Enhancing the Performance of Bandit-based Hyperparameter Optimization

Yile Chen[1,2], Zeyi Wen[2,3*], Jian Chen[1*], Jin Huang[4]
[1]*South China University of Technology,* [2]*HKUST (Guangzhou),* [3]*HKUST,* [4]*South China Normal University*
jireh.x6@gmail.com, wenzeyi@ust.hk, ellachen@scut.edu.cn, huangjin@m.scnu.edu.cn

***Abstract*—Bandit-based methods are commonly used for hyperparameter optimization (HPO), which is significant in data analytics. When confronted with numerous configurations and high-dimensional large problems, existing bandit-based methods face challenges of high evaluation cost and poor optimization performance. To address these challenges, we introduce an improved bandit-based approach that exhibits enhanced evaluation ability and is suitable for situations with limited resources. Specifically, our method first effectively utilizes the feature and label information to conduct representative groups for further evaluation. After that, two kinds of folds (i.e., general folds and special folds) are constructed to facilitate better evaluation of the configuration in the cross-validation process. Additionally, we incorporate variance and subset size into the evaluation metric to comprehensively evaluate the configuration. We integrate our proposed method into three commonly used bandit-based methods, and experimental results on multiple datasets show that our method has advantages in stability with accuracy improvement of 1% to 15% on the datasets tested. In addition, since our method can avoid configurations that are low-quality but time-consuming to evaluate, it is always more efficient than the existing bandit-based methods, and can even reduce the execution time by half in some datasets. Sometimes it takes a little more time, but the improvement in accuracy can be significant.**

***Index Terms*—HPO, Bandit-based Hyperparameter Optimization, Data Sampling**

I. Introduction

Hyperparameter configurations have a significant impact on the performance of data analytic models, and hyperparameter optimization (HPO) [19] aims to find the best configurations for different problems. Among various HPO techniques, the bandit-based methods have demonstrated high performance and reliability in practice [20], [33]. The common implementations include Successive Halving [27], Hyperband [31], ASHA [30], and BOHB [17]. In the bandit-based methods, the whole evaluation process is divided into multiple iterations, each of which needs to be completed under a pre-defined budget (e.g., the number of instances). Firstly, all the configurations are evaluated, then the low-quality ones are filtered out, and the rest are passed to the next iteration. Hence, the number of candidate configurations decreases in each iteration, while the budget for each candidate configuration increases. Compared with other HPO methods (e.g., Bayesian optimization), the bandit-based methods evaluate all the configurations to find a satisfactory configuration within a budget.

*Corresponding authors.

However, the performance of bandit-based methods degrades when dealing with a vast number of configurations and challenging problems characterized by high-dimensional large problems. More candidate configurations lead to a smaller budget for each configuration, and hence a good configuration may get filtered out due to the coarse evaluation. Besides, a high-dimensional and large problem increases the cost of evaluating configurations. This phenomenon may lead to missing good-quality configurations. Increasing the budget can improve performance, but it also results in high time consumption.

To tackle these problems, this paper presents an enhanced bandit-based method that improves the performance of optimization, and meanwhile reduces time consumption. Our proposed method integrates feature and label information to construct representative groups for a more precise subset sampling for each configuration evaluation. Moreover, our method constructs general and special folds for the cross-validation process, to better evaluate the configurations. We further augment the performance and stability of the optimization process by incorporating variance and subset information into the evaluation metric. To summarize, this paper makes the following major contributions.

- We investigate the challenges faced by bandit-based methods in terms of performance, stability, and time consumption when dealing with a large number of configurations and complex problems. Meanwhile, we further explore and highlight the significant impact of the subset sampling process and the cross-validation process on optimization.
- To overcome these issues, we present a method that utilizes both feature and label information to perform better subset sampling through group construction. Furthermore, we introduce general and specific folds in the cross-validation process to better evaluate configurations. We further integrate variance and subset size into the evaluation metric, which enhances the optimization performance.
- Our experimental and theoretical findings showcase the superiority of our method in terms of performance, stability, and efficiency, especially in scenarios with a large number of configurations. Our proposed method achieves accuracy improvement ranging from 1% to 10%, while consuming less time and exhibiting lower variance. Furthermore, we have successfully employed our techniques in cross-validation experiments and regression problems, which further vali-

dates the efficiency of our method. By conducting separate experiments for each component, we have gained a deeper understanding of the characteristics and importance of each design aspect of our method.

II. Background

In this section, we begin by introducing various HPO techniques and then delve into the bandit-based method with cross-validation, which is the commonly used optimization method in HPO. Finally, we identify the shortcomings of existing bandit-based methods.

