OpenBox: A Generalized Black-box Optimization Service

Yang Li[†], Yu Shen[†§], Wentao Zhang[†], Yuanwei Chen[†], Huaijun Jiang[†§], Mingchao Liu[†]
Jiawei Jiang[‡], Jinyang Gao[◇], Wentao Wu[*], Zhi Yang[†], Ce Zhang[‡], Bin Cui[†∇]
[†] Key Laboratory of High Confidence Software Technologies (MOE), School of EECS, Peking University, China
[‡]Department of Computer Science, Systems Group, ETH Zurich, Switzerland
[∇]Institute of Computational Social Science, Peking University (Qingdao), China
[*]Microsoft Research, USA [◇]Alibaba Group, China [§]Kuaishou Technology, China
[†]{liyang.cs, shenyu, wentao.zhang, yw.chen, jianghuaijun, by_liumingchao, yangzhi, bin.cui}@pku.edu.cn
[‡]{jiawei.jiang, ce.zhang}@inf.ethz.ch [*]wentao.wu@microsoft.com [◇]jinyang.gjy@alibaba-inc.com

ABSTRACT

Black-box optimization (BBO) has a broad range of applications, including automatic machine learning, engineering, physics, and experimental design. However, it remains a challenge for users to apply BBO methods to their problems at hand with existing software packages, in terms of applicability, performance, and efficiency. In this paper, we build OpenBox, an open-source and general-purpose BBO service with *improved usability*. The modular design behind OpenBox also facilitates flexible abstraction and optimization of basic BBO components that are common in other existing systems. OpenBox is distributed, fault-tolerant, and scalable. To improve efficiency, OpenBox further utilizes "algorithm agnostic" parallelization and transfer learning. Our experimental results demonstrate the effectiveness and efficiency of OpenBox compared to existing systems.

CCS CONCEPTS

• **Information systems**;

KEYWORDS

Bayesian Optimization, Black-box Optimization

ACM Reference Format:
Yang Li[†], Yu Shen[†§], Wentao Zhang[†], Yuanwei Chen[†], Huaijun Jiang[†§], Mingchao Liu[†] and Jiawei Jiang[‡], Jinyang Gao[◇], Wentao Wu[*], Zhi Yang[†], Ce Zhang[‡], Bin Cui[†∇]. 2021. OpenBox: A Generalized Black-box Optimization Service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467061

System/Package	Multi-obj.	FIOC	Constraint	History	Distributed
Hyperopt	×	✓	×	×	✓
Spearmint	×	×	✓	×	×
SMAC3	×	✓	×	×	×
BoTorch	✓	×	✓	×	×
GPflowOpt	✓	×	✓	×	×
Vizier	×	✓	×	△	✓
HyperMapper	✓	✓	✓	×	×
HpBandSter	×	✓	×	×	✓
OpenBox	✓	✓	✓	✓	✓

Table 1: A taxonomy of BBO systems/softwares. *Multi-obj.* notes whether the system supports multiple objectives or not. *FIOC* indicates if the system supports all Float, Integer, Ordinal and Categorical variables. *Constraint* refers to the support for inequality constraints. *History* represents the ability of the system to inject the prior knowledge from previous tasks in the search. *Distributed* notes if it supports parallel evaluations under a distributed environment. △ means the system cannot support it for many cases. Note that, BoTorch, as a framework, might provide the algorithmic building blocks for a developer to *implement* some of these capacities.

1 INTRODUCTION

Black–box optimization (BBO) is the task of optimizing an objective function within a limited budget for function evaluations. "Black-box" means that the objective function has no analytical form so that information such as the derivative of the objective function is unavailable. Since the evaluation of objective functions is often expensive, the goal of black-box optimization is to find a configuration that approaches the global optimum as rapidly as possible.

Traditional BBO with a single objective has many applications: 1) automatic A/B testing, 2) experimental design [15], 3) knobs tuning in database [45, 47], and 4) automatic hyper-parameter tuning [6, 27, 32, 43], one of the most indispensable components in AutoML systems [1] such as Microsoft's Azure Machine Learning, Google's Cloud Machine Learning, Amazon Machine Learning [34], and IBM's Watson Studio AutoAI, where the task is to minimize the validation error of a machine learning algorithm as a function of its hyper-parameters. Recently, *generalized* BBO emerges and has been applied to many areas such as 1) processor architecture and circuit design [2], 2) resource allocation [18], and 3) automatic chemical design [22], which requires more general functionalities that may not be supported by traditional BBO, such as multiple objectives and constraints. As examples of applications of generalized BBO in the software industry, Microsoft's Smart Buildings project [35] searches for the best smart building designs by minimizing both energy consumption and construction costs (i.e., BBO with multiple objectives); Amazon Web Service aims to optimize the

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467061

performance of machine learning models while enforcing fairness constraints [38] (i.e., BBO with constraints).

Many software packages and platforms have been developed for traditional BBO (see Table 1). Yet, to the best of our knowledge, so far there is no platform that is designed to target generalized BBO. The existing BBO packages have the following three limitations when applied to general BBO scenarios:

(1) Restricted scope and applicability. Restricted by the underlying algorithms, most existing BBO implementations cannot handle diverse optimization problems in a unified manner (see Table 1). For example, `Hyperopt` [6], `SMAC3` [27], and `HpBandSter` [13] can only deal with single-objective problems without constraints. Though `BoTorch` [3] and `GPflowOpt` [30] can be used, as a framework, for developers to implement new optimization problems with multi-objectives and constraints; nevertheless, their current off-the-shelf supports are also limited (e.g., the support for non-continuous parameters).

(2) Unstable performance across problems. Most existing software packages only implement one or very few BBO algorithms. According to the "no free lunch" theorem [26], no single algorithm can achieve the best performance for all BBO problems. Therefore, existing packages would inevitably suffer from unstable performance when applied to different problems. Figure 1 presents a brief example of hyper-parameter tuning across 25 AutoML tasks, where for each problem we rank the packages according to their performances. We can observe that all packages exhibit unstable performance, and no one consistently outperforms the others. This poses challenges on practitioners to select the best package for a specific problem, which usually requires deep domain knowledge/-expertise and is typically very time-consuming.

(3) Limited scalability and efficiency. Most existing packages execute optimization in a sequential manner, which is inherently inefficient and unscalable. However, extending the sequential algorithm to make it parallelizable is nontrivial and requires significant engineering efforts. Moreover, most existing systems cannot support *transfer learning* to accelerate the optimization on a similar task.

With these challenges, in this paper we propose OpenBox, a system for *generalized* black-box optimization. The design of OpenBox follows the philosophy of providing "BBO as a service" — instead of developing another software package, we opt to implement OpenBox as a distributed, fault-tolerant, scalable, and efficient *service*, which addresses the aforementioned challenges in a uniform manner and brings additional advantages such as ease of use, portability, and zero maintenance. In this regard, Google's `Vizier` [19] is perhaps the only existing BBO service as far as we know that follows the same design philosophy. Nevertheless, `Vizier` only supports traditional BBO, and cannot be applied to general scenarios with multiple objectives and constraints that OpenBox aims for. Moreover, unlike `Vizier`, which remains Google's internal service as of today, we have open-sourced OpenBox that is available at https://github.com/PKU-DAIR/open-box.

The key novelty of OpenBox lies in both the system implementation and algorithm design. In terms of system implementation, OpenBox allows users to define their tasks and access the generalized BBO service conveniently via a task description language (TDL) along with customized interfaces. OpenBox also introduces a high-level parallel mechanism by decoupling basic components in common optimization algorithms, which is "algorithm agnostic" and enables parallel execution in both synchronous and asynchronous settings. Moreover, OpenBox also provides a general transfer-learning framework for generalized BBO, which can leverage the prior knowledge acquired from previous tasks to improve the efficiency of the current optimization task. In terms of algorithm design, OpenBox can host most of the state-of-the-art optimization algorithms and make their performances more *stable* via an *automatic* algorithm selection module, which can choose proper optimization algorithm for a given problem automatically. Furthermore, OpenBox also supports *multi-fidelity* and *early-stopping* algorithms for further optimization of algorithm efficiency.

