new set of problems, (iii) set up on different computing environments, having different requirements and interfaces, and (iv) often did not open-source their code base. All of these make it difficult to compare and develop methods, necessitating an evolving set of relevant and up-to-date benchmark problems which drives continued and quantifiable progress in the community.

While there are efforts to simplify benchmarking HPO and global optimization algorithms [6–12], we are not aware of efforts to collect a diverse set of benchmarks in a single library, with a unified

[*] {eggenspk,mallik,fh}@cs.uni-freiburg.de
[†] work done prior to joining Amazon

Submitted to the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks. Do not distribute.

interface and countering potentially conflicting dependencies that may arise over time. The latter is particularly important because the rapid evolution of the Python-ML ecosystem can render a benchmark no longer usable for the community after a major release was published. This creates a significant hurdle for contribution from the community to grow a benchmark library. To solve this issue, we propose *HPOBench*, a benchmark suite for HPO problems, with a special focus on multi-fidelity problems, licensed under a permissive OSS license (*Apache 2.0*) and available at `https://github.com/automl/HPOBench`. *HPOBench* provides a common interface and an infrastructure to isolate benchmarks in their own containers and implements 12 popular optimization families, each with multiple problems and preserved with its dependencies in a container for long-term use. To enable efficient comparisons, most of these benchmarks are table- or surrogate-based, enabling resource efficient large-scale experiments, which we demonstrate in this work. Our contributions are:

1. The first available collection of multi-fidelity HPO problems. It contains 12 benchmark families with 100+ multi-fidelity HPO problems under a unified interface, comprising traditional HPO and neural architecture search (NAS). These benchmarks also define the largest collection of black-box HPO problems to date.
2. The first collection of *containerized* benchmarks to ensure the longevity, maintainability and extensibility of benchmarks.
3. The first set of HPO benchmarks that are available as both, the *raw* benchmark and the *tabular* version.
4. The first HPO benchmark that also supports multi-objective optimization and transfer-HPO across datasets (and arbitrary combinations of these with multiple fidelities).
5. We demonstrate how *HPOBench* can be used in an exemplary large-scale study with 13 optimizers from 6 optimization tools, assessing whether advanced methods outperform random search and how effective multi-fidelity HPO is.

This paper is structured as follows. We first discuss background on HPO and multi-fidelity optimization (Section 2). Then, we discuss related work on benchmarking (Section 3). Next, we describe the challenges for an HPO benchmark and how *HPOBench* alleviates them (Section 4). Then, we conduct a large-scale comparison of existing, popular HPO methods to demonstrate the usefulness of *HPOBench* (in Section 5). We conclude the paper by highlighting further advantages and potential future work (Section 6).

2 Background on Hyperparameter Optimization

With *HPOBench* we aim to provide benchmarks to evaluate HPO methods. In the following, we briefly formalize BBO for HPO and survey multi-fidelity optimization (see Feurer and Hutter [13] for a detailed overview), both with a focus on the methods used in our experiments.

2.1 Black-box Hyperparameter Optimization

Black-box optimization (BBO) aims to find a solution $\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} f(\boldsymbol{\lambda})$ where f is a black-box function, for which typically no gradients are available, we cannot make any statements about its smoothness, convexity and noise level. In summary, the only mode of interaction with black-box functions is querying them at given inputs $\boldsymbol{\lambda}$ and measuring the quantity of interest $f(\boldsymbol{\lambda})$. In the context of HPO, $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ is a hyperparameter configuration where the domain Λ_i of a hyperparameter is often bounded and continuous, but can also be integer, ordinal or categorical. There are also so-called conditional hyperparameters [14, 15] defining hierarchical search spaces; however, the first version of *HPOBench* focuses on flat configuration spaces first as all optimizers support this.

There are 3 broad families of BBO methods: (i) purely explorative approaches such as Random Search (*RS*) and grid search are simple but sample-inefficient (ii) model-free Evolutionary Algorithms (EAs) based on mutation, crossover and selection operators applied to a population of configurations require comparably large resources to evaluate the entire population but can perform very well given enough resources; (iii) iterative model-based methods, such as Bayesian Optimization [16], which are guided by a predictive model trained on prior function evaluations are known as the most sample-efficient methods. We include representative algorithms from each of these 3 families in our exemplary experiments in Section 5.

2.2 Multi-fidelity Hyperparameter Optimization

To efficiently optimize today's ever-growing ML models, multi-fidelity approaches relax the black-box assumption by allowing cheaper queries at lower fidelities b as well ($\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} f(\boldsymbol{\lambda}, b)$). Examples for these approximations include dataset subsets [2, 17, 18], feature subsets [19] or lower number of epochs [19–21]. Multi-fidelity methods have been shown to lead to speedups of up to $1000\times$ over black-box methods [2, 5]. *HPOBench* will allow the community to compare different multi-fidelity methods and in the following we give an overview of representative methods.

A popular multi-fidelity HPO approach that discretizes the fidelity space is Hyperband (*HB* [19]), a very simple method with strong empirical performance. It randomly samples new configurations and allocates more resources to promising configurations by repeatedly calling successive halving (*SH* [4]) as a sub-algorithm. The simplicity and effectiveness of *HB* have been leveraged with other popular black-box optimizers for improved performance: *BOHB* [22] combines *HB* with Bayesian Optimization (*BO*) and DEHB [5] combines it with the evolutionary approach of Differential Evolution (*DE* [23, 24]). The non-*HB*-based multi-fidelity case has also been researched extensively [2, 3, 18, 20, 21, 25–28]. Not being limited to predefined fidelity values makes these methods very powerful, but they rely on strong models to avoid poor choices of fidelities, often making HB-based fidelity selection more robust. To study the efficacy of multi-fidelity optimization, in our exemplary experiments in Section 5, we primarily compared black-box optimizers against their multi-fidelity versions (i.e., *RS* vs. *HB*, *BO* vs. *BOHB*, and *DE* vs. *DEHB*). These experiments show large speedups of multi-fidelity optimizers in the regime of small compute budgets, whereas for large compute budgets multi-fidelity optimization is less useful.

Besides multi-fidelity optimization, a very active field of study to speed up HPO is to use transfer-learning across datasets [29–33]; we note that transfer HPO methods can also be evaluated with *HPOBench* by learning across the datasets within each of its families.

