TransBO: Hyperparameter Optimization via Two-Phase Transfer Learning

Yang Li
School of CS, Peking University
Tencent Data Platform, Technology and Engineering Group, Tencent Inc.
Beijing, China
liyang.cs@pku.edu.cn

Yu Shen
Huaijun Jiang
shenyu@pku.edu.cn
jianghuaijun@pku.edu.cn
School of CS, Peking University
Beijing, China

Wentao Zhang
Zhi Yang
wentao.zhang@pku.edu.cn
yangzhi@pku.edu.cn
School of CS, Peking University
Beijing, China

Ce Zhang
DS3Lab, Systems Group, Department of Computer Science, ETH Zürich
Zürich, Switzerland
ce.zhang@inf.ethz.ch

Bin Cui
School of CS, Peking University
Institute of Computational Social Science, Peking University (Qingdao)
Beijing, China
bin.cui@pku.edu.cn

ABSTRACT

With the extensive applications of machine learning models, automatic hyperparameter optimization (HPO) has become increasingly important. Motivated by the tuning behaviors of human experts, it is intuitive to leverage auxiliary knowledge from past HPO tasks to accelerate the current HPO task. In this paper, we propose TransBO, a novel two-phase transfer learning framework for HPO, which can deal with the complementary nature among source tasks and dynamics during knowledge aggregation issues simultaneously. This framework extracts and aggregates source and target knowledge jointly and adaptively, where the weights can be learned in a principled manner. The extensive experiments, including static and dynamic transfer learning settings and neural architecture search, demonstrate the superiority of TransBO over the state-of-the-arts.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Transfer learning**.

KEYWORDS

hyperparameter optimization, black-box optimization, bayesian optimization, transfer learning

ACM Reference Format:
Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Zhi Yang, Ce Zhang, Bin Cui. 2022. TransBO: Hyperparameter Optimization via Two-Phase Transfer Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539255

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539255

1 INTRODUCTION

Machine learning (ML) models have been extensively applied in many fields such as recommendation, computer vision, financial market analysis, etc [6, 14–18]. However, the performance of ML models heavily depends on the choice of hyperparameter configurations (e.g., learning rate or the number of hidden layers in a deep neural network). As a result, automatically tuning the hyperparameters has attracted lots of interest from both academia and industry [59]. Bayesian optimization (BO) is one of the most prevailing frameworks for automatic hyperparameter optimization (HPO) [4, 20, 48]. The main idea of BO is to use a surrogate model, typically a Gaussian Process (GP) [42], to describe the relationship between a hyperparameter configuration and its performance (e.g., validation error), and then utilize this surrogate to determine the next configuration to evaluate by optimizing an acquisition function that balances exploration and exploitation.

Hyperparameter optimization (HPO) is often a computationally-intensive process as one often needs to choose and evaluate hyperparameter configurations by training and validating the corresponding ML models. However, for ML models that are computationally expensive to train (e.g., deep learning models or models trained on large-scale datasets), vanilla Bayesian optimization (BO) suffers from the low-efficiency issue [9, 31, 33] due to insufficient configuration evaluations within a limited budget.

(Opportunities) Production ML models usually need to be constantly re-tuned as new task / dataset comes or underlying code bases are updated, e.g., in the AutoML applications. The optimal hyperparameters may also change as the data and code change, and so should be frequently re-optimized. Although they may change significantly, the region of good or bad configurations may still share some correlation with those of previous tasks [60], and this provides the opportunities towards a faster hyperparameter search. Therefore, *we can leverage the tuning results (i.e., observations) from*

previous HPO tasks (source tasks) to speed up the current HPO task (target task) via a transfer learning-based framework.

(Challenges) The transfer learning for HPO consists of two key operations: *extracting* source knowledge from previous HPO tasks, and *aggregating* and *transfering* these knowledge to a target domain. To fully unleash the potential of TL, we need to address two main challenges when performing the above operations: 1) *The Complementary Nature among Source Tasks.* Different source tasks are often complementary and thus require us to treat them in a joint and cooperative manner. Ignoring the synergy of multiple source tasks might lead to the loss of auxiliary knowledge. 2) *Dynamics during Knowledge Aggregation.* At the beginning of HPO, the knowledge from the source tasks could bring benefits due to the scarcity of observations on the target task. However, as the tuning process proceeds, we should shift the focus to the target task. Since the target task gets more observations, transferring from source tasks might not be necessary anymore considering the bias and noises in the source tasks (i.e., negative transfer [36]). Existing methods [11, 46, 58] have been focusing on these two challenges. However, none of them considers both simultaneously. This motivates our work, which aims at developing a transfer learning framework that could 1) extract source knowledge in a *cooperative* manner, and 2) transfer the auxiliary knowledge in an *adaptive* way.

In this paper, we propose `TransBO`, a novel two-phase transfer learning framework for automatic HPO that tries to address the above two challenges simultaneously. `TransBO` works under the umbrella of Bayesian optimization and designs a transfer learning (TL) surrogate to guide the HPO process. This framework decouples the process of knowledge transfer into two phases and considers the knowledge extraction and knowledge aggregation separately in each phase (See Figure 1). In Phase one, `TransBO` builds a source surrogate that extracts and combines useful knowledge across multiple source tasks. In Phase two, `TransBO` integrates the source surrogate (in Phase one) and the target surrogate to construct the final surrogate, which we refer to as the transfer learning surrogate. To maximize the generalization of the transfer learning surrogate, we adopt the cross-validation mechanism to learn the transfer learning surrogate in a principled manner. Moreover, instead of combining base surrogates with independent weights, `TransBO` can learn the optimal aggregation weights for base surrogates jointly. To this end, we propose to learn the weights in each phase by solving a constrained optimization problem with a differentiable ranking loss function.

The empirical results of static TL scenarios showcase the stability and effectiveness of `TransBO` compared with state-of-the-art TL methods for HPO. In dynamic TL scenarios that are close to real-world applications, `TransBO` obtains strong performance – the top-2 results on 22.25 out of 30 tuning tasks (Practicality). In addition, when applying `TransBO` to neural architecture search (NAS), it achieves more than 5× speedups than the state-of-the-art NAS approaches (Universality).

2 RELATED WORK

Bayesian optimization (BO) has been successfully applied to hyperparameter optimization (HPO) [5, 29, 30, 32]. For ML models that are computationally expensive to train (e.g., deep learning models or models trained on large datasets), BO methods [4, 20, 48] suffer from the low-efficiency issue due to insufficient configuration evaluations within a limited budget. To speed up HPO of ML algorithms with limited trials, recent BO methods extend the traditional black-box assumption by exploiting cheaper fidelities from the current task [9, 23, 26, 27, 30, 31, 41, 52]. Orthogonal to these methods, we focus on borrowing strength from previously finished tasks to accelerate the HPO of the current task.

Transfer learning (TL) methods for HPO aim to leverage auxiliary knowledge from previous tasks to achieve faster optimization on the target task. One common way is to learn surrogate models from past tuning history and use them to guide the search of hyperparameters. For instance, several methods learn all available information from both source and target tasks in a single surrogate, and make the data comparable through a transfer stacking ensemble [38], a ranking algorithm [1], multi-task GPs [51], a mixed kernel GP [60], the GP noisy model [22], a multi-layer perceptron with Bayesian linear regression heads [39, 49] or replace GP with Bayesian neural networks [50]. SGPR [13] and SMFO [57] utilize the knowledge from all source tasks equally and thus suffer from performance deterioration when the knowledge of source tasks is not applicable to the target task. FMLP [45] uses multi-layer perceptrons as the surrogate model that learns the interaction between hyperparameters and datasets. SCoT [1] and MKL-GP [60] fit a GP-based surrogate on merged observations from both source tasks and target task. To distinguish the varied performance of the same configuration on different tasks, the two methods use the meta-features of datasets to represent the tasks; while the meta-features are often unavailable for broad classes of HPO problems [11]. Due to the high computational complexity of GP ($O(n^3)$), it is difficult for these methods to scale to a large number of source tasks and trials (scalability bottleneck).