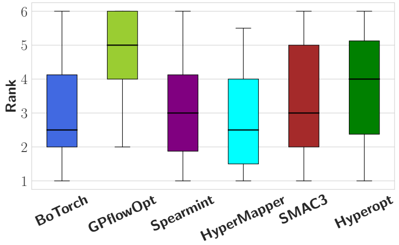


Figure 1: Performance rank of softwares on 25 AutoML tasks (*lower is better*). The box extends from the lower to the upper quartile values, with a line at the median. The whiskers that extend the box show the range of the data.

Contributions. In summary, our main contributions are:

C1. An open-sourced service for generalized BBO. To the best of our knowledge, OpenBox is the first open-sourced service for efficient and general black-box optimization.

C2. Ease of use. OpenBox provides user-friendly interfaces, visualization, resource-aware management, and automatic algorithm selection for consistent performance.

C3. High efficiency and scalability. We develop scalable and general frameworks for transfer-learning and distributed parallel execution in OpenBox. These building blocks are properly integrated to handle diverse optimization scenarios efficiently.

C4. State-of-the-art performance. Our empirical evaluation demonstrates that OpenBox achieves state-of-the-art performance compared to existing systems over a wide range of BBO tasks.

Moving Forward. With the above advantages and features, OpenBox can be used for optimizing a wide variety of different applications in an industrial setting. We are currently conducting an initial deployment of OpenBox in Kuaishou, one of the most popular "short video" platforms in China, to automate the tedious process of hyperparameter tuning. Initial results have suggested we can outperform human experts.

2 BACKGROUND AND RELATED WORK

Generalized Black-box Optimization (BBO). Black-box optimization makes few assumptions about the problem, and is thus applicable in a wide range of scenarios. We define the generalized BBO problem as follows. The objective function of generalized BBO is a vector-valued black-box function $f(x) : \mathcal{X} \rightarrow \mathbb{R}^p$, where $\mathcal{X}$ is the search space of interest. The goal is to identify the set of

Pareto optimal solutions $\mathcal{P}^* = \{\boldsymbol{f}(\boldsymbol{x}) \text{ s.t. } \nexists\, \boldsymbol{x'} \in \mathcal{X} : \boldsymbol{f}(\boldsymbol{x'}) \prec \boldsymbol{f}(\boldsymbol{x})\}$, such that any improvement in one objective means deteriorating another. To approximate $\mathcal{P}^*$, we compute the finite Pareto set $\mathcal{P}$ from observed data $\{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^n$. When $p = 1$, the problem becomes single-objective BBO, as $\mathcal{P} = \{y_{\text{best}}\}$ where y_{best} is defined as the best objective value observed. We also consider the case with black-box inequality constraints. Denote the set of feasible points by $C = \{\boldsymbol{x} : c_1(\boldsymbol{x}) \leq 0, \ldots, c_q(\boldsymbol{x}) \leq 0\}$. Under this setting, we aim to identify the feasible Pareto set $\mathcal{P}_{\text{feas}} = \{\boldsymbol{f}(\boldsymbol{x}) \text{ s.t. } \boldsymbol{x} \in C,\ \nexists\, \boldsymbol{x'} \in \mathcal{X} : \boldsymbol{f}(\boldsymbol{x'}) \prec \boldsymbol{f}(\boldsymbol{x}),\ \boldsymbol{x'} \in C\}$.

Black-box Optimization Methods. Black-box optimization has been studied extensively in many fields, including derivative-free optimization [41], Bayesian optimization (BO) [42], evolutionaray algorithms [23], multi-armed bandit algorithms [31, 44], etc. To optimize expensive-to-evaluate black-box functions with as few evaluations as possible, OpenBox adopts BO, one of the most prevailing frameworks in BBO, as the basic optimization framework. BO iterates between fitting probabilistic surrogate models and determining which configuration to evaluate next by maximizing an acquisition function. With different choices of acquisition functions, BO can be applied to generalized BBO problems.

BBO with Multiple Objectives. Many multi-objective BBO algorithms have been proposed [4, 5, 25, 29, 37]. Couckuyt et. al. [7] propose the Hypervolume Probability of Improvement (HVPOI); Yang et. al. [46] and Daulton et. al. [8] use the Expected Hypervolume Improvement (EHVI) metrics.

BBO with Black-box Constraints. Gardner et.al. [16] present Probability of Feasibility (PoF), which uses GP surrogates to model the constraints. In general, multiplying PoF with the unconstrained acquisition function produces the constrained version of it. SCBO [12] employs the trust region method and scales to large batches by extending Thompson sampling to constrained optimization. Other methods handle constraints in different ways [21, 24, 39]. For multi-objective optimization with constraints, PESMOC [17] and MESMOC [5] support constraints by adding the entropy of the conditioned predictive distribution.

BBO Systems and Packages. Many of these algorithms have available open-source implementations. `BoTorch`, `GPflowOpt` and `HyperMapper` implement several BO algorithms to solve mathematical problems in different settings. Within the machine learning community, `Hyperopt`, `Spearmint`, `SMAC3` and `HpBandSter` aim to optimize the hyper-parameters of machine learning models. Google's `Vizier` is one of the early attempts in building service for BBO. We also note that Facebook Ax[1] provides high-level API for BBO with BoTorch as its Bayesian optimization engine.

3 SYSTEM OVERVIEW

In this section, we provide the basic concepts in the paper, explore the design principles in implementing black-box optimization (BBO) as a service, and describe the system architecture.

3.1 Definitions

Throughout the paper, we use the following terms to describe the semantics of the system:

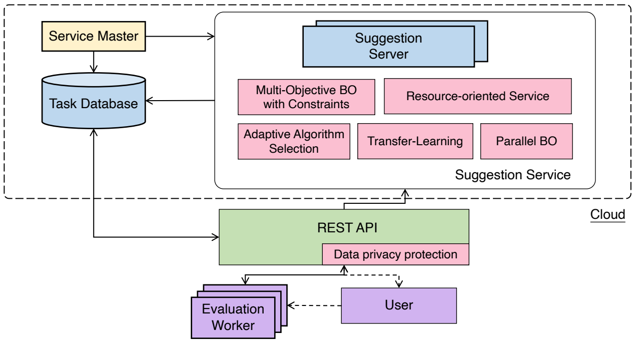


Figure 2: Architecture of OpenBox.

Configuration. Also called suggestion, a vector $\boldsymbol{x}$ sampled from the given search space $\mathcal{X}$; each element in $\boldsymbol{x}$ is an assignment of a parameter from its domain.

Trial. Corresponds to an evaluation of a configuration $\boldsymbol{x}$, which has three status: Completed, Running, Ready. Once a trial is completed, we can obtain the evaluation result $\boldsymbol{f}(\boldsymbol{x})$.

Task. A BBO problem over a search space $\mathcal{X}$. The task type is identified by the number of objectives and constraints.

Worker. Refers to a process responsible for executing a trial.

3.2 Goals and Principles

3.2.1 Design Goal. As mentioned before, OpenBox's design satisfies the following desiderata:

- *Ease of use.* Minimal user effort, and user-friendly visualization for tracking and managing BBO tasks.
- *Consistent performance.* Host state-of-the-art optimization algorithms; choose the proper algorithm automatically.
- *Resource-aware management.* Give cost-model based advice to users, e.g., minimal workers or time-budget.
- *Scalability.* Scale to dimensions on the number of input variables, objectives, tasks, trials, and parallel evaluations.
- *High efficiency.* Effective use of parallel resources, system optimization with transfer-learning and multi-fidelities, etc.
- *Fault tolerance, extensibility, and data privacy protection.*

3.2.2 Design Principles. We present the key principles underlying the design of OpenBox.

P1: Provide convenient service API that abstracts the implementation and execution complexity away from the user. For ease of use, we adopt the "*BBO as a service*" paradigm and implement OpenBox as a managed general service for black-box optimization. Users can access this service via `REST API` conveniently (see Figure 2), and do not need to worry about other issues such as environment setup, software maintenance, programming, and optimization of the execution. Moreover, we also provide a `Web UI`, through which users can easily track and manage the tasks.

P2: Separate optimization algorithm selection complexity away from the user. Users do not need to disturb themselves with choosing the proper algorithm to solve a specific problem via the automatic algorithm selection module. Furthermore, an important decision is to keep our service *stateless* (see Figure 2), so that we can seamlessly switch algorithms during a task, i.e., dynamically choose the algorithm that is likely to perform the best for a particular task.