3 Related Work

Proper benchmarking is hard. It is important to be aware of technical and methodological pitfalls, e.g. comparing implementations instead of algorithms [34, 35], comparing tuned algorithms versus untuned baselines [36, 37], to not fall for an illusion of progress [38, 39] and to know which sources of variance exist and control for them [40]. Also, there is a rich literature on how to empirically evaluate and compare methods in various domains, e.g. evolutionary optimization [41], planning [42], satisfiability and constraint satisfaction [43], algorithm configuration [44], NAS [45], and also for benchmarking optimization algorithms [46]. Our goal is not to provide further recommendations on how and why to benchmark, but to provide concrete benchmarks to simplify development and to improve the reproducibility and comparability of HPO and in particular multi-fidelity methods.

Furthermore, there have been a lot of efforts to provide optimization benchmarks for the community. Having a common set of benchmark problems in a unified format fosters and guides research. Prominent examples in the area of HPO are ACLib [47] for algorithm configuration, COCO [9] for continuous optimization, Bayesmark [8] for Bayesian optimization, Olympus [12] for optimization of experiment planning tasks, and HPO-B [48] for transfer-HPO methods (for more, see Appendix B). However, no benchmark so far has multi-fidelity optimization problems, supports preserving a diverse set of benchmarks for the longer term (containers), supports multiple objectives, and provides cheap-to-evaluate surrogate/tabular benchmarks; we hope to close this gap with *HPOBench*.

Besides benchmarks, competitions are another form of focusing research effort by providing a common goal and incentive. Famous examples are the *AutoML* challenges [49], the *AutoDL* challenge [50], the GECCO BBOB workshop series based on COCO [9] and the NeurIPS 2020 BBO challenge [51] (for more, see Appendix C). In contrast to these, we do not focus on defining concrete experimentation protocols, but rather on providing a flexible benchmarking environment to study, develop and compare optimization methods.

4 HPOBench: A Benchmark Suite for Multi-Fidelity Hyperparameter Optimization benchmarks

In this section, we present *HPOBench*, a collection of HPO benchmarks defined as follows:

Definition 1 (HPO Benchmark) *An HPO benchmark consists of a function $f : \boldsymbol{\lambda} \rightarrow \mathcal{R}$ to be minimized and a (bounded) hyperparameter space $\boldsymbol{\Lambda}$ with hyperparameters $[\Lambda_1, \ldots, \Lambda_d]$ of type continuous, integer, categorical or ordinal. In the case of multi-fidelity benchmarks, f can be queried at lower fidelities, $f : \boldsymbol{\lambda} \times \boldsymbol{b} \rightarrow \mathcal{R}$, and the fidelity space $\boldsymbol{B}$ describes which low-fidelities $[B_1, \ldots, B_e]$ of type continuous, integer or ordinal are available.*

Specifically, each benchmark consists of the implementation of that function, which returns at least one loss. Since this function typically evaluates an ML algorithm, the benchmark defines all relevant settings, dependencies and inputs, such as datasets, splits and how to compute the loss.

In the remainder of this section, we first discuss the desiderata of a benchmark that aids HPO research and then highlight the features of *HPOBench* by detailing how its design fulfills these desiderata.

4.1 Desiderata of an HPO Benchmark

One of the challenges posed to standardized HPO research lies in the varied choices of the underlying ML components – datasets and their splits, preprocessing, hyperparameter ranges, underlying software versions, and hardware used. Moreover, the practices applied in HPO research itself can vary along the lines of optimization budget, number of repetitions, metrics measured and reported. This leads to inconsistencies and difficulties in comparison of different HPO methods across publications and over time, affecting the reproducibility of experiments that hinders continued progress in HPO research.

In order to alleviate such issues and encourage participation by the research community, benchmarks need to standardize these practices to allow the community to be an active stakeholder in developing and re-using benchmarks. *HPOBench* is designed to both allow easy, flexible use with a minimal API that is identical for all benchmarks (see Figure 2); and have a low barrier for contributing new benchmark problems. We, therefore, identify 3 features of a benchmark that allow its wide-scale use and long-term applicability: (i) *efficiency* by providing tabular and surrogate benchmarks for quick, efficient experiments, along with the original benchmarks; (ii) *reproducibility* of results by containerizing benchmarks; and (iii) *flexibility* by covering different optimization landscapes and possible use cases, e.g. multi-objective, transfer-HPO, and even multi-fidelity optimization with multiple fidelity variables. To our knowledge, no other existing benchmarks offer these possibilities. *HPOBench* provides a framework to enable standardized, principled research and experimentation. We list all benchmarks hat are included in *HPOBench* in Table 1 and provide a detailed description of the respective configuration spaces in Appendix D.

4.2 Efficiency

HPO benchmarks that follow Definition 1 exhibit the drawback that they evaluate a costly function, rendering the empirical comparison of optimization algorithms expensive and ruling out such benchmarks for interactive development of new methods. To overcome this issue, beside such raw benchmarks, we also provide two well-established benchmark classes which alleviate this issue:

Definition 2 (Tabular Benchmark) *A tabular benchmark returns values from a lookup table with recorded function values of a raw HPO benchmark instead of evaluating $f(\boldsymbol{\lambda})$. The (bounded) hyperparameter space is restricted to only contain these values and therefore bears a form of discretization. In the case of multi-fidelity benchmarks, each tabular benchmark has a fidelity space and the underlying table also contains the recorded function values on the low-fidelities.*

Tabular benchmarks are popular in the HPO community as they are easy to distribute and induce little overhead [52, 30, 53–55], however, they require to discretize the hyperparameter space. Surrogate benchmarks [56, 57] are an alternative since they provide the original hyperparameter space.

Definition 3 (Surrogate Benchmark) *A surrogate benchmark returns function values predicted by an ML model trained on a tabular benchmark or recorded function values of a raw HPO benchmark. It reuses the original hyperparameter space and can be extended to the multi-fidelity case as well.*

While surrogate benchmarks are similarly cheap to query, the surrogate's internal ML model adds extra complexity and the benchmark's quality crucially depends on the quality of this model and its training data. Because surrogate benchmarks yield a drop-in replacement for raw benchmarks, they enjoy widespread adoption in the HPO community [22, 58, 57, 59–62].