To improve scalability, recent methods adopt the ensemble framework to conduct TL for HPO, where they train a base surrogate on each source task and the target task respectively and then combine all base surrogates into an ensemble surrogate with different weights. This framework ignores the two aforementioned issues and uses the *independent* weights. POGPE [46] sets the weights of base surrogates to constants. TST [58] linearly combines the base surrogates with a Nadaraya-Watson kernel weighting by defining a distance metric across tasks; the weights are calculated by using either meta-features (TST-M) or pairwise hyperparameter configuration rankings (TST-R). RGPE [11] uses the probability that the base surrogate has the lowest ranking loss on the target task to estimate the weights. Instead of resorting to heuristics, `TransBO` propose to learn the joint weights in a principled way.

Warm-starting methods [25, 34] select several initial hyperparameter configurations as the start points of search procedures. Salinas et al. [44] deal with the heterogeneous scale between tasks with the Gaussian Copula Process. ABRAC [19] proposes a multi-task BO method with adaptive complexity to prevent over-fitting on scarce target observations. TNP [55] applies the neural process to jointly transfer surrogates, parameters, and initial configurations. Recently, transferring search space has become another way for applying transfer learning in HPO. Wistuba et al. [56] prune the bad regions of search space according to the results from previous tasks. This method suffers from the complexity of obtaining meta-features

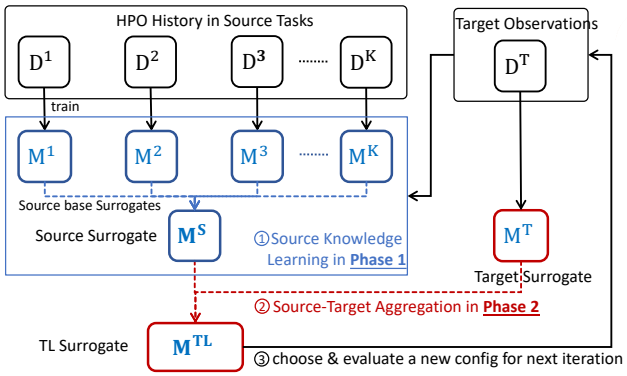


Figure 1: Two-Phase Transfer Learning Framework.

and relies on some other parameters to construct a GP model. On that basis, Perrone et al. [40] propose to utilize previous tasks to design a sub-region of the entire search space for the new task. While sharing some common spirits, these methods are orthogonal and complementary to our surrogate transfer method introduced in this paper.

In addition, our proposed two-phase framework inherits the advantages of the bi-level optimization [2]. While previous methods in the literature focus on different tasks (e.g., evolutionary computation [47]), to the best of our knowledge, TransBO is the first method that adopts the concept of bi-level optimization into hyperparameter transfer learning.

3 BAYESIAN HYPERPARAMETER OPTIMIZATION

The HPO of ML algorithms can be modeled as a black-box optimization problem. The goal is to find $argmin_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$ in hyperparameter space $\mathcal{X}$, where $f(\boldsymbol{x})$ is the ML model's performance metric (e.g., validation error) corresponding to the configuration $\boldsymbol{x}$. Due to the intrinsic randomness of most ML algorithms, we evaluate configuration $\boldsymbol{x}$ and can only get its noisy result $y = f(\boldsymbol{x}) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Bayesian optimization (BO) is a model-based framework for HPO. BO first fits a probabilistic surrogate model $M : p(f|D)$ on the already observed instances $D = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_{n-1}, y_{n-1})\}$. In the n-th iteration, BO iterates the following steps: 1) use surrogate M to select a promising configuration $\boldsymbol{x}_n$ that maximizes the acquisition function $\boldsymbol{x}_n = \arg\max_{\boldsymbol{x}\in\mathcal{X}} a(\boldsymbol{x}; M)$, where the acquisition function is to balance the exploration and exploitation trade-off; 2) evaluate this point to get its performance y_n, and add the new observation $(\boldsymbol{x}_n, y_n)$ to D; 3) refit M on the augmented D. Expected Improvement (EI) [21] is a common acquisition function defined as follows:

$$a(\boldsymbol{x}; M) = \int_{-\infty}^{\infty} \max(y^* - y, 0) p_M(y|\boldsymbol{x}) dy, \tag{1}$$

where M is the surrogate and $y^* = \min\{y_1, ..., y_n\}$. By maximizing this EI function $a(\boldsymbol{x}; M)$ over $\mathcal{X}$, BO methods can find a configuration to evaluate for each iteration.

4 THE PROPOSED METHOD

In this section, we present TransBO, a two-phase transfer learning (TL) framework for HPO. Before diving into the proposed framework, we first introduce the notations and settings for TL. Then we describe TransBO in details and end the section with discussions about its advantages.

Basic Notations and Settings. As illustrated in Figure 1, we denote observations from $K+1$ tasks as $D^1, ..., D^K$ for K source tasks and D^T for the target task. The i-th source task has n_i configuration observations: $D^i = \{(\boldsymbol{x}_j^i, y_j^i)\}_{j=1}^{n_i}$ with $i = 1, 2, ..., K$, which are obtained from previous tuning procedures. For the target task, after completing t iterations (trials), the observations in the target task are: $D^T = \{(\boldsymbol{x}_j^T, y_j^T)\}_{j=1}^{t}$.

Before optimization, we train a base surrogate model for the i-th source task, denoted by M^i. Each base surrogate M^i can be fitted on D^i in advance (offline), and the target surrogate M^T is trained on D^T on the fly. Since the configuration performance ys in each D^i and D^T may have different numerical ranges, we standardize the ys in each task by removing the mean and scaling to unit variance. For a hyperparameter configuration $\boldsymbol{x}_j$, each base surrogate M^i outputs a posterior predictive distribution at $\boldsymbol{x}_j$, that's, $M^i(\boldsymbol{x}_j) \sim \mathcal{N}(\mu_{M^i}(\boldsymbol{x}_j), \sigma^2_{M^i}(\boldsymbol{x}_j))$. For brevity, we denote the mean of this prediction at $\boldsymbol{x}_j$ as $M^i(\boldsymbol{x}_j) = \mu_{M^i}(\boldsymbol{x}_j)$.

4.1 Overview

TransBO aims to build a transfer learning surrogate model M^{TL} on the target task, which outputs a more accurate prediction for each configuration by borrowing strength from the source tasks. The cornerstone of TransBO is to decouple the combination of $K+1$ base surrogates with a novel two-phase framework:

Phase 1. To leverage the complementary nature among source tasks, TransBO first linearly combines all source base surrogates into a single source surrogate with the weights $\mathbf{w}$:

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

In this phase, the useful source knowledge from each source task is extracted and integrated into the source surrogate in a joint and cooperative manner.

Phase 2. To support dynamics-aware knowledge aggregation, TransBO further combines the aggregated source surrogate with the target surrogate M^T via weights $\mathbf{p}$ in an adaptive manner, where M^T is trained on the target observations D^T:

$$M^{TL} = \text{agg}(\{M^S, M^T\}; \mathbf{p}).$$

Such joint and adaptive knowledge transfer in two phases guarantees the efficiency and effectiveness of the final TL surrogate M^{TL} in extracting and integrating the source and target knowledge. To maximize the generalization ability of M^{TL}, the two-phase framework further learns the parameters $\mathbf{w}$ and $\mathbf{p}$ in a principled and automatic manner by solving the constrained optimization problems. In the following, we describe the parameter learning and aggregation method.