[1] https://github.com/facebook/ax

```
task_config = {
  "parameter": {
    "x1": { "type": "float", "default": 0,
      "bound": [-5, 10]},
    "x2": {"type": "int", "bound": [0, 15]},
    "x3": {"type": "cat", "default": "a1",
      "choice": ["a1", "a2", "a3"]},
    "x4": {"type": "ord", "default": 1,
      "choice": [1, 2, 3]}},
  "condition": {
      "cdn1": {"type": "equal", "parent": "x3",
        "child": "x1", "value": "a3"}},
  "number_of_trials": 200,
  "time_budget": 10800,
  "task_type": "soc",
  "parallel_strategy": "async",
  "worker_num": 10,
  "use_history": True
  }
```

Figure 3: An example of Task Description Language.

This enables OpenBox to achieve satisfactory performance once the BBO algorithm is selected properly.

P3: Support general distributed parallelization and transfer learning. We aim to provide users with full potential to improve the efficiency of the BBO service. We design an "algorithm agnostic" mechanism that can parallelize the BBO algorithms (Sec. 5.1), through which we do not need to re-design the parallel version for each algorithm individually. Moreover, if the optimization history over similar tasks is provided, our transfer learning framework can leverage the history to accelerate the current task (Sec. 5.2).

P4: Offer resource-aware management that saves user expense. OpenBox implements a resource-aware module and offers advice to users, which can save expense or resources for users especially in the cloud environment. Using performance-resource extrapolation (Sec. 4.4), OpenBox can estimate 1) the minimal number of workers users need to complete the current task within the given time budget, or 2) the minimal time budget to finish the current task given a fixed number of workers. For tasks that involve expensive-to-evaluate functions, low-fidelity or early-stopped evaluations with less cost could help accelerate the convergence of the optimization process (Sec. 5.3).

3.3 System Architecture

Based on these design principles, we build OpenBox as depicted in Figure 2, which includes five main components. *Service Master* is responsible for node management, load balance, and fault tolerance. *Task Database* holds the states of all tasks. *Suggestion Service* creates new configurations for each task. *REST API* establishes the bridge between users/workers and suggestion service. *Evaluation workers* are provided and owned by the users.

4 SYSTEM DESIGN

In this section, we elaborate on the main features and components of OpenBox from a service perspective.

4.1 Service Interfaces

4.1.1 Task Description Language. For ease of usage, we design a Task Description Language (TDL) to define the optimization task. The essential part of TDL is to define the search space, which includes the type and bound for each parameter and the relationships among them. The parameter types — `FLOAT`, `INTEGER`, `ORDINAL` and `CATEGORICAL` are supported in OpenBox. In addition, users can add conditions of the parameters to further restrict the search space. Users can also specify the time budget, task type, number of workers, parallel strategy and use of history in TDL. Figure 3 gives an example of TDL. It defines four parameters `x1-4` of different types and a condition `cdn1`, which indicates that `x1` is active only if `x3 = ‘‘a3’’`. The time budget is three hours, the parallel strategy is `async`, and transfer learning is enabled.

4.1.2 Basic Workflow. Given the TDL for a task, the basic workflow of OpenBox is implemented as follows:

```
# Register the worker with a task.
global_task_id = worker.CreateTask(task_tdl)
worker.BindTask(global_task_id)
while not worker.TaskFinished():
    # Obtain a configuration to evaluate.
    config = worker.GetSuggestions()
    # Evaluate the objective function.
    result = Evaluate(config)
    # Report the evaluated results to the server.
    worker.UpdateObservations(config, result)
```

Here `Evaluate` is the evaluation procedure of objective function provided by users. By calling `CreateTask`, the worker obtains a globally unique identifier `global_task_id`. All workers registered with the same `global_task_id` are guaranteed to link with the same task, which enables parallel evaluations. While the task is not finished, the worker continues to call `GetSuggestions` and `UpdateObservations` to pull suggestions from the suggestion service and update their corresponding observations.

4.1.3 Interfaces. Users can interact with the OpenBox service via a `REST API`. We list the most important service calls as follows:

- `Register`: It takes as input the `global_task_id`, which is created when calling `CreateTask` from workers, and binds the current worker with the corresponding task. This allows for sharing the optimization history across multiple workers.
- `Suggest`: It suggests the next configurations to evaluate, given the historical observations of the current task.
- `Update`: This method updates the optimization history with the observations obtained from workers. The observations include three parts: the values of the objectives, the results of constraints, and the evaluation information.
- `StopEarly`: It returns a boolean value that indicates whether the current evaluation should be stopped early.
- `Extrapolate`: It uses performance-resource extrapolation, and interactively gives resource-aware advice to users.

4.2 Automatic Algorithm Selection

OpenBox implements a wide range of optimization algorithms to achieve high performance in various BBO problems. Unlike the existing software packages that use the same algorithm for each task and the same setting for each algorithm, OpenBox chooses the proper algorithm and setting according to the characteristic of the incoming task. We use the classic EI [36] for single-objective optimization task. For multi-objective problems, we select EHVI [11] when the number of objectives is less than 5; we use MESMO [4] algorithm for problems with a larger number of objectives, since EHVI's complexity increases exponentially as the number of objectives increases, which not only incurs a large computational overhead but also accumulates floating-point errors. We select the surrogate models in BO depending on the configuration space and the number of trials: If the input space has conditions, such as one parameter must be less than another parameter, or there are over 50 parameters in the input space, or the number of trials exceeds 500, we choose the Probabilistic Random Forest proposed in [27] instead of Gaussian Process (GP) as the surrogate to avoid incompatibility or high computational complexity of GP. Otherwise, we use GP [10]. In addition, OpenBox will use the *L-BFGS-B* algorithm to optimize the acquisition function if the search space only contains `FLOAT` and `INTEGER` parameters; it applies an interleaved local and random search when some of the parameters are not numerical. More details about the algorithms implemented in OpenBox are discussed in Appendix A.2.

4.3 Parallel Infrastructure

OpenBox is designed to generate suggestions for a large number of tasks concurrently, and a single machine would be insufficient to handle the workload. Our suggestion service is therefore deployed across several machines, called *suggestion servers*. Each *suggestion server* generates suggestions for several tasks in parallel, giving us a massively scalable suggestion infrastructure. Another main component is *service master*, which is responsible for managing the *suggestion servers* and balancing the workload. It serves as the unified endpoint, and accepts the requests from workers; in this way, each worker does not need to know the dispatching details. The worker requests new configurations from the *suggestion server* and the *suggestion server* generates these configurations based on an algorithm determined by the automatic algorithm selection module. Concretely, in this process, the suggestion server utilizes the local penalization based parallelization mechanism (Sec. 5.1) and transfer-learning framework (Sec. 5.2) to improve the sample efficiency.

One main design consideration is to maintain a fault-tolerant production system, as machine crash happens inevitably. In OpenBox, the *service master* monitors the status of each server and preserves a table of active servers. When a new task comes, the *service master* will assign it to an active server and record this binding information. If one server is down, its tasks will be dispatched to a new server by the master, along with the related optimization history stored in the task database. Load balance is one of the most important guidelines to make such task assignments. In addition, the snapshot of *service master* is stored in the remote database service; if the master is down, we can recover it by restarting the node and fetching the snapshot from the database.

4.4 Performance-Resource Extrapolation

In the setting of parallel infrastructure with cloud computing, saving expense is one of the most important concerns from users. OpenBox can guide users to configure their resources, e.g., the minimal number of workers or time budget, which further saves expense for users. Concretely, we use a weighted cost model to extrapolate the performance vs. trial curve. It uses several parametric decreasing saturating function families as base models, and we apply MCMC inference to estimate the parameters of the model. Given the existing observations, OpenBox trains a cost model as above and uses it to predict the number of trials at which the curve approaches the optimum. Based on this prediction and the cost of each evaluation, OpenBox estimates the minimal resource needed to reach satisfactory performance (more details in Appendix A.1).

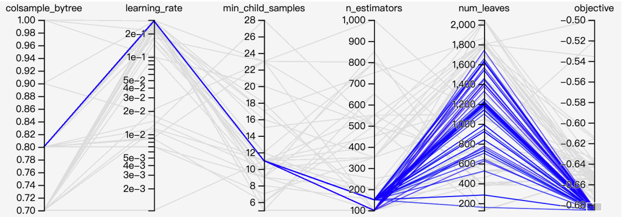


Figure 4: An example of the Parallel Coordinates Visualization for configurations when tuning *LightGBM*.