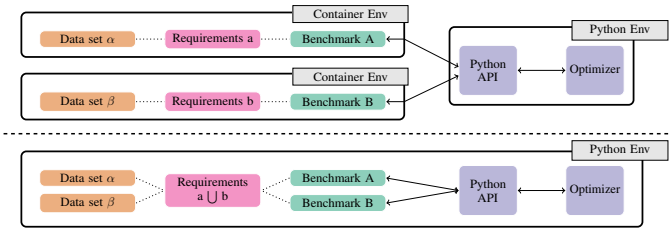

Figure 1: Overview of benchmark environments with (upper) and without (lower) using containers.

```python
from hpobench.container.benchmarks.\
    nas.nasbench_101 import NASCifar10ABenchmark

# if necessary downloads container/data
b = NASCifar10ABenchmark(rng=1)
# get hyperparameter space
space = b.get_configuration_space(seed=1)
# sample config at random
config = space.sample_configuration()

# eval at multiple low-fidelities
res = b.objective_function(
            configuration=config,
            fidelity={"budget": 12}, rng=1)
res = b.objective_function(
            configuration=config,
            fidelity={"budget": 108}, rng=1)
```

Figure 2: Code example initializing and evaluating a benchmark.

Furthermore, while *HPOBench* puts a strong focus on multi-fidelity benchmarks, it also facilitates evaluating black-box optimization algorithms. In fact, a multi-fidelity benchmark with k different fidelity levels can be used to define k separate (yet related) benchmarks for black-box optimization. As such, *HPOBench* defines more than 400 black-box HPO benchmarks.

4.3 Reproducibility

One of the challenges that come with many new benchmarks is their one-off development and their lack of maintenance. This means that any new update to the benchmark or its dependencies can easily lead to conflicts and inconsistencies with respect to software dependencies and possibly old published results (see Appendix D.1 for examples). While in practice the very same problem, also known as *dependency hell*, can also occur on the optimizer side, in this paper we focus on the benchmark side.

HPOBench circumvents such issues through the containerization of benchmarks using Singularity [63] containers.[3] Each benchmark and its dependencies are packaged as a separate container, which isolates benchmarks from each other and also from the host system. Figure 1 illustrates the advantages that containerization provides, especially when running multiple benchmarks in the same environment. Note that without containers, the environment needs to satisfy the union of all of its benchmarks' requirements (which may actually be mutually exclusive!), while with containers the dependencies for any given benchmark only need to be satisfied once: for the creation of the container. Importantly, the dependencies do not need to be satisfied again for using the benchmarks. Each benchmark is uploaded as a container to a GitLab container registry to provide the history of different versions of the benchmark. Hence, any benchmark created under the *HPOBench* paradigm remains usable without additional bookkeeping or installation overheads for long-term usage. Additionally, no effort is required for maintaining already containerized benchmarks, as long as the API does not change. Although not recommended, each benchmark can also be installed locally along with its specific dependencies without using the containers, since the interfaces of the local and the containerized version are identical and thus interchangeable. We provide a short code sample in Figure 2.

4.4 Flexibility

HPOBench is a flexible framework that can be used to validate existing HPO research, and develop and improve HPO algorithms, with a focus on multi-fidelity methods. It consists of two sets of benchmarks, which we describe in turn: 22 existing multi-fidelity benchmarks from 7 families that we collected from the multi-fidelity literature (Section 4.4.1); and 88 new benchmarks from 5 families we created to allow a much more flexible use of *HPOBench* (Section 4.4.2).

4.4.1 Existing Community Benchmarks

Firstly, to allow comparability with previous experiments, we collected 22 existing multi-fidelity benchmarks from 7 families from the multi-fidelity literature; *HPOBench* preserves these benchmarks by containerizing them and encapsulating them all under a common API (which was not the case before). This not only ensures important previous work to remain accessible publicly under the same umbrella, but it also bypasses dependency issues that have appeared in these benchmarks enabling long term usage (see Appendix D.1).

[3] We chose Singularity over the popular Docker (`https://www.docker.com/`) alternative as it (1) does not require super user access and (2) is available on the computer clusters we have access to.

Table 1: Overview of raw (✓), surrogate (✗) and tabular ((✓)) benchmarks. We report the number of benchmarks per family (*#benchs*), the number of continuous (*#cont*), integer (*#int*), categorical (*#cat*), ordinal (*#ord*) hyperparameters and how many are log-scaled. Furthermore, we report the fidelity, the optimization budget and the number of configurations for tabular and surrogate benchmarks.

Family	#benchs	#cont(log)	#int(log)	#cat	#ord	fidelity	type	opt. budget	#confs	Ref.
Cartpole	1	4(1)	3(3)	-	-	repetitions	✓	1d	-	[22]
BNN	2	3(1)	2(2)	-	-	samples	✓	1d	-	[22]
Net	6	5	1	-	-	time	✗	7d	-	[22]
NBHPO	4	-	-	3	6	epochs	(✓)	10^7sec	62 208	[64]
NB101	3	- - 21	- - 1	26 14 5	- - -	epochs	(✓)	10^7sec	423k	[54]
NB201	3	-	-	6	-	epochs	(✓)	10^7sec	15 625	[65]
NB1Shot1	3	- - -	- - -	9 9 11	- - -	epochs	(✓)	10^7sec	6 240 29 160 363 648	[66]
LogReg	20	2(2)	-	-	-	iter	✓, (✓)	100×	625	*new*
SVM	20	2(2)	-	-	-	data	✓, (✓)	average	441	*new*
RandomForest	20	1	3(2)	-	-	#trees	✓, (✓)	runtime on	10k	*new*
XGBoost	20	3(2)	1(1)	-	-	#trees	✓, (✓)	the highest	10k	*new*
MLP	8	2(2)	3(2)	-	-	epochs	✓, (✓)	fidelity	30k	*new*

Specifically, these benchmarks comprise raw benchmarks tuning a reinforcement learning agent (PPO on *Cartpole* [22]) and a Bayesian neural network (*BNN* [22]), a random forest-based surrogate benchmark tuning an MLP (*Net* [22]) and four popular NAS benchmark families (*NBHPO* [64], *NB101* [54], *NB201* [65], and *NB1Shot1* [66]). However, these existing community benchmarks also have certain limitations: they are only of limited use for transfer HPO (since there are only between 1 and 6 benchmarks per family), they only offer a single fidelity dimension, and they only evaluate a single metric. We therefore augmented them with 5 new families of benchmarks we describe next.