4.2 Parameter Learning in Two-Phase Framework

Notice that $\mathbf{w}$ and $\mathbf{p}$ play different roles — $\mathbf{w}$ combines K source base surrogates to best fit the target observations, while $\mathbf{p}$ balances between two surrogates M^S and M^T. The objective of TransBO is to maximize the generalization performance of M^{TL}. To obtain $\mathbf{w}$, we use the target observations D^T to maximize the performance of source surrogate M^S. However, if we learn the parameter $\mathbf{p}$ of M^{TL} on D^T by using the M^S and M^T, where M^S and M^T are trained on D^T directly, the learning process becomes an estimation of in-sample error and can not reflect the generalization of the final surrogate M^{TL}. To address this issue, we adopt the cross-validation mechanism to maximize the generalization ability of M^{TL} when learning $\mathbf{p}$. In the following, we first describe the general procedure to learn a surrogate M^S on given observations D (instead of D^T), and then introduce the method to learn the parameters $\mathbf{w}$ and $\mathbf{p}$, respectively.

General Procedure: Fitting M^S on Given Observations D. Our strategy is to obtain the source surrogate M^S as a weighted combination of the predictions of source base surrogates $\{M^1, ..., M^K\}$:

$$M^S(\boldsymbol{x}) = \sum_{i=1}^{K} w_i M^i(\boldsymbol{x}), \tag{2}$$

where $\sum_i w_i = 1$ and $w_i \in [0, 1]$. Intuitively, the weight w_i reflects the quality of knowledge extracted from the corresponding source tasks. Instead of calculating weights independently, which may ignore the complementary nature among source tasks, we propose to combine source base surrogates M^is in a joint and supervised manner, which reveals their cooperative contributions to M^S.

To derive M^S in a principled way, we use a differentiable pairwise ranking loss function to measure the fitting error between the prediction of M^S and the available observations D. In HPO, ranking loss is more appropriate than mean square error — the actual values of predictions are not the most important, and we care more about the partial orders over the hyperparameter space, e.g., the location of the optimal configuration. This ranking loss function is defined as follows:

$$\begin{aligned} &\mathbb{L}(\mathbf{w}, M^S; D) = \frac{1}{n^2} \sum_{j=1}^{n} \sum_{k=1, y_j < y_k}^{n} \phi(M^S(\boldsymbol{x}_k) - M^S(\boldsymbol{x}_j)), \\ &\phi(z) = log(1 + e^{-z}), \end{aligned} \tag{3}$$

where n is the number of observations in D, y is the observed performance of configuration $\boldsymbol{x}$ in D, and the prediction of $M^S(\boldsymbol{x}_j)$ at configuration $\boldsymbol{x}_j$ is obtained by linearly combining the predictive mean of M^i with a weight w_i, that's, $M^S(\boldsymbol{x}_j) = \sum_i w_i M^i(\boldsymbol{x}_j)$.

We further turn the learning of source surrogate M^S, i.e., the learning of $\mathbf{w}$, into the following constrained optimization problem:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \mathbb{L}(\mathbf{w}, M^S; D) \\ \text{s.t.} \quad & \mathbf{1}^\top \mathbf{w} = 1, \mathbf{w} \geq \mathbf{0}, \end{aligned} \tag{4}$$

where the objective is the ranking loss of M^S on D. This optimization objective is continuously differentiable, and concretely, it is twice continuously differentiable. So we can have the first derivative of the objective $\mathbb{L}$ as follows:

$$\begin{aligned} &\frac{\partial \mathbb{L}}{\partial \mathbf{w}} = \sum_{(j,k) \in \mathbb{P}} \frac{neg\_ez}{1 + neg\_ez} * (A_{[j]} - A_{[k]}), \\ &neg\_ez = e^{(A_{[j]}\mathbf{w} - A_{[k]}\mathbf{w})}, \end{aligned} \tag{5}$$

where $\mathbb{P}$ consists of pairs (j, k) satisfying $y_j < y_k$, A is the matrix formed by putting the predictions of $M^{1:K}$s together where the element at the i-th row and j-th column is $M^i(\boldsymbol{x}_j)$, and $A_{[j]}$ is the row vector in the j-th row of matrix A. Furthermore, this optimization problem can be solved efficiently by applying many existing sequential quadratic programming (SQP) solvers [28].

Learning Parameter w. As stated previously, to maximize the (generalization) performance of M^S, we propose to learn the parameter $\mathbf{w}$ by fitting M^S on the whole observations D^T. In this way, the useful source knowledge from multiple source tasks can be fully extracted and integrated in a joint manner. Therefore, the parameters $\mathbf{w}$ can be obtained by calling the general procedure, i.e., solving the problem 4, where the available observations D are set to D^T.

Learning Parameter p. To reflect the generalization in M^{TL}, the parameter $\mathbf{p}$ is learned with the cross-validation mechanism. We first split D^T into N_{cv} partitions: D_1^T, ..., $D_{N_{cv}}^T$ with $N_{cv} = 5$. For each partition $i \in [1 : N_{cv}]$, we first fit a partial surrogate M_{-i}^S on the observations D_{-i}^T with observations in the i-th partition removed from D^T, and the surrogate M_{-i}^S is learned on D_{-i}^T using the general procedure; in addition, we also fit a partial surrogate model M_{-i}^T on D_{-i}^T directly. Then we combine the surrogates M_{-i}^S and M_{-i}^T linearly to obtain a M_{-i}^{TL}:

$$M_{-i}^{TL} = p^S M_{-i}^S + p^T M_{-i}^T, \tag{6}$$

where $\mathbf{p} = [p^S, p^T]$. Therefore, we can obtain N_{cv} partial surrogates M_{-i}^S and M_{-i}^T with $i \in [1 : N_{cv}]$. Based on the differentiable pairwise ranking loss function in Eq. 3, the loss of M_{-i}^{TL} on D^T is defined as:

$$\begin{aligned} &\mathbb{L}_{cv}(\mathbf{p}, M_{-i}^{TL}; D^T) = \frac{1}{n^2} \sum_{j=1}^{n} \sum_{k=1, y_j^T < y_k^T, k \in D_i^T}^{n} \phi(z), \\ &\phi(z) = log(1 + e^{-z}), z = M_{-i}^{TL}(\boldsymbol{x}_k) - M_{-H(j)}^{TL}(\boldsymbol{x}_j) \end{aligned} \tag{7}$$

where n is the number of observations in D^T, y^T is the observed performance of configuration $\boldsymbol{x}^T$ in D^T, $H(j)$ indicates the partition id that configuration $\boldsymbol{x}_j$ belongs to, and the prediction of M_{-i}^{TL} at configuration $\boldsymbol{x}_k$ is obtained by linearly combining the predictive mean of M_{-i}^S and M_{-i}^T with weight $\mathbf{p}$, that's, $M_{-i}^{TL}(\boldsymbol{x}_k) = p^S M_{-i}^S(\boldsymbol{x}_k) + p^T M_{-i}^T(\boldsymbol{x}_k)$. So the parameter $\mathbf{p}$ can be learned by solving a similar constrained optimization problem on D^T:

$$\begin{aligned} \underset{\mathbf{p}}{\text{minimize}} \quad & \sum_{i=1}^{N_{cv}} \mathbb{L}_{cv}(\mathbf{p}, M_{-i}^{TL}; D^T) \\ \text{s.t.} \quad & \mathbf{1}^\top \mathbf{p} = 1, \mathbf{p} \geq \mathbf{0}. \end{aligned} \tag{8}$$

Following the solution introduced in problem 4, the above optimization problem can be solved efficiently.