Application Example. Two interesting applications that save expense for users are listed as follows:
Case 1. Given a fixed number of workers, OpenBox outputs a minimal time budget B_{min} to finish this task based on the estimated evaluation cost of workers. With this estimation, users can stop the task in advance if the given time budget $B_{\text{task}} > B_{\text{min}}$; otherwise, users should increase the time budget to B_{min}.
Case 2. Given a fixed time budget B_{task} and initial number of workers, OpenBox can suggest the minimal number of workers N_{min} to finish the current task within B_{task} by adjusting the number of workers to N_{min} dynamically.

4.5 Augmented Components in OpenBox

Extensibility and Benchmark Support. OpenBox's modular design allows users to define their suggestion algorithms easily by inheriting and implementing an abstract `Advisor`. The key abstraction method of `Advisor` is `GetSuggestions`, which receives the observations of the current task and suggests the next configurations to evaluate based on the user-defined policy. In addition, OpenBox provides a *benchmark suite* of various BBO problems to benchmark the optimization algorithms.
Data Privacy Protection. In some scenarios, the names and ranges of parameters are sensitive, e.g., in hyper-parameter tuning, the parameter names may reveal the architecture details of neural networks. To protect data privacy, the `REST API` applies a transformation to anonymize the parameter-related information before sending it to the service. This transformation involves 1) converting the parameter names to some regular ones like "param1" and 2) rescaling each parameter to a default range that has no semantic. The workers can perform an inverse transformation when receiving an anonymous configuration from the service.
Visualization. OpenBox provides an online dashboard based on `TensorBoardX` which enables users to monitor the optimization process and check the evaluation info of the current task. Figure 4 visualizes the evaluation results in a hyper-parameter tuning task.

5 SYSTEM OPTIMIZATIONS

5.1 Local Penalization based Parallelization

Most proposed Bayesian optimization (BO) approaches only allow the exploration of the parameter space to occur sequentially. To fully utilize the computing resources in a parallel infrastructure, we provide a mechanism for distributed parallelization, where multiple configurations can be evaluated concurrently across workers. Two parallel settings are considered (see Figure 5):

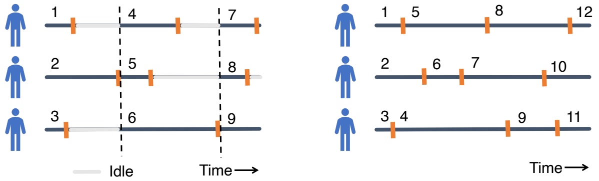


Figure 5: An illustration of the synchronous (left) and asynchronous (right) parallel methods using three workers. The numbers above the horizontal lines are the configuration ids, and the short vertical lines indicate when a worker finished the evaluation of last configuration.

1) Synchronous parallel setting. The worker pulls new configuration from *suggestion server* to evaluate until all the workers have finished their last evaluations.

2) Asynchronous parallel setting. The worker pulls a new configuration when the previous evaluation is completed.

Our main concern is to design an algorithm-agnostic mechanism that can parallelize the optimization algorithms under the sync and async settings easily, so we do not need to implement the parallel version for each algorithm individually. To this end, we propose a local penalization based parallelization mechanism, the goal of which is to sample new configurations that are promising and far enough from the configurations being evaluated by other workers. This mechanism can handle the well-celebrated exploration vs. exploitation trade-off, and meanwhile prevent workers from exploring similar configurations. Algorithm 1 gives the pseudo-code of sampling a new configuration under the sync/async settings. More discussion about this is provided in Appendix A.4.

5.2 General Transfer-Learning Framework

When performing BBO, users often run tasks that are similar to previous ones. This fact can be used to speed up the current task. Compared with `Vizier`, which only provides limited transfer learning functionality for single-objective BBO problems, OpenBox employs a general transfer learning framework with the following advantages: 1) support for the generalized black-box optimization problems, and 2) compatibility with most BO methods.

OpenBox takes as input observations from $K + 1$ tasks: D^1, ..., D^K for K previous tasks and D^T for the current task. Each $D^i = \{(\boldsymbol{x}_j^i, \boldsymbol{y}_j^i)\}_{j=1}^{n_i}$, $i = 1, ..., K$, includes a set of observations. Note that, $\boldsymbol{y}$ is an array, including multiple objectives for configuration $\boldsymbol{x}$.

For multi-objective problems with p objectives, we propose to transfer the knowledge about p objectives individually. Thus, the transfer learning of multiple objectives is turned into p single-objective transfer learning processes. For each dimension of the objectives, we take RGPE [14] as the base method. 1) We first train a surrogate model M^i on D^i for the i^{th} prior task and M^T on D^T; based on $M^{1:K}$ and M^T, we then build a transfer learning surrogate by combining all base surrogates:

$$M^{\text{TL}} = \texttt{agg}(\{M^1, ..., M^K, M^T\}; \mathbf{w});$$

3) the surrogate M^{TL} is used to guide the configuration search, instead of the original M^T. Concretely, we combine the multiple base surrogates (agg) linearly, and the parameters $\mathbf{w}$ are calculated based on the ranking of configurations, which reflects the similarity between the source tasks and the target task (see details in Appendix A.3).

Algorithm 1: Pseudo code for *Sample* configuration

```
Input: the hyper-parameter space $\mathcal{X}$, configuration observations $D = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^n$,
       configurations being evaluated $C_{\text{eval}}$, surrogate model $M$, and acquisition
       function $\alpha(\cdot)$.
calculate $\hat{\boldsymbol{y}}$, the median of $\{\boldsymbol{y_i}\}_{i=1}^n$;
create new observations $D_{\text{new}} = \{(\boldsymbol{x}_{\text{eval}}, \hat{\boldsymbol{y}}) : \boldsymbol{x}_{\text{eval}} \in C_{\text{eval}}\}$;
fit a surrogate model $M$ (e.g., a GP) on $D_{\text{aug}}$, where $D_{\text{aug}} = D \cup D_{\text{new}}$, and build the
   acquisition function $\alpha(\boldsymbol{x}, M)$ using $M$;
return the configuration $\bar{\boldsymbol{x}} = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}} \alpha(\boldsymbol{x}, M)$.
```

Scalability discussion A more intuitive alternative is to obtain a transfer learning surrogate by using all observations from $K + 1$ tasks, and this incurs a complexity of $O(k^3 n^3)$ for k tasks with n trials each (since GP has $O(n^3)$ complexity). Therefore, it is hard to scale to a larger number of source tasks (a large k). By training base surrogates individually, the proposed framework is a more computation-efficient solution that has $O(kn^3)$ complexity.

5.3 Additional Optimizations

OpenBox also includes two additional optimizations that can be applied to improve the efficiency of black-box optimizations.

5.3.1 Multi-Fidelity Support and Applications.

During each evaluation in the multi-fidelity setting [33, 40], the worker receives an additional parameter, indicating how many resources are used to evaluate this configuration. The resource type needs to be specified by users. For example, in hyper-parameter tuning, it can be the number of iterations for an iterative algorithm and the size of dataset subset. The trial with partial resource returns a low-fidelity result with a cheap evaluation cost. Though not as precise as high-fidelity results, the low-fidelity results can provide some useful information to guide the configuration search. In OpenBox, we have implemented several multi-fidelity algorithms, such as `MFES-HB` [33].

5.3.2 Early-Stopping Strategy.

Orthogonal to the above optimization, early-stopping strategies aim to stop a poor trial in advance based on its intermediate results. In practice, a worker can periodically ask suggestion service whether it should terminate the current evaluation early. In OpenBox, we provide two early-stopping strategies: 1) learning curve extrapolation based methods [9, 28] that stop the poor configurations by estimating the future performance, and 2) mean or median termination rules based on comparing the current result with previous ones.