A. Hyperparameter Optimization

The quality of machine learning models is significantly impacted by the selection of hyperparameters, such as the learning rate for neural networks. How to select hyperparameters for specific problems, namely hyperparameter optimization (HPO), has recently gained significant attention in various domains. For example, AutoSF [49] investigates how to automatically select scoring functions for knowledge graphs. Auto-Model [45] utilizes existing research papers and HPO techniques to address the algorithm selection and HPO problem.

Traditionally, researchers use grid search [29], random search [8], or some rule-based heuristic methods [23], [25] to find good configurations. However, these methods are time-consuming and ineffective when dealing with high-dimensional large problems and numerous configurations [15]. State-of-the-art methods mostly rely on empirical information to build predictive models for HPO. These methods can be categorized into two groups based on their use of empirical information: utilizing the results from other datasets and utilizing the results from the current dataset.

The first group of methods utilizes results from other datasets to find high-quality configurations for new datasets. This involves constructing a mapping model between datasets and their best configurations. Meta-learning-based methods are commonly used in this category of HPO [24]. For example, auto-sklearn [18] uses the dataset similarity to recommend configurations for warm-starting, while methods like SmartML [35] and cSmartML [16] utilize meta-learning techniques to make recommendations. In addition, some researchers utilize transfer learning for HPO by exploiting different datasets [32], [41].

The second group of methods involves evaluating some configurations on the current dataset, and using the results to construct a model that predicts the quality of not yet evaluated ones for future recommendations. This group of methods includes three major types: Bayesian optimization based method, genetic algorithm based method and heuristic based methods. The Bayesian optimization based methods [43], one of the Sequential Model-Based Optimization (SMBO) methods [7], [26], use the Gaussian model to fit the mapping relationship between configurations and model performance. Genetic algorithm based HPO methods also utilize configurations evaluated on the current dataset for the next configuration recommendation [46], [48]. Optuna [2] is a representative of the heuristic based methods, and uses the trajectory details from past evaluations to identify promising configurations for finding the best hyperparameters in the shortest possible time.

The above-mentioned two groups of methods have their pros and cons. Configuration evaluation using information from other datasets offers the benefit of faster evaluation and the ability to swiftly tackle various problems. However, due to the differences between datasets, it is quite challenging to obtain a large amount of unified expression. Additionally, the differences in dataset distributions also affect the accuracy of the optimization. In contrast, training exclusively on the current dataset tends to produce more reliable and applicable results at each stage, but it is more time-consuming and often cannot cover every possible hyperparameter configuration. Next, we review bandit-based methods [11] which combine the advantages of two groups of optimization methods.

B. Bandit-based Optimization and Cross-Validation

Multi-fidelity optimization [19], especially the bandit-based methods [11], combines the advantages of two groups of optimization methods. Bandit-based methods strike a balance between budget (e.g., the number of instances) and the number of configurations. The fundamental concept of this method involves assigning a partial budget for each configuration and using the local results to filter low-quality configurations. Thus, this technique saves time during the evaluation process.

Building on this idea, researchers have proposed various bandit-based optimization algorithms. Successive Halving (SHA) [27] is one of the most well-known methods in bandit-based optimization. In each iteration, SHA distributes the budget evenly and evaluates the performance of each configuration. Based on the evaluation results, half of the configurations with poor performance are eliminated, and the remaining half are passed to the next iteration. This process is repeated until the most promising configuration is obtained. To better demonstrate this process, Figure 1 provides an example of SHA, which considers 8 configurations. In the first iteration, each of the 8 configurations is allocated $1/8$ of the total budget for evaluation. Based on the evaluation results, the top 4 configurations are retained and they enter into the next iteration of evaluation. During the second iteration, each configuration is evaluated with $1/4$ of the budget. This process is iterated until only one configuration remains in the fourth round. Finally, the model trained on the full dataset using the remained configuration becomes the result of SHA.

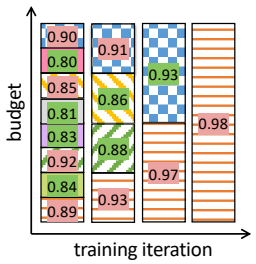

Fig. 1. SHA example.

Nevertheless, SHA may squander resources on substandard configurations due to inadequate resource allocation (i.e., some configurations need more resources while others allow for less). HyperBand [31] addresses this issue by using an "exploration-exploitation" strategy to efficiently allocate resources for various configurations, and it performs multiple

runs of SHA. HyperBand allocates different partial budgets for different configurations based on their performance in the previous iterations (i.e., exploiting high-quality configurations as a prior). Since the proposed of HyperBand, much work has been done to improve it: (i) to mitigate the slow convergence caused by random sampling in configurations of HyperBand, BOHB [17] integrates Bayesian Optimization into HyperBand; (ii) Asynchronous SHA (ASHA) [30] enhances the efficiency of the original HyperBand through asynchronous parallelization; (iii) Progressive ASHA (PASHA) [10] dynamically allocates computational resources during the optimization process, enabling more efficient exploration of the search space and faster convergence to suitable solutions; (iv) Differential Evolution HyperBand (DEHB) [5] utilizes a differential evolution algorithm to select configurations; (v) SMAC3 [34] integrates HyperBand and random forest into the process of Bayesian optimization.