Final TL Surrogate. After $\mathbf{w}$ and $\mathbf{p}$ are obtained, as illustrated in Figure 1, we first combine the source base surrogates into the source

Algorithm 1 The TransBO Framework.

Input: maximum number of trials N^T, observations from K source tasks: $D^{1:K}$, and config. space $\mathcal{X}$.

1: **for** $i \in \{1, 2, ..., N^T\}$ **do**
2: Calculate the weight $\mathbf{w}_i$ in M^S by solving (4).
3: Calculate the weight $\mathbf{p}_i$ in M^{TL} by solving (8).
4: Employ non-decreasing prior on p^T: $p_i^T = \max(p_i^T, p_{i-1}^T)$.
5: Build M^S, M^{TL} with weights $\mathbf{w}_i$ and $\mathbf{p}_i$, respectively.
6: Sample a large number of configurations randomly from $\mathcal{X}$, compute their acquisition values according to the EI criterion in Eq.1, where $M = M^{TL}$, and choose the configuration $\boldsymbol{x}_i = \text{argmax}_{x \in \mathcal{X}} ei(x, M^{TL})$.
7: Evaluate $\boldsymbol{x}_i$ and get its performance y_i, augment observations D^T with $(\boldsymbol{x}_i, y_i)$ and refit M^T on the augmented D^T.
8: **end for**
9: **return** the best configuration in D^T.

surrogate M^S with $\mathbf{w}$ (the Phase 1), and then integrate M^S and M^T with $\mathbf{p}$ to obtain the final TL surrogate M^{TL} (the Phase 2). To ensure the surrogate M^{TL} still works in the BO framework, it is required to be a GP. How to obtain the unified posterior predictive mean and variance from multiple GPs (base surrogates) is still an open problem. As suggested by [11], the linear combination of multiple base surrogates works well in practice. Therefore, we aggregate the base surrogates with linear combination. That's, suppose there are N_B GP-based surrogates, and each base surrogate M^b has a weight w_b with $b = 1, ..., N_B$, the combined prediction under the linear combination technique is give by: $\mu_C(\boldsymbol{x}) = \sum_b w_b \mu_b(\boldsymbol{x})$ and $\sigma_C^2(\boldsymbol{x}) = \sum_b w_b^2 \sigma_b^2(\boldsymbol{x})$.

Algorithm Summary At initialization, we set the weight of each source surrogate in $\mathbf{w}$ to $1/K$, and $\mathbf{p} = [1, 0]$ when the number of trials is insufficient for cross-validation. Algorithm 1 illustrates the pseudo code of TransBO. In the i-th iteration, we first learn the weights $\mathbf{p}_i$ and $\mathbf{w}_i$ by solving two optimization problems (Lines 2-3). Since we have the prior: as the HPO process of the target task proceeds, the target surrogate owns more and more knowledge about the objective function of the target task, therefore the weight of M^T should increase gradually. To this end, we employ a *max* operator, which enforces that the update of p^T should be non-decreasing (Line 4). Next, by using linear combination, we build the source surrogate M^S with weight $\mathbf{w}_i$, and then construct the final TL surrogate M^{TL} with $\mathbf{p}_i$ (Line 5). Finally, TransBO utilizes M^{TL} to choose a promising configuration to evaluate, and refit the target surrogate on the augmented observation (the BO framework, Lines 6-7).

4.3 Discussion: Advantages of TransBO

To our knowledge, TransBO is the first method that conducts transfer learning for HPO in a supervised manner, instead of resorting to some heuristics. In addition, this method owns the following desirable properties simultaneously. 1) **Practicality.** A practical HPO method should be insensitive to its hyperparameters, and do not depend on meta-features. The goal of HPO is to optimize the ML hyperparameters automatically while having extra (or sensitive) hyperparameters itself actually violates its principle. In addition, many datasets, including image and text data, lack appropriate meta-features to represent the dataset [11, 45, 58]. The construction of TL surrogate in TransBO is insensitive to its hyperparameters and does not require meta-features. 2) **Universality.** The 1st property enable TransBO to be a general transfer learning framework for Black-box optimizations, e.g., experimental design [12], neural architecture search [8], etc; we include an experiment to evaluate TransBO on the NAS task in the section of experiment). 3) **Scalability.** Compared with the methods that combine k source tasks with n trials into a single surrogate ($O(k^3n^3)$), TransBO has a much lower complexity $O(kn^3)$, which means that TransBO could scale to a large number of tasks and trials easily. 4) **Theoretical Discussion.** TransBO also provides theoretical discussions about preventing the performance deterioration (negative transfer). Base on cross-validation and the non-decreasing constraint, *the performance of* TransBO*, given sufficient trials, will be no worse than the method without transfer learning*, while the other methods cannot have this (See Appendix A.4 for more details).

5 EXPERIMENTS AND RESULTS

In this section, we evaluate TransBO from three perspectives: 1) stability and effectiveness on static TL tasks, 2) practicality on real-world dynamic TL tasks, and 3) universality when conducting neural architecture search.

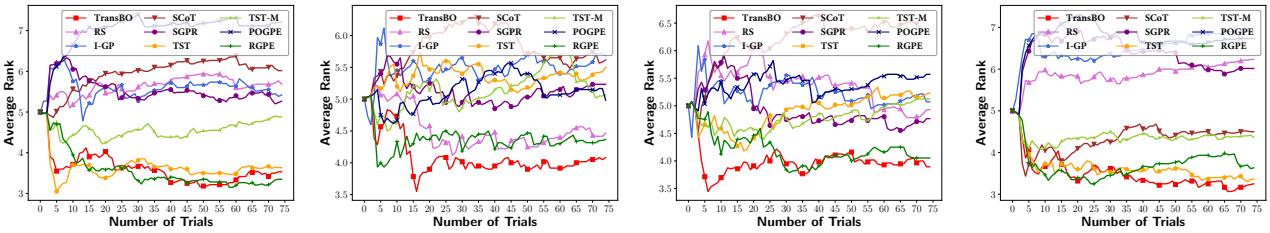
5.1 Experimental Setup

Baselines. We compare TransBO with eight baselines – two non-transfer methods: (1) Random search [3], (2) I-GP: independent Gaussian process-based surrogate fitted on the target task without using any source data, (3) SCoT [1]: it models the relationship between datasets and hyperparamter performance by training a single surrogate on the scaled and merged observations from both source tasks and the target task, (4) SGPR: the core TL algorithm used in the well-known service — Google Vizier [13], and four ensemble based TL methods: (5) POGPE [46], (6) TST [58], (7) TST-M: a variant of TST using dataset meta-features [58], and (8) RGPE [11].

Benchmark on 30 OpenML Datasets. To evaluate the performance of TransBO, we create and publish a large-scale benchmark. Four ML algorithms, including Random Forest, Extra Trees, Adaboost and LightGBM [24], are tuned on 30 real-world datasets (tasks) from OpenML repository [53]. The design of hyperparameter space and meta-feature for each dataset is adopted from the implementation in Auto-Sklearn [10]. For each ML algorithm on each dataset, we sample 20k configurations from the hyperparameter space randomly and store the corresponding evaluation results. It takes more than 200k CPU hours to collect these evaluation results. Note that, for reproducibility, we provide more details about this benchmark, including the datasets, the hyperparameter space of ML algorithms, etc., in Appendix A.1.