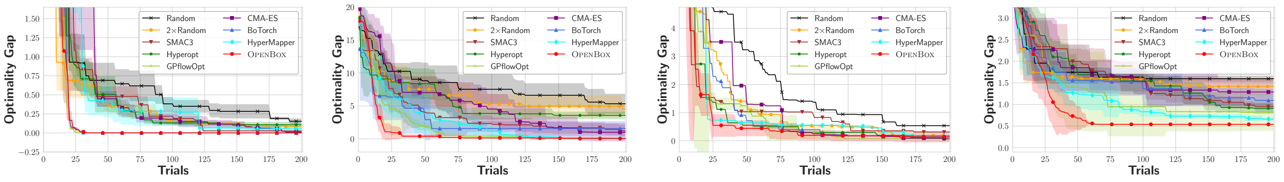
6 EXPERIMENTAL EVALUATION

In this section, we compare the performance and efficiency of OpenBox against existing software packages on multiple kinds of black-box optimization tasks, including tuning tasks in AutoML.

6.1 Experimental Setup

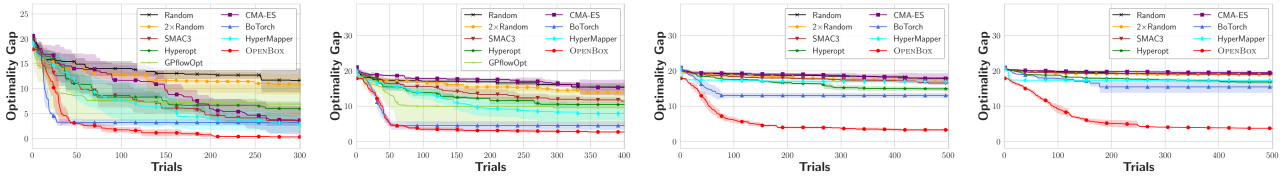
6.1.1 Baselines.

Besides the systems mentioned in Table 1, we also use `CMA-ES` [23], `Random Search` and 2×`Random Search` (Random Search with double budgets) as baselines. To evaluate transfer learning, we compare OpenBox with `Google Vizier`. For multi-fidelity experiments, we compare OpenBox against `HpBandSter` and `BOHB`, the details of which are in Appendix A.5.



(a) 2d-Branin (b) 2d-Ackley (c) 2d-Beale (d) 6d-Hartmann

Figure 6: Results for four black-box problems with single objective.



(a) 4d-Ackley (b) 8d-Ackley (c) 16d-Ackley (d) 32d-Ackley

Figure 7: Scalability results on solving `Ackley` with different input dimensions.

6.1.2 Problems. We use 12 black-box problems (mathematical functions) from [48] and two AutoML optimization problems on 25 OpenML datasets. In particular, 2d-Branin, 2d-Beale, 6d-Hartmann and (2d, 4d, 8d, 16d, 32d)-Ackley are used for single-objective optimization; 2d-Townsend, 2d-Mishra, 4d-Ackley and 10d-Keane are used for constrained single-objective optimization; 3d-ZDT2 with two objectives and 6d-DTLZ1 with five objectives are used for multi-objective optimization; 2d-CONSTR and 2d-SRN with two objectives are used for constrained multi-objective optimization. All the parameters for mathematical problems are of the `FLOAT` type and the maximum trials of each problem depend on its difficulty, which ranges from 80 to 500. For AutoML problems on 25 datasets, we split each dataset and search for the configuration with the best validation performance. Specifically, we tune `LightGBM` and `LibSVM` with the linear kernel, where the parameters of `LightGBM` are of the `FLOAT` type while `LibSVM` contains `CATEGORICAL` and conditioned parameters.

6.1.3 Metrics. We employ the three metrics as follows.
1. Optimality gap is used for single-objective mathematical problem. That is, if x^* optimizes f, and $\hat{x}$ is the best configuration found by the method, then $|f(\hat{x}) - f(x^*)|$ measures the success of the method on that function. In rare cases, we report the objective value if the ground-truth optimal x^* is extremely hard to obtain.
2. Hypervolume indicator given a reference point $\boldsymbol{r}$ measures the quality of a Pareto front in multi-objective problems. We report the *difference* between the hypervolume of the ideal Pareto front $\mathcal{P}^*$ and that of the estimated Pareto front $\mathcal{P}$ by a given algorithm, which is $HV(\mathcal{P}^*, \boldsymbol{r}) - HV(\mathcal{P}, \boldsymbol{r})$.
3. Metric for AutoML. For single-objective AutoML problems, we report the validation error. To measure the results across different datasets, we use `Rank` as the metric.

6.1.4 Parameter Settings. For both OpenBox and the considered baselines, we use the default setting. Each experiment is repeated 10 times, and we compute the mean and variance for visualization.

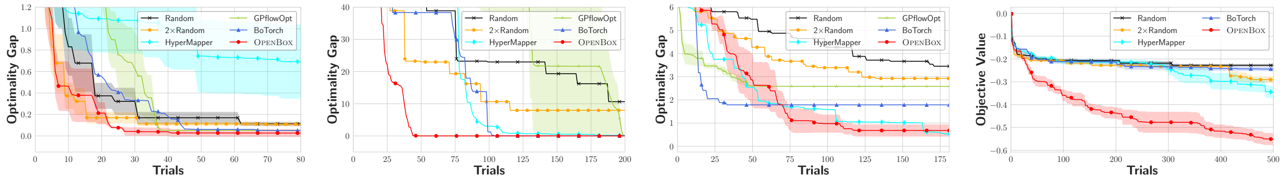
6.2 Results and Analysis

6.2.1 Single-Objective Problems without Constraints. Figure 6 illustrates the results of OpenBox on different single-objective problems compared with competitive baselines while Figure 7 displays the performance with the growth of input dimensions. In particular, Figure 6 shows that OpenBox, `HyperMapper` and `BoTorch` are capable of optimizing these low-dimensional functions stably. However, when the dimensions of the parameter space grow larger, as shown in Figure 7, only OpenBox achieves consistent and excellent results while the other baselines fail, which demonstrates its scalability on input dimensions. Note that, OpenBox achieves more than 10-fold speedups over the baselines when solving `Ackley` with 16 and 32-dimensional inputs.

6.2.2 Single-Objective Problems with Constraints. Figure 8 shows the results of OpenBox along with the baselines on four constrained single-objective problems. Besides `Random Search`, we compare OpenBox with three of the software packages that support constraints. OpenBox surpasses all the considered baselines on the convergence result. Note that on the 10-dimensional `Keane` problem in which the ground-truth optimal value is hard to locate, OpenBox is the only method that successfully optimizes this function while the other methods fail to suggest sufficient feasible configurations.

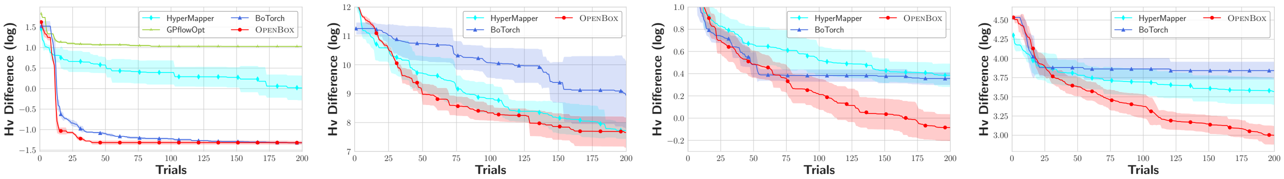
6.2.3 Multi-Objective Problems without Constraints. We compare OpenBox with three baselines that support multiple objectives and the results are depicted in Figure 9(a) and 9(b). In Figure 9(a), the hypervolume difference of `GPflowOpt` and `Hypermapper` decreases slowly as the number of trials grow, while `BoTorch` and OpenBox obtain a satisfactory Pareto Front quickly within 50 trials. In Figure 9(b) where the number of objectives is 5, `BoTorch` meets the bottleneck of optimizing the Pareto front while OpenBox tackles this problem easily by switching its inner algorithm from EHVI to MESMO; `GPflowOpt` is missing due to runtime errors.