In bandit-based methods, cross-validation is a widely used technique for the evaluation of individual configurations. The k-fold cross-validation divides the dataset into k folds, where $k-1$ folds are used for training a model and the remaining fold is used for validation to obtain a score. The process is repeated for k times and the average score serves as the evaluation result of the configuration. Experiments and research have shown that this is an effective validation method to mitigate overfitting in supervised learning [9], [22], [47]. However, due to the budget limit in bandit-based methods, the size of each fold may be quite small, especially when the number of configurations is large. As a result, the quality of evaluation using k-fold cross-validation in bandit-based methods is unstable, and we aim to improve it in this paper.

C. Limitations of Bandit-based Optimization

Current research mainly focuses on the selection of configurations and the allocation of the budget, but there is less attention given to the allocation quality. The budget allocated for each configuration is mostly constructed by random or stratified sampling in current bandit-based methods, which is also the same in the cross-validation process. This paper discovers that a more detailed instance sampling method can lead to notable enhancements in bandit-based optimization. At the same time, we have also identified three challenges with existing bandit-based optimization, which are outlined below,

- **Unstable Results**: The reliability of sampling is heavily influenced by the sampling size. In cases where the sample size is small, the sampling method can greatly impact the optimizing process. However, the sample size is often small in bandit-based methods, particularly when there are numerous configurations together with the need for further division into folds in k-fold cross-validation. This instability is most pronounced during the initial stages of optimization, impacting the evaluation of the majority of configurations and ultimately affecting the overall stability of the optimization process. This issue often leads to high variance.
- **Affected Performance**: A limited sampling size not only undermines the stability of finding high-quality configurations, but also compromises the accuracy performance of the recommended configuration. As only one model is trained on the entire dataset, other configurations are excluded during the optimization process. Unstable evaluation outcomes can result in superior configurations being disregarded, thereby affecting the final accuracy performance. This issue results in lower quality of the found configurations.
- **Time-consuming Process**: Unreliable evaluations can easily overlook superior configurations, resulting in a waste of time on suboptimal configurations. When these configurations require more search time, it increases the overall cost. While more attempts or a larger budget can decrease this instability, they also increase time consumption. Furthermore, as the number of potential hyperparameters, dataset complexity, and model size increases, the elapsed time also increases. Hence, discovering a way to achieve a more stable outcome in a shorter time is vital for the bandit-based method.

III. Our Proposed Method

To address the limitations of bandit-based optimization methods, we propose a novel bandit-based method in this paper, which considers the sampling and variance information to enhance optimization performance and stability. To facilitate presentation and understanding, a description of the main notations used in the paper is given in Table I.

TABLE I
THE DESCRIPTION OF NOTATIONS USED IN THIS PAPER.

Notation	Description
D	the dataset of n instances with u classes $\{d_i \mid i = 1, 2, ..., n\}$
$\mathcal{T}$	the space with m configurations $\{\tau_i \mid i = 1, 2, ..., m\}$
B	the budget allocated for the problem, which is the same as the instance number n in our method
$\mathcal{C}$	feature clusters for group construction $\{c_i \mid i = 1, 2, ..., v\}$
Ω	the intermediate sampling groups for subsequent cross-validation folds $\{\omega_i \mid i = 1, 2, ..., v\}$
$\mathcal{F}$	the set of k folds for k-fold cross-validation

Here, we introduce our method, utilizing SHA as the underlying framework, although our method is applicable to all other bandit-based methods. Figure 2 provides the overall framework of the proposed method. Prior to starting the HPO process, our method clusters the instances based on their features. Then, we can divide the original dataset into multiple groups using clustering results and instance labels, to assist in subsequent configuration evaluation (from subfigure (a) to subfigure (d)). Once the HPO process starts, the method iteratively evaluates and filters candidate hyperparameter configurations until the desired configuration is selected (from subfigure (e) to subfigure (j)). During each evaluation of a configuration τ_i, our proposed method constructs two types of folds, general and special folds, from the groups for cross-validation (subfigure (f)). After obtaining evaluation results on each fold (subfigure (g)), the method calculates the evaluation score s using the mean μ, the variance σ, and the subset size γ to aid in the halving operation (subfigure (h)). In the following