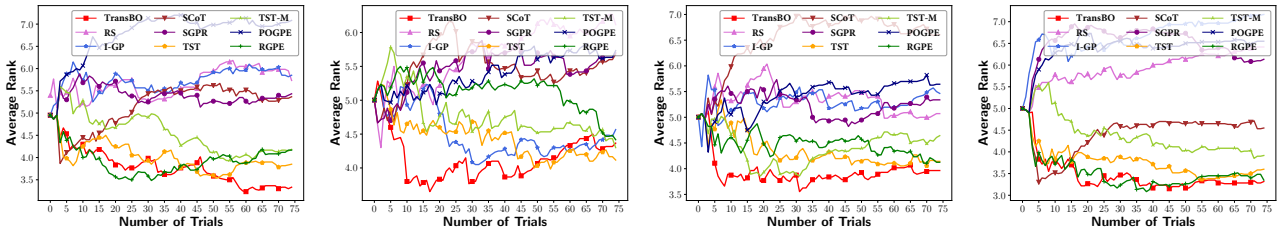
AutoML HPO Tasks. To evaluate the performance of each method, the experiments are performed in a leave-one-out fashion. Each method optimizes the hyperparameters of a specific task over 20k configurations while treating the remaining tasks as the source tasks. In each source task, only N_S instances (here $N_S = 50$) are used to extract knowledge from this task in order to test the efficiency of TL [11, 58].

We include the following three kinds of tasks:



(a) Random Forest (b) LightGBM (c) Adaboost (d) Extra Trees

Figure 2: Static TL results for four algorithms with $N_{task} = 29$ source tasks.



(a) Random Forest (b) LightGBM (c) Adaboost (d) Extra Trees

Figure 3: Static TL results for four algorithms with $N_{task} = 5$ source tasks

(a) **Static TL Setting.** This experiment is performed in a leave-one-out fashion, i.e., we optimize the hyperparameters of the target task while treating the remaining tasks as the source tasks.

(b) **Dynamic TL Setting.** It simulates the real-world HPO scenarios, in which 30 tasks (datasets) arrive sequentially; when the i-th task appears, the former $i-1$ tasks are treated as the source tasks.

(c) **Neural Architecture Search (NAS).** It transfers tuning knowledge from conducting NAS on CIFAR-10 and CIFAR-100 to accelerate NAS on ImageNet16-120 based on NAS-Bench201 [7].

In addition, following [11], all the compared methods are initialized with three randomly selected configurations, after which they proceed sequentially with a total of N_T evaluations (trials). To avoid the effect of randomness, each method is repeated 30 times, and the averaged performance metrics are reported.

Evaluation Metric. Comparing each method in terms of classification error is questionable because the classification error is not commensurable across datasets. Following the previous works [1, 11, 58], we adopt the metrics as follows:

Average Rank. For each target task, we rank all compared methods based on the performance of the best configuration they have found so far. Furthermore, ties are being solved by giving the average rank. For example, if one method observes the lowest validation error of 0.2, another two methods find 0.3, and the last method finds only 0.45, we would rank the methods with 1, $\frac{2+3}{2}$, $\frac{2+3}{2}$, 4.

Average Distance to Minimum. The average distance to the global minimum after t trials is defined as:

$$ADTM(\mathcal{X}_t) = \frac{1}{K} \sum_{i \in [1:K]} \frac{min_{\boldsymbol{x} \in \mathcal{X}_t} y^i_{\boldsymbol{x}} - y^i_{min}}{y^i_{max} - y^i_{min}}, \tag{9}$$

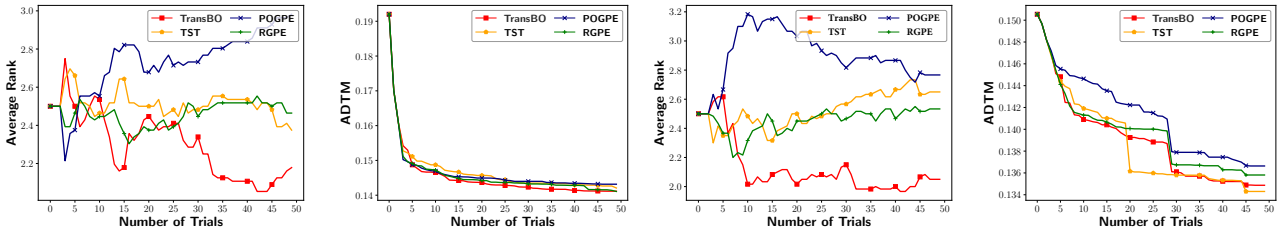
where y^i_{min} and y^i_{max} are the best and worst performance value on the i-th task, K is the number of tasks, i.e., $K = 30$, $y^i_{\boldsymbol{x}}$ corresponds to the performance of configuration $\boldsymbol{x}$ in the i-th task, and $\mathcal{X}_t$ is the set of hyperparameter configurations that have been evaluated in the previous t trials. The relative distances over all considered tasks are averaged to obtain the final ADTM value.

Implementations & Parameters. `TransBO` implements the Gaussian process using SMAC3[1] [20, 35], which can support a complex hyperparameter space, including numerical, categorical, and conditional hyperparameters, and the kernel hyperparameters in GP are inferred by maximizing the marginal likelihood. The two optimization problems in `TransBO` are solved by using SQP methods provided in SciPy [2] [54]. In the BO module, the popular EI acquisition function is used. As for the parameters in each baseline, the bandwidth ρ in TST [58] is set to 0.3 for all experiments; in RGPE, we sample 100 times ($S = 100$) to calculate the weight for each base surrogate; in SGPR [13], the parameter α, which determines the relative importance of standard deviations of past tasks and the current task, is set to 0.95 (Check Appendix B for reproduction details).

5.2 Comprehensive Experiments in Two TL Settings

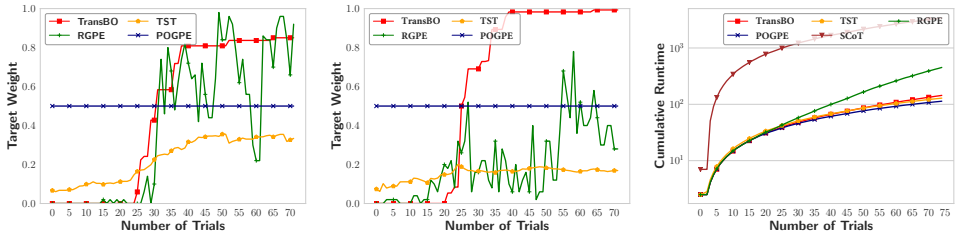
Static TL Setting. To demonstrate the efficiency and effectiveness of transfer learning in the static scenario, we compare `TransBO` with the baselines on four benchmarks (i.e., Random Forest, LightGBM, Adaboost, and Extra Trees). Concretely, each task is selected as the

[1] https://github.com/automl/SMAC3
[2] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html



(a) Adaboost (Average Rank) (b) Adaboost (ADTM) (c) LightGBM (Average Rank) (d) LightGBM (ADTM)

Figure 4: Results on source knowledge learning.



(a) Random Forest (*cpu_small*) (b) Adaboost (*hypothyroid(2)*) (c) Scalability on Random Forest

Figure 5: Target weight and scalability analysis.

target task in turn, and the remaining tasks are the source tasks; then we can measure the performance of each baseline based on the results when tuning the hyperparameters of the target task. Furthermore, we use 29 and 5 source tasks respectively to evaluate the ability of each method when given a different amount of source knowledge in terms of the number of source tasks N_{task}. Note that, for each target task, the maximum number of trials is 75. Figure 2 and Figure 3 show the experiment results on four benchmarks with 29 and 5 source tasks respectively, using average rank; more results on ADTM can be found in Appendix A.3.