4.4.2 New Benchmarks

To substantially increase the range of possible applications of *HPOBench*, we defined 5 new benchmark families with up to 20 different datasets per family, comprising a total of 88 new multi-fidelity benchmarks. These new benchmarks also provide multiple metrics and multiple fidelity dimensions to go beyond the aforementioned limitations of the community benchmarks.

Our new benchmarks are based on the following popular ML algorithms: ***SVM***, ***LogReg***, ***XGBoost***, ***RandomForest***, and ***MLP***. All of them evaluate the respective ML algorithm as implemented in scikit-learn [67] and XGBoost [68] on 20 publicly available datasets (8 for the *MLP* due to its high computational cost) from the OpenML AutoML benchmark [69]. We give the OpenML [70] task IDs in Table 7 in Appendix D, which provide fixed train-test splits; for each such task, we used 33% of the training set as the validation split, determined through stratified sampling under a fixed seed. The entire objective function then consists of preprocessing, training the model on the remaining 66% of the fixed OpenML training split, prediction on the fixed validation split, evaluating 4 different metrics, and recording model fit and inference times. The fidelities are algorithm-specific if possible (number of trees, iterations, epochs) or dataset subsets otherwise (which is used for *SVM*). These benchmarks are available both as raw and tabular versions, have the same API and exist in independent, non-conflicting containers; for the tabular versions, we discretized each hyperparameter (and fidelity) and evaluated 5 different seeds for each configuration of the resulting grid.

Also, four of our new benchmark families (*LogReg*, *RandomForest*, *XGBoost*, *MLP*) allow up to two fidelity dimensions. This enables the development and benchmarking of methods for multi-fidelity optimization with multiple fidelity dimensions, a direction that we deem very promising yet understudied. Similarly, our tabular data collected over multiple datasets (up to 20) allows the effective use of these benchmarks for transfer-HPO, and the recording of multiple evaluation metrics also

allows these benchmarks to be used for multi-objective optimization. Moreover, each configuration is recorded on different fidelities with their associated costs, which further lends *HPOBench* great potential in future research in cost-based meta-learning or multi-fidelity multi-objective optimization.

To demonstrate the diversity of our new benchmarks, we show the *empirical cumulative distribution function* (ECDF) for each family in Figure 3. Each line corresponds to one dataset and shows how the objective values are distributed. From the varying amounts of well and badly performing normalized regrets we can conclude that the benchmarks yield different landscapes and thus are diverse in smoothness, which results in varying algorithm performance. Moreoever, the 5 new spaces vary in their dimensionality (up to 5 for *MLP*), in the hyperparameter data types and their range (see Appendix D).

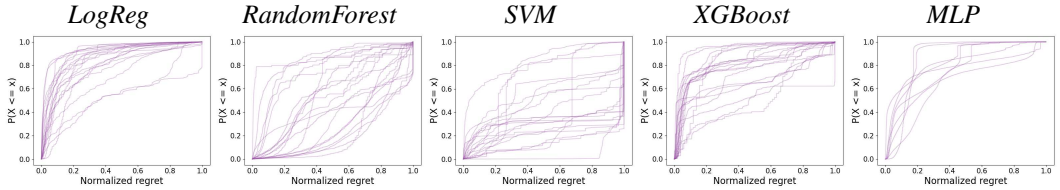

Figure 3: Empirical cumulative distribution. Each plot corresponds to one ML algorithm, and each line within a plot corresponds to one dataset. The lines show the ECDF of the normalized regret of all evaluated configurations of the respective ML algorithm on the respective dataset.

5 Experiments

Now, we turn to an exemplary use of our benchmarks in order to demonstrate some features of *HPOBench* and its utility for HPO research. We used our benchmark suite to run a large-scale empirical study comparing 13 optimization methods on our 12 benchmark families (we report detailed results in Appendix H). We first give details on the experimental setup and then study the following two exemplary research questions: **(RQ1)** *Do advanced methods improve over random baselines?* and **(RQ2)** *Do multi-fidelity methods improve over single-fidelity methods?*

5.1 Experimental Setup

For each benchmark and optimizer, we conducted 32 repetitions with a different seed each. For our new benchmarks, which have multiple metrics, we minimized $1-$accuracy. For each run, we allowed an optimization budget as described in Table 1 and accumulate time taken by the benchmark (recorded time for tabular benchmarks, predicted time for surrogate benchmarks and wallclock time for raw benchmarks; for our new benchmarks, we used the tabular versions to avoid unnecessary compute costs and CO2 exhaustion) and the optimizer (wallclock time). We kept track of all evaluations and computed trajectories, i.e., the best-seen value at each time step, as follows: If for an evaluation we cannot find another evaluation conducted on the same or a higher fidelity, we treat it as the best-seen value; if it is on the highest fidelity evaluated so far, we treat it as the best seen value if it has a lower loss than the best-seen so far on that fidelity; otherwise, we do not consider this evaluation for the trajectory. This decision reflects the multi-fidelity setting, where a higher budget results in a better estimate of the actual value of interest but can cause jumps in the optimization trajectory, (e.g., when a configuration is the first to be evaluated on a higher budget but is worse than the best configuration on a lower budget). To aggregate and report results, we use either the *final performance* (per benchmark, see Appendix H), *performance-over-time* (per benchmark, see Appendix H) or *rank-over-time* (across multiple benchmarks). For tabular and surrogate benchmarks we report optimization regret (the difference between the best-found value and the best-known value) and for the other benchmarks, we report the actual optimized objective value.[4]

We give details on the hardware and required compute resources in Appendix E and F and release code for the experiments here: `https://github.com/automl/HPOBenchExperimentUtils`.

[4]Since we study optimizers, we report optimization performance (in the case of ML the validation performance, which is the objective value seen by the optimizer. We note that *HPOBench* in principle allows to compute test performance (the loss computed on a separate test set on the highest fidelity).