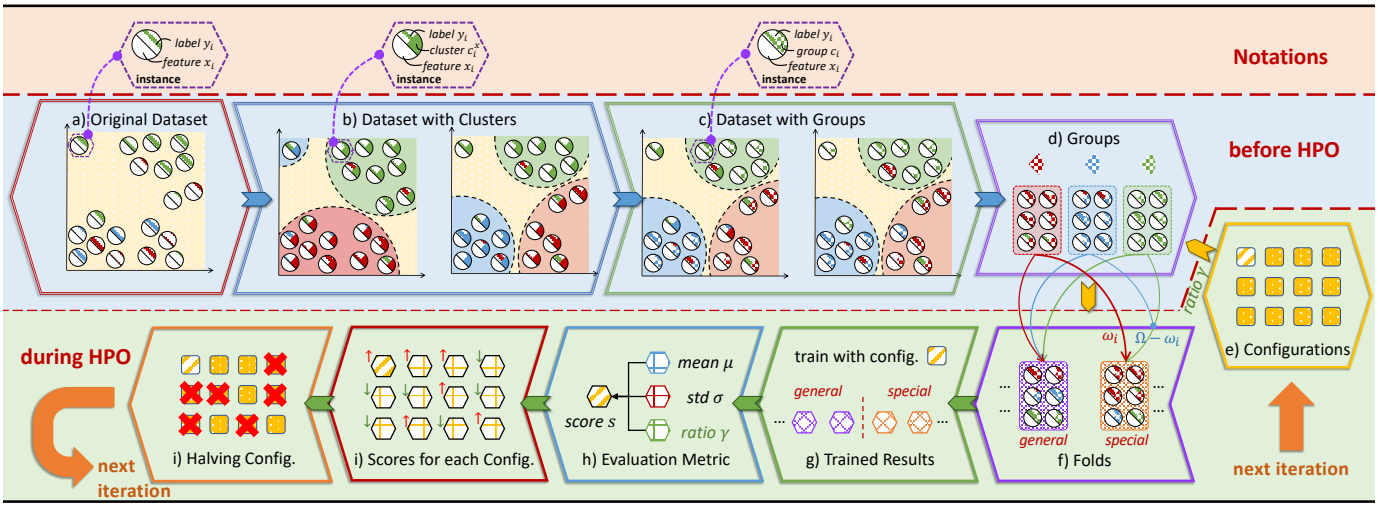

Fig. 2. The overall framework of our proposed sampling-based hyperparameter optimization methods.

section, we provide a detailed explanation of three main parts of the proposed method, including: i) the instance grouping; ii) the fold construction; and iii) the score calculation.

A. Instance Grouping based on Features and Labels

Bandit-based optimization with instances as budget is a method that samples a subset from the entire dataset and evaluates the performance of hyperparameter configurations on that subset. Current methods usually employ a random or stratified approach to sample these subsets. When the subset is sizable, constructing it in a vanilla way can effectively capture the overall dataset's distribution. However, for hyperparameter optimization, the candidate hyperparameter space is often vast and intricate, resulting in a limited number of instances that can be assigned to each subset. Therefore, subsets obtained using vanilla methods often face difficulty in accurately reflecting the overall distribution and exhibit randomness.

To overcome this limitation, we propose a grouping-based subset sampling method to obtain a more representative subset for evaluation. Specifically, our method involves constructing multiple groups with label and feature information before initiating hyperparameter optimization process, and subsequently sampling subsets from these groups for evaluating configurations. The group construction process is depicted in the blue part of Figure 2, specifically in subfigures (a) through (d). Initially, each instance d_i in the dataset D consists of a feature vector $\boldsymbol{x}_i$ and a label y_i, as shown in Figure 2(a). We start by applying clustering methods to cluster the instances based on their features, which provide category characteristics and are denoted as c_i^x. Additionally, we process the label information to obtain category characteristics for each instance and denote it as c_i^y, as illustrated in Figure 2(b). Finally, we merge the two types of category information, resulting in the final groups c_i shown in Figure 2(d).