First, we can observe that the average rank of TransBO in Figure 2 and Figure 3 decreases sharply in the initial 20 trials. Compared with other TL methods, it shows that TransBO can extract and utilize the auxiliary source knowledge efficiently and effectively. Remarkably, TransBO exhibits a strong stability from two perspectives: 1) TransBO is stable on different benchmarks; and 2) it still performs well when given a different number of source tasks, e.g., in Figure 2 $N_{task} = 29$, and $N_{task} = 5$ in Figure 3. RGPE is one of the most competitive baselines, and we take it as an example. RGPE achieves comparable or similar performance with TransBO in Figure 2(b) and Figure 2(c) where $N_{task} = 29$. However, in Figure 3(b) and Figure 3(c) RGPE exhibits a larger fluctuation over the trials compared with TransBO when $N_{task} = 5$. Unlike the baselines, TransBO extracts the source knowledge in a principled way, and the empirical results show it performs well in most circumstances, thus demonstrating its superior efficiency and effectiveness.

Dynamic TL Setting. To simulate the real-world transfer learning scenario, we perform the dynamic experiment on different benchmarks. In this experiment, 30 tasks arrive sequentially; when the i-th task arrives, the previous i-1 tasks are used as the source tasks.

Table 1: Dynamic TL results for Tuning four ML algorithms.

Method	Adaboost		Random Forest		Extra Trees		LightGBM	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd
POGPE	0	2	0	1	0	2	1	2
TST	8	12	9	9	7	12	10	9
RGPE	8	5	6	14	10	9	9	10
TransBO	**14**	11	**15**	6	**14**	7	**12**	10

The maximum number of trials for each task is 50, and we compare TransBO with TST, RGPE, and POGPE based on the best-observed performance on each task. Table 1 reports the number of tasks on which each TL method gets the highest and second-highest performance. Note that the sum of each column may be more than 30 since some of the TL methods are tied for first or second place.

As shown in Table 1, TransBO achieves the largest number of top1 and top2 online performance among the compared methods. Take Adaboost as an example, TransBO gets 25 top2 results among 30 tasks, while this number is 13 for RGPE. RGPE gets a similar performance with TST on Lightgbm and Extra Trees, but its performance decreases on Adaboost. Thus, RGPE is not stable in this scenario. Compared with the baselines, TransBO could achieve more stable and satisfactory performance in the dynamic setting.

5.3 Applying TransBO to NAS

To investigate the universality of TransBO in conducting Neural Architecture Search (NAS), here we use TransBO to extract and integrate the optimization knowledge from NAS tasks on CIFAR-10 and CIFAR-100 (with 50 trials each) to accelerate the NAS task on

ImageNet with NAS-Bench201 [7]. From Figure 6, we have that TransBO could achieve more than 5x speedups over the state-of-the-art NAS methods – Bayesian Optimization (BO) and Regularized Evolution Algorithm (REA) [43]. Therefore, TransBO can also be applied to the NAS tasks.

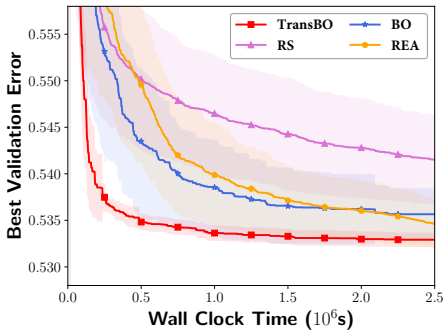


Figure 6: Results on optimizing NAS on NASBench201.

5.4 Ablation Studies

Source Knowledge Learning. This experiment is designed to evaluate the performance of source surrogate M^S learned in Phase 1. M^S corresponds to the source knowledge extracted from the source tasks. In this setting, the source surrogate is used to guide the optimization of hyperparameters instead of the final TL surrogate M^{TL}. The quality of source knowledge learned by each TL method thus can be measured by the performance of M^S. Figure 4 shows the results of TransBO and three one-phase framework based methods: POGPE, TST, and RGPE on two benchmarks — Adaboost and LightGBM. We can observe that the proposed TransBO outperforms the other three baselines on both two metrics: average rank and ADTM. According to some heuristics, these baselines calculate the weights in M^S independently. Instead, by solving the constrained optimization problem, TransBO can learn the optimal weights in M^s in a joint and principled manner. More results on the other two benchmarks can be found in Appendix A.3.

Target Weight Analysis. Here we compare the target weight obtained in POGPE, RGPE, TST, and TransBO. Figure 5(a) and 5(b) illustrate the trend of target weight on two benchmarks: Random Forest and Adaboost. The target weight in POGPE is fixed to a constant - 0.5, regardless of the increasing number of trials; TST's remains low even when the target observations are sufficient; RGPE's shows a trend of fluctuation because the sampling-based ranking loss is not stable. TransBO's keeps increasing with the number of trials, which matches the intuition that the importance of the target surrogate should be low when target observations are insufficient and gradually increase as target observations grow.

Scalability Analysis. In the static TL setting, we include different number of source tasks when conducting transfer learning (See Figures 2 and 3, where N_{task} = 5 and N_{task} = 29); the stable and effective results show the scalability in terms of the number of source tasks. We further investigate the optimization overhead of suggesting a new configuration, and measure the runtime of the baselines - POGPE, RGPE, TST, SCoT, and TransBO on Random Forest with 75 trials. To investigate the scalability of TransBO, we measure the runtime of the competitive TL methods: POGPE, RGPE, TST, SCoT, and TransBO. Each method is tested on Random Forest with 75 trials, and we repeat each method 20 times. Figure 5(c) shows the experiment results, where the y-axis is the mean cumulative runtime in seconds on a log scale. We do not take the evaluation time of each configuration into account, and only compare the optimization overhead of suggesting a new configuration. ScoT's runtime increases rapidly among the compared methods as it has the $O(k^3n^3)$ complexity. Since both the two-phase and one-phase framework-based methods own the $O(n^3)$ complexity, it takes nearly the same optimization overhead for TST, POGPE, and TransBO to suggest a configuration in the first 75 trials. Although RGPE also has the $O(n^3)$ complexity, it depends on a sampling strategy to compute the surrogate weight, which introduces extra overhead to configuration suggestion. Instead, TransBO exhibits a similar scalability result like POGPE, which incorporates no optimization overhead due to the constant weights. This shows that TransBO scales well in both the number of trials and tasks.

6 CONCLUSION

In this paper, we introduced TransBO, a novel two-phase transfer learning (TL) method for hyperparameter optimization (HPO), which can leverage the auxiliary knowledge from previous tasks to accelerate the HPO process of the current task effectively. This framework can extract and aggregate the source and target knowledge jointly and adaptively. In addition, we published a large-scale TL benchmark for HPO with up to 1.8 million model evaluations; the extensive experiments, including static and dynamic transfer learning settings and neural architecture search, demonstrate the superiority of TransBO over the state-of-the-art methods.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No.61832001), Beijing Academy of Artificial Intelligence (BAAI), PKU-Tencent Joint Research Lab. Bin Cui is the corresponding author.