5.2 Considered Optimizers

We evaluated a wide set of optimizers including baselines for black-box and multi-fidelity optimization which we briefly describe (for more details see Appendix G). Our selection of optimization algorithm does *not* aim at finding the best optimization algorithm, but to study a broad range of different implementations and tools. While there are more recent and efficient BO and black-box methods available [71, 72], we focus here on multi-fidelity methods and only include basic black-box methods for which there are easy-to-use tools available.

As black-box optimizers, which only access the highest fidelity, we considered random search (*RS*), differential evolution (*DE* [23, 24]) and *BO* with different models: a Gaussian Process model (BO_{GP}), a random forest (BO_{RF} [73]), a kernel density estimator (KDE) (BO_{KDE} [22]) and a tree Parzen estimator (TPE) (Ray_{hyp} [14]). For multi-fidelity optimization, we used the multi-fidelity extensions of the methods mentioned above: Hyperband (*HB* [19]) and its combination with KDE-based *BO* (*BOHB* [22]), with RF-based *BO* (*SMAC-HB*) and with *DE* (*DEHB* [5]). Additionally, we use Dragonfly [74] using a GP with multi-fidelity optimization. Furthermore, we ran combinations of optimization and multi-fidelity algorithms implemented in Ray [75] and Optuna [76]. [5]

5.3 RQ1: Do advanced methods improve over random search?

To demonstrate the validity of our benchmarks, we independently replicate the findings of the 1st NeurIPS Blackbox Optimization challenge [51]: "*decisively showing that BO and similar methods are superior choices over RS and grid search for tuning hyperparameters of ML models*". While this question has already been studied before [14, 77, 36, 15, 33, 72, 78], we will also study it w.r.t. multi-fidelity optimization and using the popular *HB* baseline. We leave out grid search as *RS* has been shown to be superior [77] and as there is no multi-fidelity version of grid search.

We report ranks-over-time in Figure 4, comparing black-box (*DE*, BO_{GP}, BO_{RF}, Ray_{hyp}, BO_{KDE}; 1st column) and multi-fidelity (*BOHB*, *DEHB*, *SMAC-HB*, *DF*, Ray^{asha}_{hyp}; 2nd column) optimizers on *existing community* (top row) and *new* (bottom row) benchmarks. On both benchmark sets most black-box and multi-fidelity optimizers clearly outperform the respective baseline (*RS* (blue) and *HB* (light green)) on average. We also observe that *BO* improves over the evolutionary algorithm *DE* in the beginning but looses to it in the very long run on the *existing community* benchmarks [79, 60, 5]. This does not happen on the *new* benchmarks, as their time limits are set more aggressively and the methods developed for this setting (BO_{GP}, BO_{RF} and *SMAC-HB* [80]) achieve lower ranks. Considering per-benchmark results (Appendix H), we also observe that methods which appear clearly inferior in the ranking plots perform very well on individual benchmarks (e.g. *DF*[6] on *NB201* and Ray^{asha}_{hyp} on Net_{Adult}).

Besides qualitative measures, we also quantitatively measure whether the advanced methods outperform the respective baselines by counting the number of wins, ties and losses and using the sign test to verify significance [81] on the existing community benchmarks in Table 2 (the new benchmarks yield similar results; see Appendix H). We can observe that four out of five black-box methods are significantly better than *RS*. In the multi-fidelity case, only two out of five methods were significantly better than *HB*, one method was insignificantly better than *HB* and two methods were consistently worse than *HB*. Overall, we conclude that advanced methods consistently outperform random search.

5.4 RQ2: Do multi-fidelity methods improve over black-box methods?

Next, we study whether multi-fidelity optimization methods are able to consistently improve over black-box optimization methods given a fixed time budget. For this, we look again at ranking-over-time in Figure 4. We first compare black-box methods with their respective multi-fidelity extension, i.e., *DE* vs. *DEHB* and BO_{KDE} vs. *BOHB* in the two plots in the rightmost column. We can see that in the beginning *HB* and the multi-fidelity optimizers perform very similarly and consistently

[5] We include these frameworks to show compatibility of *HPOBench* with popular frameworks, but note that these expect to freeze and thaw evaluations. *HPOBench* implements a stateless objective function and, thus, runs that could be thawed and continued instead get accounted the full costs of rerunning them, which slows down optimization. We defer stateful benchmarks to future work.

[6] We would like to note that the bad rank of *DF* for some benchmarks is due to its overhead which prevented it from spending sufficient budget on function evaluations; see Section F for details.

Table 2: P-value of a sign test for the hypothesis that advanced methods outperform the baseline *RS* for black-box optimization and *HB* for multi-fidelity optimization. We underline p-values that are below $\alpha = 0.05$ and also boldface p-values that are below $\alpha = 0.05$ after multiple comparison correction (dividing α by the number of comparisons, i.e. 5; boldface/underlined implies that the advanced method is better). We also give the wins/ties/losses of *RS* and *HB* against the challengers.

	DE	*BO*$_{GP}$	*BO*$_{RF}$	*Ray*$_{hyp}$	*BO*$_{KDE}$
p-value against *RS*	**0.00006**	**0.00360**	**0.00001**	**0.00043**	0.41591
wins/ties/losses of *RS*	1/2/19	3/3/16	0/3/19	2/2/18	9/2/11
	BOHB	*DEHB*	*SMAC-HB*	*DF*	*Ray*$_{hyp}^{asha}$
p-value against *HB*	0.26173	**0.00001**	**0.00011**	0.99640	0.97376
wins/ties/losses of *HB*	8/2/12	0/3/19	0/5/17	16/1/5	15/0/7