To perform the feature clustering process, our method can employ various clustering algorithms such as k-means, mean-shift, and affinity propagation. For the sake of simplicity and efficiency, we utilize the k-means method for clustering in this paper. As the oldest but most popular method, k-means has been extensively studied and applied in various domains [38], [50]. In our method, we propose to use k-means as the clustering algorithm. Performing k-means should consider the number of instances within each cluster to avoid imbalanced instance distribution affecting subsequent cross-validation. Therefore, the clustering process involves iteratively performing k-means multiple times. If a particular cluster has very few instances (less than r_{group} ratio of the average number of instances per cluster, $\frac{n}{k} \times r_{group}$), we remove these instances and re-cluster the rest until each cluster has the desired number of instances, as shown in Figure 2(b). After completing the clustering, each instance is associated with a cluster and we can represent the clustering results by $C_x = \{c_1^x, c_2^x, ..., c_n^x\}$, where the superscript indicates that the results are obtained based on feature information (i.e., $\boldsymbol{x}$). To distinguish the notation, we use the v to represent the number of clusters to construct. Considering the fold construction in subsequent cross-validation, we typically choose a smaller value as the v (such as 2-5). We provide further details on this value setting in Section III-B when we elaborate on the details of special folds. Similarly, we can obtain label information for each instance (i.e., $C_y = \{c_1^y, c_2^y, ..., c_n^y\}$), and we can use the original label y_i directly as c_i^y. However, when dealing with highly imbalanced datasets where there are very few instances in a certain class (less than $\frac{n}{\mu} \times 10\%$), we merge that class with other less frequent classes for further processing. As for the regression problem without classification labels, we can directly divide numerical labels based on their magnitude and assign them to different categories.

Through label-based and feature-based division, we get two different categories of labels, the feature category C_x obtained by clustering and the label category C_y obtained by classification. Both categories are utilized to construct groups for future configuration evaluation in our proposed method. The number of groups constructed in our method equals the cluster number v, which serves for subsequent cross-validation. Operation 1 shows the group construction

Operation 1: **GenGroups**(C_x, C_y, D).

Input: Feature Cluster $C_x = \{c_1^x, ..., c_n^x\}$; Label Class $C_y = \{c_1^y, ..., c_n^y\}$; Original Dataset $D = \{d_1, ..., d_n\}$.

1 $\Omega = \{\omega_1, ..., \omega_\nu\}$;
2 /* count class-cluster: $counts[i, j]$ is the number of instance with class i & cluster j */
3 $counts \leftarrow$ CountNumber(C_x, C_y);
4 /* s1. allocate ins. in clusters */
5 $D_{ing} \leftarrow D$;
6 **for** $j \in \nu$ **do**
7 $class \leftarrow$ top-$k(counts[:, j])$;
8 $\omega_j \leftarrow \{d_i; c_i^x = j, c_i^y \in class\}$;
9 $D_{ing} \leftarrow D_{ing} \setminus \omega_j$;
10 **end**
11 /* s2. allocate remaining ins. */
12 **for** $i \in m$ **do**
13 $c \leftarrow$ Argmax$(p[i, :])$;
14 $\omega_c \leftarrow \omega_c \bigcup \{d_k; d_k \in D_{ing}, c_k^x = c\}$;
15 $D_{ing} \leftarrow D_{ing} \setminus \omega_c$;
16 **end**

Output: the instance groups $\Omega = \{\omega_1, ..., \omega_\nu\}$

process balancing instances from two categories. Meanwhile, Figure 2(c) provides an example of group construction with three classes and three clusters. Firstly, our method counts the number of instances with different classes c_i^y and clusters c_i^x (Line 2). Then, the method analyzes each cluster individually to assign their corresponding group labels (Lines 3-8). In a single cluster j, the top-k classes with the highest proportions are selected (Line 5), and their instances are assigned to the corresponding group (Line 6), shown in the left subfigure of Figure 2(c). The selection of the k value here is determined by the total number of categories in our method. Afterward, the remaining unallocated instances are assigned based on the relationship between categories and clusters. Each instance is assigned to the group corresponding to the cluster with the highest proportion in that category, as shown in the right subfigure of Figure 2(c). Thus, we can generate the groups and obtain the group labels c_i corresponding to each instance d_i, as depicted in Figure 2(d).

B. General and Special Fold Construction

During the evaluation of each configuration, bandit-based optimization typically employs k-fold cross-validation to obtain a more generalizable result. In this approach, each fold is obtained by random or stratified sampling based on the label. However, when the subset for configuration evaluation is small, we find that such a sampling method easily results in unstable evaluation since the subset distribution is difficult to reflect the characteristics of the complete dataset. In contrast, our method achieves a more comprehensive evaluation for configurations by utilizing general folds, which closely reflect the average distribution of the complete dataset, and special folds, which reflect the distribution specific to particular groups to address the problem of insufficient data. The construction process of these folds is depicted in Figure 2(f).