REFERENCES

[1] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *ICML*. 199–207.
[2] Kristin P Bennett, Gautam Kunapuli, Jing Hu, and Jong-Shi Pang. 2008. Bilevel optimization and machine learning. In *IEEE World Congress on Computational Intelligence*. Springer, 25–47.
[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.
[4] 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.
[5] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. 2021. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847* (2021).
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[7] Xuanyi Dong and Yi Yang. 2019. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *International Conference on Learning Representations*.
[8] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. 2020. BRP-NAS: Prediction-based NAS using GCNs. *Advances*

in Neural Information Processing Systems 33 (2020).
[9] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale.. In *ICML*. 1436–1445.
[10] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*. 2962–2970.
[11] 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*.
[12] Adam Foster, Martin Jankowiak, Elias Bingham, Paul Horsfall, Yee Whye Teh, Thomas Rainforth, and Noah Goodman. 2019. Variational Bayesian optimal experimental design. *Advances in Neural Information Processing Systems* 32 (2019).
[13] 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 International Conference on Knowledge Discovery and Data Mining*. ACM, 1487–1495.
[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.
[17] Bruno Miranda Henrique, Vinicius Amorim Sobreiro, and Herbert Kimura. 2019. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications* 124 (2019), 226–251.
[18] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.
[19] Samuel Horváth, Aaron Klein, Peter Richtárik, and Cédric Archambeau. 2021. Hyperparameter transfer learning with adaptive complexity. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1378–1386.
[20] 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.
[21] Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13, 4 (1998), 455–492.
[22] Tinu Theckel Joy, Santu Rana, Sunil Kumar Gupta, and Svetha Venkatesh. 2016. Flexible transfer learning framework for Bayesian optimisation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 102–114.
[23] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. 2017. Multi-fidelity bayesian optimisation with continuous approximations. In *ICML*. PMLR, 1799–1808.
[24] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).
[25] Jungtaek Kim, Saehoon Kim, and Seungjin Choi. 2017. Learning to Transfer Initializations for Bayesian Hyperparameter Optimization. *ArXiv* abs/1710.06219 (2017).
[26] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, Frank Hutter, et al. 2017. Fast Bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics* 11, 2 (2017), 4945–4968.
[27] Aaron Klein, S. Falkner, Jost Tobias Springenberg, and F. Hutter. 2017. Learning Curve Prediction with Bayesian Neural Networks. In *ICLR*.
[28] Dieter Kraft. 1994. Algorithm 733: TOMP–Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)* 20, 3 (1994), 262–281.
[29] Yang Li, Jiawei Jiang, Jinyang Gao, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Efficient automatic CASH via rising bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4763–4771.
[30] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Jixiang Li, Ji Liu, Ce Zhang, and Bin Cui. 2022. Hyper-Tune: Towards Efficient Hyper-parameter Tuning at Scale. *Proceedings of the VLDB Endowment* 15 (2022).
[31] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2021. MFES-HB: Efficient Hyperband with Multi-Fidelity Quality Measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. AAAI Press, 8491–8500.
[32] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, et al. 2021. Openbox: A generalized black-box optimization service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3209–3219.
[33] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, et al. 2021. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *Proceedings of the VLDB Endowment* 14 (2021).
[34] Marius Lindauer and Frank Hutter. 2018. Warmstarting of model-based algorithm configuration. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
[35] Marius Thomas Lindauer, Katharina Eggensperger, Matthias Feurer, Andr'e Biedenkapp, Difan Deng, Caroline Benjamins, René Sass, and Frank Hutter. 2021. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *ArXiv* abs/2109.09831 (2021).
[36] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.
[37] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.
[38] David Pardoe and Peter Stone. 2010. Boosting for regression transfer. In *ICML*. 863–870.
[39] Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cédric Archambeau. 2018. Scalable hyperparameter transfer learning. *Advances in neural information processing systems* 31 (2018).
[40] Valerio Perrone, Huibin Shen, Matthias W Seeger, Cedric Archambeau, and Rodolphe Jenatton. 2019. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. *Advances in Neural Information Processing Systems* 32 (2019).
[41] Matthias Poloczek, Jialei Wang, and Peter Frazier. 2017. Multi-information source optimization. In *Advances in Neural Information Processing Systems*. 4288–4298.
[42] Carl Edward Rasmussen. 2004. Gaussian processes in machine learning. In *Advanced lectures on machine learning*. Springer, 63–71.
[43] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4780–4789.
[44] David Salinas, Huibin Shen, and Valerio Perrone. 2020. A quantile-based approach for hyperparameter transfer learning. In *ICML*. PMLR, 8438–8448.
[45] Nicolas Schilling, Martin Wistuba, Lucas Drumond, and Lars Schmidt-Thieme. 2015. Hyperparameter optimization with factorized multilayer perceptrons. In *ECML PKDD*. 87–103.
[46] Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. 2016. Scalable hyperparameter optimization with products of gaussian process experts. In *ECML PKDD*. Springer, 33–48.
[47] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. 2017. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 276–295.
[48] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
[49] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. 2015. Scalable bayesian optimization using deep neural networks. In *ICML*. PMLR, 2171–2180.
[50] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. 2016. Bayesian optimization with robust Bayesian neural networks. *Advances in neural information processing systems* 29 (2016).
[51] Kevin Swersky, Jasper Snoek, and Ryan P Adams. 2013. Multi-task bayesian optimization. *Advances in neural information processing systems* 26 (2013).
[52] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. 2014. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896* (2014).
[53] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.
[54] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.
[55] Ying Wei, Peilin Zhao, and Junzhou Huang. 2021. Meta-learning Hyperparameter Performance Prediction with Neural Processes. In *ICML*. PMLR, 11058–11067.
[56] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *ECML PKDD*. Springer, 104–119.
[57] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Sequential model-free hyperparameter tuning. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 1033–1038.
[58] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2016. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *ECML PKDD*. 199–214.
[59] Quanming Yao, Mengshuo Wang, H. Escalante, I. Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. *ArXiv* abs/1810.13306 (2018).
[60] Dani Yogatama and Gideon Mann. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial Intelligence and Statistics*. 1077–1085.

A APPENDIX

A.1 The Details of Benchmark

As described in Section 5, we create a benchmark to evaluate the performance of TL methods. We choose four ML algorithms that are widely used in data analysis, including Random Forest, Extra Trees, Adaboost and Lightgbm. The implementation of each algorithm and the design of their hyperparameter space follows Auto-sklearn. For each algorithm, the range and default value of each hyperparameter are illustrated in Tables 2, 3 and 4. To collect sufficient source HPO data for transfer learning, we select 30 real-world datasets from OpenML repository, and evaluate the validation performance (i.e., the balanced accuracy) of 20k configurations for each benchmark, which are randomly sampled from the hyperparameter space. The datasets used in our benchmarks are of medium size, whose number of rows ranges from 2000 to 8192. For more details, see Table 5. The total number of model evaluations (observations) in our benchmarks reaches 1.8 million and it takes more than 100k CPU hours to evaluate all the configurations. For reproduction purposes, we also upload the benchmark data (e.g., evaluation results and the corresponding scripts) along with this submission. The benchmark data (with size – 477.7Mb); due to the space limit (maximum 20Mb) on CMT3, we only upload a small subset of benchmark on one algorithm — LightGBM. After the review process, we will make the complete benchmark publicly available (e.g., on Google Drive).

Hyperparameter	Range	Default
n_estimators	[50, 500]	50
learning_rate (log)	[0.01, 2]	0.1
algorithm	{SAMME.R, SAMME}	SAMME.R
max_depth	[2, 8]	3

Table 2: Hyperparameters of Adaboost.