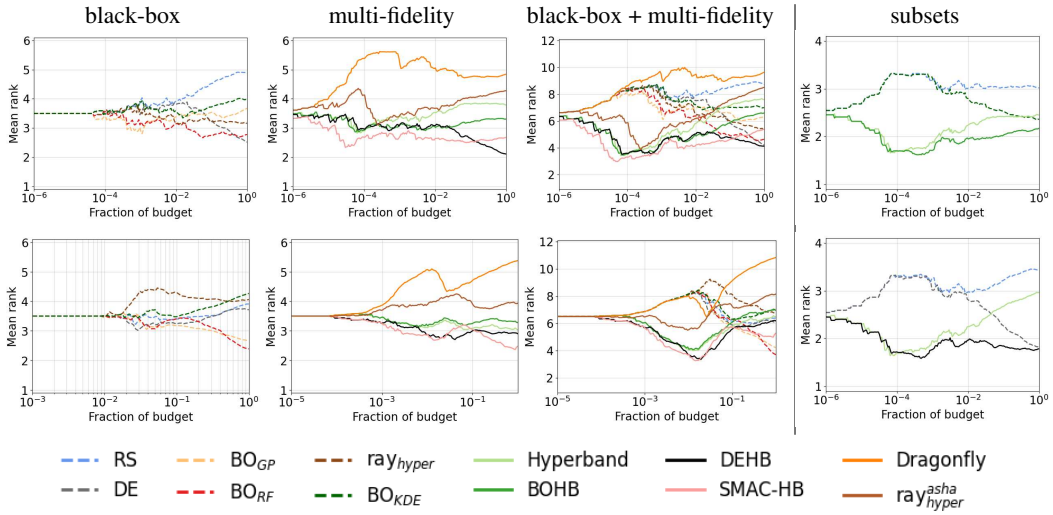

Figure 4: Mean *rank-over-time* across 32 repetitions of different sets of optimizers (lower is better). The left part shows rank across all *existing community* benchmarks (upper row) and *new* benchmarks (lower row). The right part report results on the *existing community* benchmarks only for subsets of optimizers.

outperform *RS* and the respective black-box version. After a while, the multi-fidelity versions improve over the *HB* baseline, and given enough time, the black-box versions catch up. Second, we compare all optimizers on the *existing community* (3rd column, top) and *new* (3rd column, bottom) benchmarks. Here, we can observe a similar pattern in that *HB* is a very competitive baseline in the beginning but is outperformed first by the advanced multi-fidelity methods and then also by the black-box methods. This is less pronounced on the *new* benchmarks, which we attribute to the tighter time limits.

Similarly to RQ1, we again counted the wins, ties and losses and used the sign test to verify significance [81] on the existing community benchmarks for 100%, 10% and 1% of the total budget in Table 3 (the new benchmarks yield similar results; see Appendix H). We can observe that only *HB* is able to outperform its black-box counterpart for all three budgets we check. For the other three multi-fidelity methods there is a significant improvement over the black-box methods for 1% of the budget. For 10% only *BOHB* is still significantly better, while *DEHB* is only insignificantly better and *SMAC-HB* ties. For the full budget we can no longer state that any of the multi-fidelity methods is statistically better than their counterpart, but judging by the wins and losses the multi-fidelity methods are still competitive.

Overall, multi-fidelity optimizers outperform black-box optimizers for relatively small compute budgets. Given enough budget, black-box optimizers become competitive with their multi-fidelity versions; in particular, *DE* and *BO*$_{RF}$ performed very well in the end. However, we need to take into account that for the *existing community* benchmarks the potential catch-up (if at all) only happens after

a very substantial amount of (simulated) wallclock time (e.g., 10 Mio. seconds). Hence, multi-fidelity methods are crucial to efficiently tackle real, expensive optimization problems.

Table 3: P-values of a sign test for the hypothesis that multi-fidelity outperform their black-box counterparts. We boldface p-values that are below $\alpha = 0.05$ (boldface implies that the multi-fidelity method is better).

Budget		*RS* vs *HB*	*DE* vs *DEHB*	BO_{KDE} vs *BOHB*	BO_{RF} vs *SMAC-HB*
100%	p-values	**0.00217**	0.58409	0.14314	0.85686
	w/t/l	2/4/16	7/8/7	7/2/13	10/6/6
10%	p-values	**0.00360**	0.06690	**0.01330**	0.58409
	w/t/l	3/3/16	3/8/11	4/3/15	8/6/8
1%	p-values	**0.00074**	**0.00845**	**0.02624**	**0.03918**
	w/t/l	2/3/17	4/2/16	4/4/14	5/3/14

To conclude, in general when low-fidelities are available and they are representative of the true objective function, multi-fidelity methods are clearly beneficial. In practice, we found that *DEHB* and *SMAC-HB* are reliable multi-fidelity optimizers that work well across the whole collection of benchmarks, while other multi-fidelity optimizers are not able to improve over *HB* consistently. By exploring a very broad range of benchmarks, we also found an existence proof that black-box methods can outperform multi-fidelity methods for very high budgets and that even advanced methods can be outperformed by *RS* in individual benchmarks. We pose it as a challenge to the field to develop methods that do not exhibit poor performance in *any* of the many benchmarks in *HPOBench*.

6 Discussion and Future Work

We proposed *HPOBench*, a library for multi-fidelity HPO benchmarks. It serves two purposes: (a) to provide benchmarks with a unified API, and (b) to make them easy to install and use by containerizing them and thus enable rapid prototyping and the development of new multi-fidelity methods that are crucial for ML research and applications. Finally, our library is open-source and we welcome contributions of new benchmarks to keep the library up-to-date and evolve it.

On the technical side, we so far focused on developing a benchmark library, but we see a large potential in connecting our library with other benchmarking frameworks (e.g. COCO [9] and Bayesmark [8]), optimization frameworks (e.g. Nevergrad [10] and Sherpa [82]) and extending it with further benchmarks [11, 60, 83, 62, 12] to increase diversity and to simplify evaluation and comparison of optimizers. For this it would be interesting to also containerize the optimizers since they can suffer from the same issues as benchmarks. Furthermore, so far *HPOBench* only contains stateless benchmarks starting a single container. We would like to extend the library to also support optimizers requiring stateful benchmarks (to freeze and thaw evaluations) or running in parallel.

Our set of benchmarks already covers raw, tabular, and surrogate benchmarks, but it would be useful to have all three versions available for all benchmarks, and to automatically generate tabular and surrogate-based benchmarks from raw benchmarks. Also, our new benchmarks can be used to evaluate multi-objective (multiple metrics) and meta-learning (across datasets) methods or even meta-learned multi-fidelity multi-objective methods. We hope for the community to play a large role in defining the protocols for the different special cases; e.g., budgets need to be set differently for black-box multi-objective optimization and single-objective hyperparameter transfer learning. Additionally, it would be interesting to study hierarchical search spaces to cover work in *AutoML*. Furthermore, there is a large potential in automatically creating multi-fidelity benchmarks from any ML algorithm by using data subsets as a low-fidelity.