Similar to stratified sampling, a general fold is uniformly sampled from different groups, which tries to simulate the global distribution. Compared to stratified sampling, which only considers the class labels, group-based sampling comprehensively considers the whole dataset distribution. The folds obtained by sampling in this manner better align with the overall dataset distribution, and the scores obtained by training and calculating on such folds are more similar to the overall dataset than other random folds, resulting in good generalization. However, the generalization performance is closely related to the subset size used for evaluation. Meanwhile, the subset size is relatively small in the bandit-based optimization when a large number of configurations are involved, especially at the start of the optimization process. To address the shortcomings of general folds when dealing with small subsets, our method introduces the special fold in cross-validation.

Different from the general folds, the special folds try to find a set of instances that deviates from the overall distribution. Meanwhile, there are significant differences between each special fold to ensure the diversity of folds. Although the result on a single special fold may not accurately reflect the configuration performance on the entire dataset, the combined results from multiple special folds can provide insight into its performance under different conditions. Specifically, our method samples corresponding special folds based on the previously generated groups. Each of these special folds represents the data distribution within a particular group.

Operation 2: **GenFolds**$(\Omega, k_{gen}, k_{spe})$.

Input: Instance Groups $\Omega = \{\omega_1, ..., \omega_m\}$

1 **Initialize:** $\mathcal{F} \leftarrow \{\}$;
2 /* Generate general Folds */
3 $\mathcal{F}_{gen} \leftarrow$ StratifiedKFold(Ω, k_{gen});
4 /* Generate special Blocks */
5 **for** $i \in k_{spe}$ **do**
6 $\mathcal{F}_{spe} \leftarrow \mathcal{F}_{spe} +$ {StratifiedSample$(\Omega \setminus \omega_i)$, Sample$(\omega_i)$}
7 **end**

Output: the instance folds $\mathcal{F} = \mathcal{F}_{gen} \bigcup \mathcal{F}_{spe}$

The overall algorithm for the fold construction is shown as Operation 2. Firstly, the method performs stratified sampling on groups to obtain k_{gen} general folds (Line 2). Then, for each group ω_i, our method samples several instances from ω_i (e.g., 80% of the fold) and some instances from remaining groups $\Omega \setminus \omega_i$ (e.g., 20% of the fold) to construct k_{spe} special folds (i.e., the number of clusters v in Section III-A). This approach ensures that the folds have both a general distribution and a special distribution for subsequent evaluation. Furthermore, while methods like the elbow strategy [42] automatically select the value of v to improve the quality of clusters, considering that v is also needed in fold construction and cross-validation, we choose to set v to be not larger than 5. This allows the total number of folds, $k_{gen} + k_{spe}$, to be equal to the commonly used setting of 5 in cross-validation for small subsets [3], [4], [12]. Additionally, a smaller value for v can also help reduce the cost of clustering and cross-validation.

C. Evaluation Metric with Variance and Sampling Size

After obtaining the folds, we proceed to train and calculate accuracy results for each fold, as depicted in Figure 2(g). Traditional optimization methods typically evaluate the performance of configurations based on the average accuracy μ.

However, to find high-quality configurations and prevent the exclusion of potentially good configurations, we incorporate information on variance σ and sampling size γ into our evaluation metrics during the development process. In the following section, we provide a comprehensive explanation of our metric design.

Using the Variance Information:

The bandit-based method allocates budgets to different configurations, and evaluating each configuration on a relatively small number of instances is often unstable due to the sampling process. Inspired by the acquisition function in Bayesian optimization, we propose a novel metric that considers both the average accuracy and the variance to effectively analyze the future performance of each configuration.

In Bayesian optimization, the algorithm trains a surrogate model on the available results to obtain the mean μ and variance σ accuracy of the not yet evaluated points. The exploration-exploitation trade-off is then adjusted through the acquisition function to select the next configuration. Different acquisition functions balance the mean and variance differently, but all tend to favor configurations with higher mean (better exploitation) and higher variance (better exploration). The Upper Confidence Bound (UCB) is a direct metric design that combines the mean and standard deviation results in a weighted manner, as shown in the formula below,

$$UCB = \mu(x) + \alpha\sigma(x) \tag{1}$$

where α represents the adjustment parameter.

In this paper, we adopt this weighted form to comprehensively evaluate configurations by considering both mean and variance information. Unlike the results estimated by the Bayesian optimization surrogate model, both the mean and variance results used in our method are derived from actual results. For the selection of weights, we still take positive values to ensure their exploratory ability. Although a smaller variance indicates more stable results, a larger variance is more meaningful for bandit-based subset training evaluation. However, the influence of variance is also related to the subset size, which is also the reason we take sampling information into consideration when designing the evaluation metrics.