Hyperparameter	Range	Default
criterion	{gini, entropy}	gini
max_features	[0, 1]	0.5
min_sample_split	[2, 20]	2
min_sample_leaf	[1, 20]	1
bootstrap	{True, False}	True

Table 3: Hyperparameters of Random Forest and Extra Trees.

Hyperparameter	Range	Default
n_estimators	[100, 1000]	500
num_leaves	[31, 2047]	127
learning_rate (log)	[0.001, 0.3]	0.1
min_child_samples	[5, 30]	20
subsample	[0.7, 1]	1
colsample_bytree	[0.7, 1]	1

Table 4: Hyperparameters of LightGBM.

Name	#Rows	#Columns	#Categories
balloon	2001	1	2
kc1	2109	21	2
quake	2178	3	2
segment	2310	19	7
madelon	2600	500	2
space_ga	3107	6	2
splice	3190	60	3
kr-vs-kp	3196	36	2
sick	3772	29	2
hypothyroid(1)	3772	29	4
hypothyroid(2)	3772	29	2
pollen	3848	5	2
analcatdata_supreme	4052	7	2
abalone	4177	8	26
spambase	4600	57	2
winequality_white	4898	11	7
waveform-5000(1)	5000	40	3
waveform-5000(2)	5000	40	2
page-blocks(1)	5473	10	5
page-blocks(2)	5473	10	2
optdigits	5610	64	10
satimage	6430	36	6
wind	6574	14	2
musk	6598	167	2
delta_ailerons	7129	5	2
mushroom	8124	22	2
puma8NH	8192	8	2
cpu_small	8192	12	2
cpu_act	8192	21	2
bank32nh	8192	32	2

Table 5: Details of 30 datasets used in the benchmarks.

A.2 Feasibility of Transfer Learning

To verify the feasibility of transfer learning in the setting of HPO, we conduct an HPO experiment on two datasets — quake and hypothyroid(2). We tune the learning rate and n_estimators of Adaboost while fixing the other hyperparameters, and then evaluate the validation performance (the balanced accuracy) of each configuration. Figure 9 shows the performance on 2500 Adaboost configurations, where deeper color means better performance.

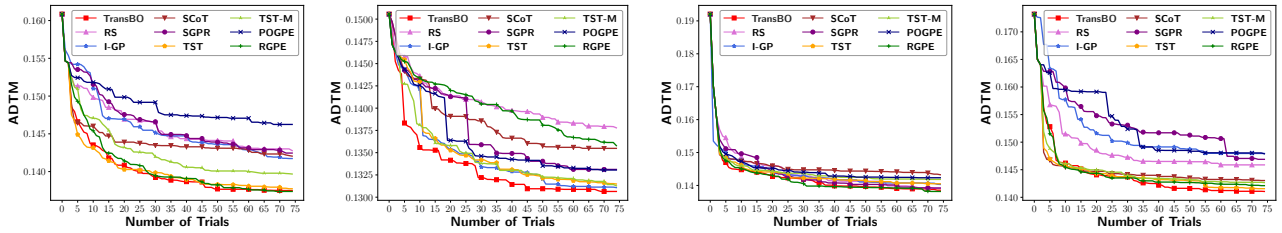
It is quite clear that the optimal configuration differs on the two datasets (tasks), which means re-optimization is essential for HPO. However, the performance distribution is somehow similar on the two datasets. For example, they both perform badly in the lower-right region and perform well in the upper region. Based on this observation, it is natural to accelerate the re-optimization process with the auxiliary knowledge acquired from the previous tasks.

A.3 More Experiment Results

In this section, we provide more experiment results besides those in Section 5.

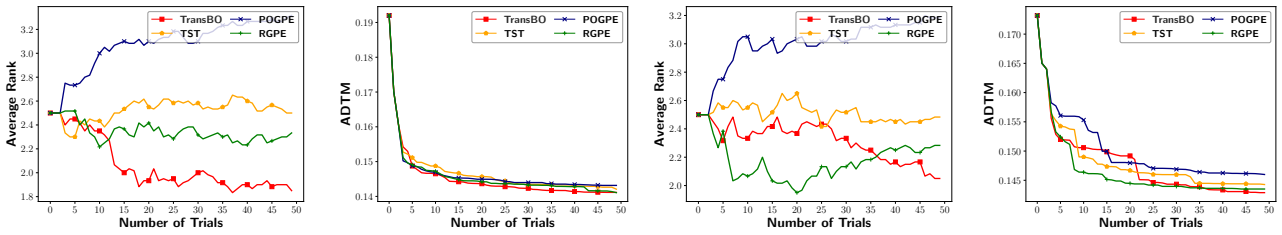
Static Experiments Figure 7 shows the results of all considered methods on the four benchmarks, where the metric is ADTM. We can observe that the proposed `TransBO` exhibits strong stability, and performs well across benchmarks.

Source Knowledge Learning The additional results on Random Forest and Extra Trees are illustrated in Figure 8. Similar to the



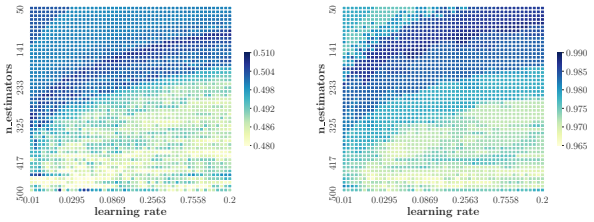
(a) Random Forest (ADTM) (b) LightGBM (ADTM) (c) Adaboost (ADTM) (d) Extra Trees (ADTM)

Figure 7: Static results on four benchmarks with 75 trials.



(a) Random Forest (Average Rank) (b) Random Forest (ADTM) (c) Extra Trees (Average Rank) (d) Extra Trees (ADTM)

Figure 8: Results on source knowledge learning.



(a) quake (b) hypothyroid(2)

Figure 9: Performance of 2500 Adaboost configurations on two tasks, in which each hyperparameter has 50 settings.

findings in Section 5.4, our method - TransBO shows excellent ability in extracting and integrating the source knowledge from previous tasks.

A.4 Convergence Discussion about TransBO

In TransBO, when sufficient trials on the target task are obtained, the weight of target surrogate p^T will approach 1 as the HPO proceeds. Based on the mechanism we adopted in TransBO— cross-validation, we can observe that p_i^T in the i-th trial will approach 1 as i increases. Therefore, the final TL surrogate M^{TL} will be set to the target surrogate M^T. So we can have that,

With sufficient trials, the final TL surrogate will find the same optimum as the target surrogate does; that's, the final solution of surrogate M^{TL} will be no worse than the one in M^T given sufficient trials.

The above finding demonstrates that TransBO can alleviate negative transfer [37]. In other words, it can avoid performance degradation compared with non-transfer methods – the traditional BO methods.

B REPRODUCTION DETAILS

The source code and the benchmark data are available in the compressed file *"benchmark_data_and_source_code.zip"* on CMT3. The source code is also available in the anonymous repository [3] now. All files in the benchmark should be placed under the folder *'data/hpo_data'* of the project root directory. To reproduce the experimental results in this paper, an environment of Python 3.6+ is required. We introduce the experiment scripts and installation of required tools in *README.md* and list the required Python packages in *requirements.txt* under the root directory. Take one experiment as an example, to evaluate the static TL performance of TransBO and other baselines on Random Forest using 29 source tasks with 75 trials, you need to execute the following script:

python tools/static_benchmark.py –trial_num 75 –algo_id random_forest –methods rgpe,pogpe,tst,transbo –num_source_problem 29

Please check the document *README.md* in this repository for more details, e.g., how to use the benchmark, and how to run the other experiments.

[3] https://anonymous.4open.science/r/TransBO-EE01/