We also conducted a large-scale study evaluating 13 algorithm implementations to demonstrate compatibility with a wide range of optimization tools, and we thus believe that our library is well suited for future research on multi-fidelity optimization. We showed that advanced HPO methods are preferable over *RS* and *HB* baselines, and that multi-fidelity extensions of popular optimizers improve over their black-box version. Lastly, to reduce computational effort, we would like to study whether we can learn which benchmarks are hard and whether it suffices by executing only a representative subset of the benchmarks [84].

References

[1] A. Forrester, A. Sobester, and A. Keane. Multi-fidelity optimization via surrogate modelling. *Proceedings of The Royal Society A Mathematical Physical and Engineering Sciences*, 463: 3251–3269, 2007.

[2] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In A. Singh and J. Zhu, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54. Proceedings of Machine Learning Research, 2017.

[3] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In D. Precup and Y. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 1799–1808. Proceedings of Machine Learning Research, 2017.

[4] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In A. Gretton and C. Robert, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51. Proceedings of Machine Learning Research, 2016.

[5] N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2147–2153. International Joint Conferences on Artificial Intelligence Organization, 2021.

[6] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13)*, 2013.

[7] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv:1810.05281 [cs.NE]*, 2018.

[8] R. Turner and D. Eriksson. Bayesmark: Benchmark framework to easily compare bayesian optimization methods on real machine learning tasks. `github.com/uber/bayesmark`, 2019.

[9] N. Hansen, A. Auger, R. Ros, O. Mersman, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 2020.

[10] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

[11] L. Bliek, A. Guijt, R. Karlsson, S. Verwer, and M. de Weerdt. EXPObench: Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *arXiv:2106.04618 [cs.LG]*, 2021.

[12] F. Häse, M. Aldeghi, R. Hickman, L. Roch, M. Christensen, E. Liles, J. Hein, and A. Aspuru-Guzik. Olympus: a benchmarking framework for noisy optimization and experiment planning. *Machine Learning: Science and Technology*, 2(3), 2021.

[13] M. Feurer and F. Hutter. Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, volume 5 of *The Springer Series on Challenges in Machine Learning*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book.

[14] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*, pages 2546–2554. Curran Associates, 2011.

[15] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855. ACM Press, 2013.

[16] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[17] J. Petrak. Fast subsampling performance estimates for classification algorithm selection. Technical Report TR-2000-07, Austrian Research Institute for Artificial Intelligence, 2000.

[18] K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13)*, pages 2004–2012. Curran Associates, 2013.

[19] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[20] K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014.

[21] T. Domhan, J. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3460–3468, 2015.

[22] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1437–1446. Proceedings of Machine Learning Research, 2018.

[23] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[24] 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.

[25] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In S. Matwin, S. Yu, and F. Farooq, editors, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1487–1495. ACM Press, 2017.

[26] J. Wu, S. Toscano-Palmerin, P. Frazier, and A. Wilson. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In R. Adams and V. Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference (UAI'20)*, volume 115, pages 788–798. PMLR, 2020.

[27] S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, and M. Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In H. Daume III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 98, pages 9334–9345. Proceedings of Machine Learning Research, 2020.

[28] J. Song, Y. Chen, and Y. Yue. A general framework for multi-fidelity Bayesian optimization with Gaussian processes. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 3158–3167. Proceedings of Machine Learning Research, 2019.

[29] J. Vanschoren. Meta-learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, volume 5 of *The Springer Series on Challenges in Machine Learning*, pages 35–61. Springer, 2019. Available for free at http://automl.org/book.

[30] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 199–207. Omnipress, 2013.

[31] D. Yogatama and G. Mann. Efficient transfer learning method for automatic hyperparameter tuning. In S. Kaski and J. Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 1077–1085, 2014.

[32] M. Feurer, J. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1128–1135. AAAI Press, 2015.

[33] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

[34] H.-P. Kriegel, E. Schubert, and A. Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowl. Inf. Syst.*, 52(2):341–378, 2017.

[35] S. Narang, H. Chung, Y. Tay, W. Fedus, T. Fevry, M. Matena, K. Malkan, N. Fiedel, N. Shazeer, Z. Lan, Y. Zhou, W. Li, N. Ding, J. Marcus, A. Roberts, and C. Raffel. Do transformer modifications transfer across implementations and applications? *arxiv:2102.11972 [cs.LG]*, 2021.

[36] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 115–123. Omnipress, 2013.

[37] G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*, 2018. Published online: `iclr.cc`.

[38] D. Hand. Classifier Technology and the Illusion of Progress. *Statistical Science*, 21(1):1 – 14, 2006.

[39] M. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *RecSys '19: Proceedings of the 13th ACM Conference on Recommender Systems*, page 101–109, New York, NY, USA, 2019. Association for Computing Machinery.

[40] X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, N. Sepah, E. Raff, K. Madan, V. Voleti, S. Kahou, V. Michalski, D. Serdyuk, T. Arbel, C. Pal, G. Varoquaux, and P. Vincent. Accounting for variance in machine learning benchmarks. *arXiv:2103.03098 [cs.LG]*, 2021.

[41] T. Weise, R. Chiong, and K. Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology*, 27:907–936, 2012.

[42] A. Howe and E. Dahlman. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 17:1–33, 2002.

[43] I. Gent, S. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. Smith, and T. Walsh. How not to do it. Technical Report 97.92, University of Leeds, 1997.

[44] K. Eggensperger, M. Lindauer, and F. Hutter. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, pages 861–893, 2019.

[45] M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21:1–18, 2020.

[46] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. López-Ibáñez, K. Malan, J. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. Benchmarking in optimization: Best practice and open issues. *arXiv:2007.03488v2 [cs.NE]*, 2020.

[47] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Stützle. AClib: a benchmark library for algorithm configuration. In P. Pardalos and M. Resende, editors, *Proceedings of the Eighth International Conference on Learning and Intelligent Optimization (LION'14)*, Lecture Notes in Computer Science, pages 36–40. Springer, 2014.

[48] S. Pineda, H. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. *arXiv:2106.06257 [cs.LG]*, 2021.