6.2.4 Multi-Objective Problems with Constraints. We compare OpenBox with `Hypermapper` and `BoTorch` on constrained multi-objective problems (See Figure 9(c) and 9(d)). Figure 9(c) demonstrates the performance on a simple problem, in which the convergence result of OpenBox is slightly better than the other two baselines. However, in Figure 9(d) where the constraints are strict, `BoTorch` and `Hypermapper` fail to suggest sufficient feasible configurations to update the Pareto Front. Compared with `BoTorch` and `Hypermapper`,



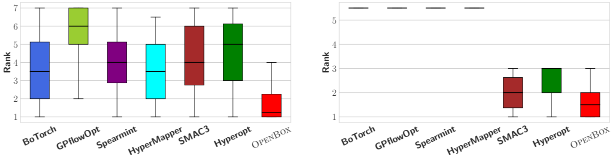
(a) 2d-Townsend (b) 2d-Mishra (c) 4d-Ackley (d) 10d-Keane

Figure 8: Results for solving four single-objective black-box problems with constraints.



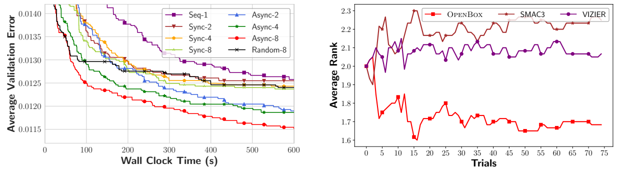
(a) 3d-ZDT2 (b) 6d-DTLZ1 (c) 2d-CONSTR (d) 2d-SRN

Figure 9: Results on multi-objective problems without (a and b) and with (c and d) constraints.



(a) AutoML Benchmark on LightGBM (b) AutoML Benchmark on LibSVM

Figure 10: Performance rank on 25 datasets (the lower is the better). The box extends from the lower to upper quartile values, with a line at the median. The whiskers extend from the box to show the range of the data.



(a) Parallel Experiments on Optdigits (b) Transfer Learning

Figure 11: Average validation error under two parallel settings (left figure) and average rank of tuning `LightGBM` with transfer learning (right figure). "Seq", "Sync" and "Async" refer to the sequential, sync and async mode respectively. The number of parallel workers is given after '-'.

OpenBox has more stable performance when solving multi-objective problems with constraints.

6.3 Results on AutoML Tuning Tasks

6.3.1 AutoML Tuning on 25 OpenML datasets. Figure 11 demonstrates the universality and stability of OpenBox in 25 AutoML tuning tasks. We compare OpenBox with `SMAC3` and `Hyperopt` on `LibSVM` since only these two baselines support `CATEGORICAL` parameters with conditions. In general, OpenBox is capable of handling different types of input parameters while achieving the best median performance among the baselines considered.

6.3.2 Parallel Experiments. To evaluate OpenBox with parallel settings, we conduct an experiment to tune the hyper-parameters of `LightGBM` on `Optdigits` with a budget of 600 seconds. Figure 11(a) shows the average validation error with different parallel settings. We observe that the asynchronous mode with 8 workers achieves the best results and outperforms `Random Search` with 8 workers by a wide margin. It brings a speedup of 8× over the sequential mode, which is close to the ideal speedup. In addition, although the synchronous mode brings a certain improvement over the sequential mode in the beginning, the convergence result is usually worse than the asynchronous mode due to stragglers.

6.3.3 Transfer Learning Experiment. In this experiment, we remove all baselines except `Vizier`, which provides the transfer learning functionality for the traditional black-box optimization. We also add `SMAC3` that provides a non-transfer reference. In addition, this experiment involves tuning `LightGBM` on 25 OpenML datasets, and it is performed in a leave-one-out fashion, i.e, we tune the hyperparameters of `LightGBM` on a dataset (target problem), while taking the tuning history on the remaining datasets as prior observations. Figure 11(b) shows the average rank for each baseline. We observe that 1) `Vizier` and OpenBox show improved sample efficiency relative to `SMAC3` that cannot use prior knowledge from source problems, and 2) the proposed transfer learning framework in OpenBox performs better than the transfer learning algorithm used in `Vizier`. Furthermore, it is worth mentioning that OpenBox also supports transfer learning for the generalized black-box optimization, while `Vizier` does not.

7 CONCLUSION

In this paper, we have introduced a service that aims for solving generalized BBO problems – OpenBox, which is open-sourced and highly efficient. We have presented new principles from a service perspective that drive the system design, and we have proposed efficient frameworks for accelerating BBO tasks by leveraging local-penalization based parallelization and transfer learning. OpenBox hosts lots of state-of-the-art optimization algorithms with consistent performance, via adaptive algorithm selection. It also offers a set of advanced features, such as performance-resource extrapolation, multi-fidelity optimization, automatic early stopping, and data privacy protection. Our experimental evaluations have also showcased the performance and efficiency of OpenBox on a wide range of BBO tasks.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (No.2018YFB1004403), NSFC (No.61832001, U1936104), Beijing Academy of Artificial Intelligence (BAAI), and Kuaishou-PKU joint program. Bin Cui is the corresponding author.

REFERENCES

[1] Leonel Aguilar Melgar, David Dao, Shaoduo Gan, Nezihe M. Gürel, Nora Hollenstein, Jiawei Jiang, Bojan Karlaš, Thomas Lemmin, Tian Li, Yang Li, Susie Rao, Johannes Rausch, Cedric Renggli, Luka Rimanic, Maurice Weber, Shuai Zhang, Zhikuan Zhao, Kevin Schawinski, Wentao Wu, and Ce Zhang. 2021. In *Proceedings of the Annual Conference on Innovative Data Systems Research (CIDR), 2021*. CIDR.

[2] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, New York, NY, USA.

[3] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2020. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *NeurIPS*.

[4] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value entropy search for multi-objective Bayesian optimization. In *NeurIPS*.

[5] Syrine Belakaria, Aryan Deshwal, Nitthilan Kannappan Jayakodi, and Janardhan Rao Doppa. 2020. Uncertainty-aware search framework for multi-objective Bayesian optimization. In *AAAI*, Vol. 34. 10044–10052.

[6] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.

[7] Ivo Couckuyt, Dirk Deschrijver, and Tom Dhaene. 2014. Fast Calculation of Multiobjective Probability of Improvement and Expected Improvement Criteria for Pareto Optimization. *J. of Global Optimization* 60, 3 (2014), 575–594.

[8] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Optimization. *arXiv preprint arXiv:2006.05078* (2020).

[9] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI International Joint Conference on Artificial Intelligence*.

[10] Katharina Eggensperger, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2015. Efficient Benchmarking of Hyperparameter Optimizers via Surrogates.. In *AAAI*. 1114–1120.

[11] M. T. M. Emmerich, K. C. Giannakoglou, and B. Naujoks. 2006. Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation* (2006).

[12] David Eriksson and Matthias Poloczek. 2021. Scalable constrained bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 730–738.

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774* (2018).

[14] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. 2018. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In *AutoML Workshop at ICML*.

[15] Adam Foster, Martin Jankowiak, Eli Bingham, Paul Horsfall, Yee Whye Teh, Tom Rainforth, and Noah Goodman. 2019. Variational bayesian optimal experimental design. *arXiv preprint arXiv:1903.05480* (2019).

[16] Jacob R. Gardner, Matt J. Kusner, Zhixiang Xu, Kilian Q. Weinberger, and John P. Cunningham. 2014. Bayesian Optimization with Inequality Constraints. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org.

[17] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. 2019. Predictive entropy search for multi-objective bayesian optimization with constraints. *Neurocomputing* 361 (2019), 50–68.

[18] Michael Adam Gelbart. 2015. *Constrained Bayesian Optimizationand Applications*. Ph.D. Dissertation. Harvard University, Graduate School of Arts & Sciences.

[19] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD*. ACM, 1487–1495.

[20] Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. 2016. Batch bayesian optimization via local penalization. In *AISTATS 2016*. arXiv:1505.08052

[21] Robert B Gramacy, Genetha A Gray, Sébastien Le Digabel, Herbert KH Lee, Pritam Ranjan, Garth Wells, and Stefan M Wild. 2016. Modeling an augmented Lagrangian for blackbox constrained optimization. *Technometrics* 58, 1 (2016), 1–11.

[22] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. 2020. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chem. Sci.* 11 (2020).

[23] N. Hansen and A. Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies.

[24] José Miguel Hernández-Lobato, Michael Gelbart, Matthew Hoffman, Ryan Adams, and Zoubin Ghahramani. 2015. Predictive entropy search for bayesian optimization with unknown constraints. In *International conference on machine learning*. PMLR, 1699–1707.