Using the Sampling Information:

In the design of the metric, our consideration of variance is not only limited to the exploration, but also the reliability of the evaluation. In order to achieve better exploration, the method selects configurations with higher variance. However, as the bandit-based optimization progresses, the candidate hyperparameter configurations gradually decrease, and the subset size used for evaluating each configuration increases. Models trained on subsets of different sizes have varying levels of stability, and results obtained from larger subsets are more reliable. In other words, for datasets of different sizes, we need to adjust the weight for variance in the evaluation metrics. Therefore, we add a new weight β to Equation 1 to consider the impact of the subset size.

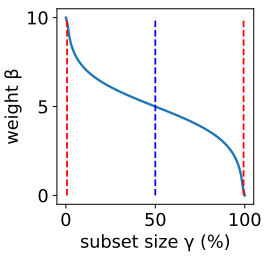

Fig. 3. $\beta - \gamma$ line figure.

The weight β represents the impact of the subset size on the evaluation, and its design is based on two assumptions: i) the larger the dataset, the smaller the weight, and ii) the weight change should not be a uniform process. Firstly, the subset size directly affects the difference between the evaluation results and the overall results. Evaluations on smaller datasets are more unstable and more prone to bias, so in this case, greater consideration should be given to variance in the evaluation. Secondly, in bandit-based optimization, the subset size is determined by the number of configurations, denoted as B/n, and changes exponentially. Therefore, the change in weight should be similar to the change in the number of configurations, rather than a linear relationship with the size. In other words, the weight should change more significantly for smaller sizes. Additionally, considering that the method can also be directly applied to cross-validation, we have made a symmetric design for sizes larger than 50% (corresponding to only two configurations participating in the optimization). To address these considerations, we use the hyperbolic tangent function $\tanh(x)$ and hyperbolic arctangent function $\text{atanh}(x)$ to design the sampling weight, which aligns with the above hypothesis. The definitions of $\tanh(x)$ and $\text{atanh}(x)$ are as follows,

$$\tanh(x) = \frac{\sinh(\text{x})}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\text{atanh}(x) = \frac{1}{2}\log\frac{1+x}{1-x}$$

We set the maximum weight value as β_{max}, which is recommended to be $1/\alpha$ to get achieve the normalization for the combined weight $\alpha \times \beta$. Then, β is expressed as follows,

$$\beta(\gamma) = 2\text{atanh}(1 - 2\max(\gamma_{\text{min}}, \min(\gamma_{\text{max}}, \gamma))) + \frac{\beta_{\text{max}}}{2} \tag{2}$$

where, $\gamma_{\text{min}} = 50(1 - \tanh(\frac{\beta_{\text{max}}}{4}))$ and $\gamma_{\text{max}} = 50(1 - \tanh(-\frac{\beta_{\text{max}}}{4}))$ represent the maximum and minimum thresholds, preventing weights from being excessively large or small. Figure 3 shows a line graph of β changing with the sampling ratio $\gamma = \frac{\|b_t\|}{\|B\|} \times 100$, in which $\beta_{\text{max}} = 10$. The design of weight β complies with the two assumptions mentioned above and exhibits the expected effects in the experiments.

The Final Evaluation Metric:

Combining the variance weight α (as Equation 1) and sampling weight β (as Equation 2), we obtain the evaluation metric of the proposed method in this paper as follows,

$$s(x, y, \gamma) = \mu(x, y) + \alpha\beta(\gamma)\sigma(x, y) \tag{3}$$

where $\boldsymbol{x}$, y, and γ represent the features, labels, and dataset sampling size used for training respectively, while μ and σ represent the mean and standard deviation of the results across different fold results.

Based on the results of cross-validation and the subset size used for training, we obtain the metric for evaluating

configurations, as shown in Figure 2(h). After obtaining scores for different configurations, the halving operation can be performed based on these results, as illustrated in Figure 2(i) and (j). Through iterative steps, our proposed method can determine the best configuration τ^*.

D. Overview of Our Method

Here, we present an overall explanation of our proposed method by providing the pseudocode in Algorithm 1 after introducing its three main components. Consistent with the vanilla optimization method, we have the dataset D, a budget B (i.e., the total number n of instances in the training dataset), and m configurations at the beginning of optimization.

Before beginning the evaluation of each configuration, the method first constructs multiple groups Ω based on the feature and label information of the dataset D to assist with the subsequent optimization process (Line 3). Specifically, the method performs multiple rounds of k-means clustering on all instances based on their features $\boldsymbol{x}$ to obtain v clusters $\mathcal{C}$ with relatively even assignments. Each instance d_i is assigned a feature-based category c_i^x, and a corresponding label-based category c_i^y can be obtained from the label y. Afterward, our method mixes the two types of categories and generates the desired groups Ω, as shown in Operation 1.