[49] I. Guyon, L. Sun-Hosoya, M. Boullé, H. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W. Tu, and E. Viegas. Analysis of the AutoML challenge series 2015-2018. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, volume 5 of *The Springer Series on Challenges in Machine Learning*. Springer, 2019. Available for free at http://automl.org/book.

[50] Z. Liu, A. Pavao, Z. Xu, S. Escalera, F. Ferreira, I. Guyon, S. Hong, F. Hutter, R. Ji, J. Jacques Junior, G. Li, M. Lindauer, Z. Luo, M. Madadi, T. Nierhoff, K. Niu, C. Pan, D. Stoll, S. Treguer, J. Wang, P. Wang, C. Wu, , Y. Xiong, A. Zela, and Y. Zhang. Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019. *hal-02957135*, 2020.

[51] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *arXiv:2104.10201 [cs.LG]*, 2021.

[52] J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pages 2960–2968. Curran Associates, 2012.

[53] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In B. Berendt, B. Bringmann, E. Fromont, G. Garriga, P. Miettinen, N. Tatti, and V. Tresp, editors, *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'16)*, volume 9851 of *Lecture Notes in Artificial Intelligence*, pages 199–214. Springer, 2016.

[54] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, volume 97, pages 7105–7114. Proceedings of Machine Learning Research, 2019.

[55] L. Metz, N. Maheswaranathan, R. Sun, C. Freeman, B. Poole, and J. Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv:2002.11887 [cs.LG]*, 2020.

[56] K. Eggensperger, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, pages 1114–1120. AAAI Press, 2015.

[57] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1): 15–41, 2018.

[58] V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, pages 12751–12761. Curran Associates, 2018.

[59] R. Martinez-Cantin. Funneled Bayesian optimization for design, tuning and control of autonomous systems. *IEEE Transactions on Cybernetics*, 49(4):1489–1500, 2019.

[60] A. Klein, Z. Dai, F. Hutter, N. Lawrence, and J. Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pages 6267–6277. Curran Associates, 2019.

[61] E. Daxberger, A. Makarova, M. Turchetta, and A. Krause. Mixed-variable bayesian optimization. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pages 2633–2639. ijcai.org, 2020.

[62] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv:2008.09777 [cs.LG]*, 2020.

[63] G. Kurtzer, V. Sochat, and M. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5), 2017.

[64] A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv:1905.04970 [cs.LG]*, 2019.

[65] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`.

[66] A. Zela, J. Siems, and F. Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`.

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

[68] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794. ACM Press, 2016.

[69] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An open source automl benchmark. In K. Eggensperger, M. Feurer, F. Hutter, and J. Vanschoren, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2019)*, 2019.

[70] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[71] A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. Griffiths, A. Maraval, H. Jianye, J. Wang, J. Peters, and H. Ammar. An empirical study of assumptions in bayesian optimisation. *arXiv:2012.03826 [cs.LG]*, 2021.

[72] D. Eriksson, M. Pearce, J. Gardner, R. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*. Curran Associates, 2019.

[73] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

[74] K. Kandasamy, K. Vysyaraju, W. Neiswanger, B. Paria, C. Collins, J. Schneider, B. Poczos, and E. Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with Dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

[75] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. *arXiv:1712.05889v2 [cs.DC]*, 2018.

[76] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyper-parameter optimization framework. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD'19*, pages 2623–2631. ACM Press, 2019.

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

[78] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020.

[79] F. Hutter and M. Osborne. A kernel for hierarchical parameter spaces. *arXiv:1310.5738v1 [stats.ML]*, 2013.

[80] M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter. Towards assessing the impact of bayesian optimization's own hyperparameters. In P. De Causmaecker, M. Lombardi, and Y. Zhang, editors, *IJCAI 2019 DSO Workshop*, 2019.

[81] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[82] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi. Sherpa: Robust hyperparameter optimization for machine learning. *SoftwareX*, 12:100591, 2020.

[83] Y. Xiao, E. Xing, and W. Neiswanger. Amortized auto-tuning: Cost-efficient transfer optimization for hyperparameter recommendation. *arXiv:2106.09179 [cs.LG]*, 2021.

[84] L. Cardoso, V. Santos, R. Francês, R. Prudêncio, and R. Alves. Data vs classifiers, who wins? *arXiv:2107.07451 [cs.LG]*, 2021.

[85] T. Gebru, J. Morgenstern, B. Vecchione, J. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for datasets. *arXiv:1803.09010 [cs.DB]*, 2020.

[86] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs.LG]*, 2016.

[87] B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. Mantovani, J. N. van Rijn, and J. Vanschoren. Openml benchmarking suites. *arXiv:1708.03731v2*, 2019.

[88] D. Molina, A. Latorre, and F. Herrera. An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions. *Cognitive Computation*, 10:517–544, 2018.

[89] A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: a TensorFlow library for applied reinforcement learning. Web page, 2017. URL `https://github.com/tensorforce/tensorforce`.

[90] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs.LG]*, 2017.

[91] T. Chen, E.B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*. Omnipress, 2014.

[92] J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*. Curran Associates, 2016.

[93] D. Dua and C. Graff. UCI machine learning repository, 2019. URL `http://archive.ics.uci.edu/ml`.

[94] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. De Fauw, M. Heilman, diogo149, B. McFee, H. Weideman, takacsg84, peterderivaz, Jon, instagibbs, K. Rasul, CongLiu, Britefury, and J. Degrave. Lasagne: First release., 2015.

[95] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv:1605.02688 [cs.SC]*, 2016.

[96] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[97] F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada, 2009.

[98] M. Lindauer, K. Eggensperger, M. Feurer, S. Falkner, A. Biedenkapp, and F. Hutter. SMACv3: Algorithm configuration in Python. `github.com/automl/SMAC3`, 2017.

[99] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Bentzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems 2*, volume 2, pages 230–246, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Yes, we release all scripts here: `https://github.com/automl/HPOBenchExperimentUtils`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1 and the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We do where applicable (see figures in the appendix), but we also compute statistical tests to quantify the statistical significance of our results (see Tables 2, Table 3 and Appendix H).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix E and Appendix F.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Table 1
   (b) Did you mention the license of the assets? [Yes] See Section 1.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our benchmarks are available here: `https://github.com/automl/HPOBench`

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

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

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

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

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

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