[25] José Miguel Hernández-Lobato, Michael A. Gelbart, Ryan P. Adams, Matthew W. Hoffman, and Zoubin Ghahramani. 2016. A General Framework for Constrained Bayesian Optimization using Information-based Search. *Journal of Machine Learning Research* 17, 160 (2016), 1–53.

[26] Yu-Chi Ho and David L Pepyne. 2001. Simple explanation of the no free lunch theorem of optimization. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)*, Vol. 5. IEEE, 4409–4414.

[27] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[28] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2017. Learning Curve Prediction With Bayesian Neural Networks. *ICLR* (2017).

[29] Joshua Knowles. 2006. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation* (2006).

[30] Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. 2017. GPflowOpt: A Bayesian Optimization Library using TensorFlow. *arXiv preprint – arXiv:1711.03845* (2017).

[31] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Proceedings of the ICLR* (2018), 1–48.

[32] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient Automatic CASH via Rising Bandits. In *AAAI*, Vol. 34. 4763–4771.

[33] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2020. MFES-HB: Efficient Hyperband with Multi-Fidelity Quality Measurements. *arXiv preprint arXiv:2012.03011* (2020).

[34] Edo Liberty, Zohar Karnin, Bing Xiang, Laurence Rouesnel, Baris Coskun, Ramesh Nallapati, Julio Delgado, Amir Sadoughi, Yury Astashonok, Piali Das, et al. 2020. Elastic Machine Learning Algorithms in Amazon SageMaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 731–737.

[35] Microsoft. 2020. Smart buildings: From design to reality. https://azure.microsoft.com/en-us/resources/smart-buildings-from-design-to-reality/.

[36] J Močkus. 1975. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*. Springer, 400–404.

[37] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. 2020. A flexible framework for multi-objective Bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*. PMLR, 766–776.

[38] Valerio Perrone, Michele Donini, Muhammad Bilal Zafar, Robin Schmucker, Krishnaram Kenthapadi, and Cédric Archambeau. 2020. Fair bayesian optimization. *arXiv preprint arXiv:2006.05109* (2020).

[39] Victor Picheny, Robert B Gramacy, Stefan M Wild, and Sebastien Le Digabel. 2016. Bayesian optimization under mixed constraints with a slack-variable augmented Lagrangian. *arXiv preprint arXiv:1605.09466* (2016).

[40] Matthias Poloczek, Jialei Wang, and Peter Frazier. 2017. Multi-information source optimization. In *Advances in Neural Information Processing Systems*. 4288–4298.

[41] L. M. Rios and N. Sahinidis. 2013. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization* 56 (2013), 1247–1293.

[42] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2016. Taking the human out of the loop: A review of Bayesian optimization.

[43] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *NIPS*. 2951–2959.

[44] N. Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *ICML*.

[45] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 SIGMOD*. 1009–1024.

[46] Kaifeng Yang, Michael Emmerich, André Deutz, and Thomas Bäck. 2019. Multi-Objective Bayesian Global Optimization using expected hypervolume improvement gradient. *Swarm and evolutionary computation* 44 (2019), 945–956.

[47] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.

[48] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8, 2 (2000), 173–195.

A APPENDIX

A.1 Performance-Resource Extrapolation

While optimizing various black-box problems, we observe that the optimization curve (performance vs. trials) is often *saturating*, i.e., after a certain number of trials, more evaluations will not cause a meaningful improvement $\delta > 0$ in performance. OpenBox applies a combined learning curve extrapolation method inspired by [9], which early stops the training procedure of neural networks when the performance of the network becomes less likely to improve.

We measure the performance by negative hypervolume indicator (HV) of the Pareto set $\mathcal{P}$ bounded above by reference point r, denoted by $HV(\mathcal{P}, r)$. In single-objective case, $\mathcal{P} = \{y_{\text{best}}\}$. Note that in both cases, the performance is *decreasing*.

Denote the performance at timestep t by z_t. Given observed data $z_{1:n} := \{z_1, \ldots, z_n\}$, a natural idea is to estimate whether the performance at a future timestep $t > n$ will exceed the current best performance z_n. We extrapolate the performance curve z_t with a weighted probabilistic model

$$g_{\text{comb}}(t|\Theta) = \sum_{k=1}^{K} w_k g_k(t|\boldsymbol{\theta_k}) + \varepsilon,$$

where each of $g_1, \ldots, g_K$ is a parametric family of decreasing saturating functions, and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. We estimate $\Theta = (w_1, \ldots, w_K, \theta_1, \ldots, \theta_K, \sigma^2)$ using Markov Chain Monte Carlo (MCMC) inference. The prior and posterior distribution over Θ are as follows

$$p(\Theta) \propto \Big(\prod_{k=1}^{K} p(w_k)p(\boldsymbol{\theta_k})\Big)p(\sigma^2)\mathbb{1}(g_{\text{comb}}(1|\Theta) > g_{\text{comb}}(t|\Theta)),$$

$$P(\Theta|z_{1:n}) \propto P(z_{1:n}|\Theta)P(\Theta),$$

where $t > n$.

We sample Θ from the posterior and compute $P(z_t < z_n - \delta|z_{1:n})$, which is the probability that the optimization procedure yields a meaningful improvement δ at timestep t.

A.2 Bayesian Optimization Algorithms

The BO algorithms in OpenBox include three parts: surrogate models, acquisition functions, and acquisition function optimizers.

Surrogate Models. OpenBox selects different surrogate models based on the number of trials. For tasks with under 500 trials, OpenBox defaults to using Gaussian Process (GP) from `scikit-optimize` package. We use a Matérn kernel with automatic relevance determination (ARD) for continuous parameters and a Hamming kernel for categorical parameters. When both continuous and categorical parameters exist, we use the product of these two kernels. The parameters of GP are fitted by optimizing the marginal log-likelihood with the gradient-based method (as default) or MCMC sampling. Due to the high computational complexity $O(n^3)$, GP cannot scale well to the setting with too many trials (a large n). Therefore, for tasks with more than 500 trials, the surrogate model is switched to probabilistic random forest proposed in [27], which incurs less complexity.

Acquisition Functions. By default, OpenBox uses Expected Improvement (EI) [36] for single-objective optimization, Expected Hypervolume Improvement (EHVI) [11] for multi-objective optimization, and Probability of Feasibility (PoF) [16] for constraints. OpenBox computes these acquisition functions analytically [46] (by default) or through Monte Carlo integration [8]. In addition, OpenBox includes multiple acquisition functions to meet the needs of different problem settings. For single-objective optimization, Expected Improvement per second (EIPS) [43] can be used to find a good configuration as quickly as possible, and Expected Improvement with Local Penalization (LP-EI) [20] utilizes local penalizers to propose batches of configurations simultaneously. For multi-objective optimization, Max-value Entropy Search for Multi-objective Optimization (MESMO) [4] and Uncertainty-aware Search framework [5] for Multi-objective Optimization (USeMO) work efficiently when the number of objectives is large. Other implemented acquisition functions include Probability of Improvement (PI), and Upper Confidence Bound (UCB) [44].

Acquisition Function Optimizers. To support generic surrogate models that are not differentiable, we maximize the acquisition function via the following two methods: 1) interleaved local and random search (gradient-free) which can handle categorical parameters, and 2) multi-start staged optimizer of random search and L-BFGS-B from `Scipy` (estimate gradient by 2-point finite difference) which can locate the global optimum in high dimensional design space efficiently.

A.3 Transfer Learning Details

In OpenBox, we expand RGPE [14], a state-of-the-art transfer learning method on single-objective problems, into generalized settings.

First, for each prior task i, we train surrogates $M^i_{1:m}$ for m objectives on the corresponding observations from D^i. Then we build surrogates $M^{\text{TL}}_{1:m}$ to guide the optimization instead of using the original surrogates $M^T_{1:m}$ fitted on D^T only. For ease of description, we assume there is only one surrogate M^{TL} since the method of building surrogate for each objective is exactly the same. The prediction of M^{TL} at point $\boldsymbol{x}$ is given by $y \sim \mathcal{N}(\sum_i \mu_{\text{TL}}(\boldsymbol{x}), \sigma^2_{\text{TL}}(\boldsymbol{x}))$, where

$$\mu_{\text{TL}}(\boldsymbol{x}) = (\sum_i \mu_i(\boldsymbol{x})\mathbf{w}_i\sigma_i^{-2}(\boldsymbol{x}))\sigma^2_{\text{TL}}(\boldsymbol{x}),$$

$$\sigma^2_{\text{TL}}(\boldsymbol{x}) = (\sum_i \mathbf{w}_i\sigma_i^{-2}(\boldsymbol{x}))^{-1},$$

where $\mathbf{w}_i$ is the weight of base surrogate M^i, and μ_i and σ_i^2 are the predictive mean and variance from base surrogate M^i. The weight $\mathbf{w}_i$ reflects the similarity between the previous task and current task. Therefore, M^{TL} carries the knowledge of the prior tasks, which could greatly accelerate the convergence of the optimization on the current task. We then use the following ranking loss function L, i.e., the number of misranked pairs, to measure the similarity between previous tasks and current task:

$$L(M^i, D^T) = \sum_{j=1}^{n^T}\sum_{k=1}^{n^T} \mathbb{1}((M^i(\boldsymbol{x}_j) < (\boldsymbol{x}_k) \oplus (y_j < y_k)), \qquad (1)$$

where $\oplus$ is the exclusive-or operator, $n^T = |D^T|$, $\boldsymbol{x}_j$ and y_j are the sampled point and its performance in D^T, and $M^i(\boldsymbol{x}_j)$ means the prediction of M^i on the point $\boldsymbol{x}_j$. Based on the ranking loss function, the weight $\mathbf{w}_i$ is set to the probability that M^i has the smallest ranking loss on D^T, that is, $\mathbf{w}_i = P(i = \text{argmin}_j\, L(M^j, D^T))$. This probability can be estimated using the MCMC sampling.

A.4 Discussions about Local Penalization based Parallelization

Algorithm 1 parallelizes BO algorithms by imputing the configurations being evaluated with the median of the evaluated data $D_n = \{\boldsymbol{x_i}, \boldsymbol{y_i}\}_{i=1}^n$. For notational simplicity, we discuss the single-objective case with EI as acquisition function. Denote the median of observed values $\{y_i\}_{i=1}^n$ by $\hat{y}$, and the smallest observed value by η. Define $u = f(\boldsymbol{x})$, $u \sim \mathcal{N}(\mu_n(\boldsymbol{x}), \sigma_n^2(\boldsymbol{x}))$, where $\mu_n(\boldsymbol{x})$ and $\sigma_n^2(\boldsymbol{x})$ are the mean and variance of the posterior distribution of the surrogate model trained on D_n. The expected improvement is

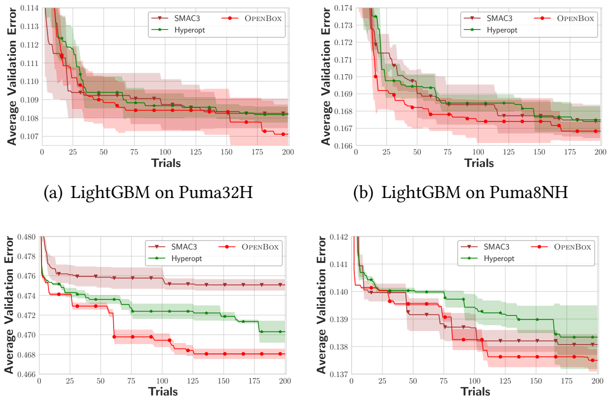
$$\begin{aligned} \alpha_{\mathrm{EI}}(\boldsymbol{x}; D_n) &= \mathbb{E}_u[(\eta - u)\mathbb{1}(u < \eta)] \\ &= (\eta - \mu_n(\boldsymbol{x}))\Phi(z) + \sigma_n(\boldsymbol{x})\phi(z) \end{aligned} \tag{2}$$

when $\sigma_n > 0$ and vanishes otherwise. Here, Φ and ϕ are the CDF and PDF of the standard normal distribution, $z = \frac{\eta - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}$.

We first show that, with our imputation strategy, $\alpha_{\mathrm{EI}}(\boldsymbol{x}; D_{\mathrm{aug}})$ will be sufficiently small if $\boldsymbol{x}$ is close to some $\boldsymbol{x}_{\mathrm{eval}} \in D_{\mathrm{aug}}$, i.e., locally penalized near $\boldsymbol{x}_{\mathrm{eval}}$. For all probabilistic surrogate models, $\mu_n(\boldsymbol{x}) = f(\boldsymbol{x})$, $\sigma_n(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in D_n$, which means $\alpha_{\mathrm{EI}}(\boldsymbol{x}) = 0$, $\forall \boldsymbol{x} \in D_n$. By augmenting D_n with $D_{\mathrm{new}} = \{(\boldsymbol{x}_{\mathrm{eval}}, \hat{y}) : \boldsymbol{x}_{\mathrm{eval}} \in C_{\mathrm{eval}}\}$, we have $\alpha_{\mathrm{EI}}(\boldsymbol{x}_{\mathrm{eval}}) = 0$, $\forall \boldsymbol{x}_{\mathrm{eval}} \in C_{\mathrm{eval}}$. Since $\alpha_{EI}(\boldsymbol{x}; D_{\mathrm{aug}})$ is continuous if the surrogate is GP and flat if the surrogate is random forest, when $\boldsymbol{x}$ is close to some $\boldsymbol{x}_{\mathrm{eval}} \in C_{\mathrm{eval}}$, $\eta - \mu_n(\boldsymbol{x}) \approx \eta - \hat{y}$ and $z = (\eta - \mu_n(\boldsymbol{x}))/\sigma_n(\boldsymbol{x})$ are negative and sufficiently small. Hence, both terms in (2) are small and $\boldsymbol{x}$ is unlikely to be the maximum of α_{EI}. This conclusion can be naturally extended to cases with multiple objectives, and more generally, other acquisition functions.

Moreover, although Algorithm 1 changes the posterior distribution of the surrogate by imposing a local penalty, it helps avoid over-exploitation. Considering the configurations evaluated at the same time as a "batch", Algorithm 1 simplified the complex joint optimization problem by assigning a different region for each worker to explore. From the experiment results shown in Figure 11(a), we observe that Algorithm 1 is a highly efficient, as well as widely applicable parallelization heuristic.

A.5 More Experimental Results



(a) LightGBM on Puma32H

(b) LightGBM on Puma8NH

(c) LibSVM on Pollen

(d) LibSVM on Wind

Figure 12: Performance of two AutoML tasks on 4 datasets.

AutoML Performance. Besides the rank of convergence results shown in Figure 11, we present Figure 12 that demonstrates the optimization process of OpenBox on AutoML tasks. OpenBox achieves 2.0-3.3× speedups over the best baseline in each task.

Muiti-fidelity Acceleration. Figure 13 shows the acceleration of OpenBox using multi-fidelity optimization compared with SMAC3 and two other multi-fidelity packages, HpBandSter and BOHB. The dataset used in this experiment is Covtype, which is a large-scale dataset with over 580k samples. We observe that though HpBandSter and BOHB accelerates the optimization in the beginning, their convergence results are worse than that of SMAC3. However, OpenBox obtains a 3.8 × speedup over SMAC3 when achieving the comparable convergence performance.

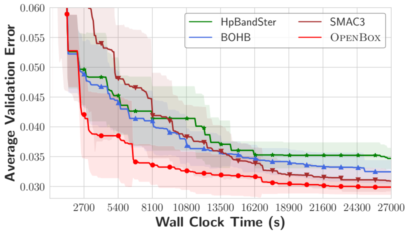


Figure 13: Multi-fidelity experiment on tuning hyper-parameters of LightGBM.

A.6 Reproduction Instructions

We run our experiments on 2 machines with 56 Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. The versions of baselines are 1) BoTorch 0.3.3, 2) GPflowOpt 0.1.1, 3) HyperMapper master branch [2], 4) SMAC3 0.8.0, 5) Hyperopt 0.2.3 and 6) Spearmint master branch [3]. The source code of OpenBox is written in Python 3.7 and is already available in Github [4]. We place the code for reproduction under the directory *test/reproduction*. For example, to run single-objective experiment on Branin, the script is as follows:

```
python test/reproduction/so/benchmark_xxx.py
-problem branin -n 200.
```

[2] https://github.com/luinardi/hypermapper
[3] https://github.com/JasperSnoek/spearmint
[4] https://github.com/PKU-DAIR/open-box