During the optimization process, similar to vanilla bandit-based methods, the proposed method iteratively evaluates and halves candidate configurations (Lines 5-18) until the final configuration τ^* is selected (Line 20). In a single iteration, our method allocates a budget b_t for each configuration, and in the case of SHA, the budget is allocated evenly to all candidate configurations (Line 7). Once the allocated budget is obtained, the method evaluates each configuration through cross-validation (Lines 9-14) and performs halving operations based on the evaluation results (Line 16).

Our method also enhances the vanilla cross-validation from two aspects: (i) the fold construction (Line 11) and (ii) the evaluation design (Line 13). Firstly, in the construction of folds, unlike the vanilla method that obtains folds through random sampling or stratified sampling from the subset with b_t, our method uses the groups obtained before optimization to construct folds. Moreover, to better evaluate with small subsets, we construct two types of folds, general and special, in cross-validation, as shown in Operation 2. The general fold is obtained by uniform sampling from each group, while the special fold is obtained through biased sampling. Biased sampling means sampling the most instances from a certain group and a few instances from other groups. The number of special folds, k_{spe}, is equal to the number of groups v. It is advisable to keep the value of v within 5 in order to ensure that the fold number $k_{gen} + k_{spe}$ remains 5. Secondly, in the design of the evaluation metric, we add considerations for variance and sampling information to the vanilla metric of mean accuracy. The complete metric is shown in Equation 3. The variance information is added to the metric in a weighted form and the subset size γ used for evaluation is also included in the metric as a parameter for the weight β. This metric considers the significance of variance in the evaluation under subsets of different sizes. Specifically, in small subsets, the validation stability is weak, and considering variance helps retain configurations with greater potential. In large subsets, the validation results are closer to the results of overall training, and the mean value is more important than the variance. Through the two improvements, our improved cross-validation can better adapt to the characteristics of resource allocation and iterative optimization in bandit-based methods.

Algorithm 1: Our proposed optimization method.

Input: Dataset $D = \{D_{train}, D_{test}\}$, budget B, m configurations $\mathcal{T} = \{\tau_1, ..., \tau_m\}$

1 **initialize:** $t \leftarrow 0$, $\mathcal{T}_t \leftarrow \mathcal{T}$
2 /* form groups for optimization */
3 $\Omega \leftarrow$ GenGroups(D)
4 /* evaluation for configurations */
5 **while** $\|\mathcal{T}_t\| > 1$ **do**
6 // budget for each τ_i
7 $b_t \leftarrow B/\|\mathcal{T}_t\|$
8 // evaluate configuration
9 **for** $\tau_i \in \mathcal{T}_t$ **do**
10 // generate folds
11 $\mathcal{F}_i \leftarrow$ GenFolds(D_{train}, Ω, b_t)
12 // calculate the score
13 $S_t \leftarrow$ CalScore$(\tau_i, \mathcal{F}_i)$
14 **end**
15 // filtrate the configurations
16 $\mathcal{T}_{t+1} \leftarrow$ HalvingConfig$(\mathcal{T}_t, S_t)$
17 $t \leftarrow t + 1$
18 **end**
19 /* get the best configuration */
20 $\tau^* \leftarrow$ GetBestConfig$(\mathcal{T}_t)$

Output: the selected configuration τ^*

E. Discussion on Our Method

After the description of our method, we discuss our method from the accuracy performance, time cost, and memory cost in this section.

Accuracy Performance: The bandit-based method, which uses the instances as the budget, reduces the optimization time by evaluating configurations on subsets rather than the whole dataset. However, the results on the subset lead to unstable evaluation results and are more likely to discard good configurations in early iterations, especially when there are a large number of configurations. In this paper, we improve the performance of bandit-based methods by considering variance and sampling information in subset sampling and cross-validation. A more detailed subset sampling approach that utilizes both features and labels makes the sampling subsets more representative, helping boost small subset evaluation quality. Moreover, the cross-validation using both general and special folds is more suitable for small subsets in bandit-based optimization than the vanilla approach. In addition, the proposed method considers variance and subset size in the evaluation metric, which helps preserve configurations with high possibility and alleviates prematurely discarding good configurations.

Optimization Stability: In terms of stability, our method exhibits better stability compared to existing methods in both sampling methods and optimization approaches. Firstly, the group-based sampling method proposed in this paper effectively utilizes information from both features and labels, making it more stable compared to random sampling and stratified sampling which only utilize label information. We analyze the stability of sampling using a binary distribution as an example, as shown in Proposition 1. Secondly, the stability of the sampled subsets contributes to the overall stability of the training and evaluation process in optimization. By ensuring stable sampling, the subsets used for training become more consistent, leading to reduced variability among the trained models and improved overall stability. Furthermore, it is worth noting that when the evaluation is performed on a small subset, the performance may not accurately reflect the results of the entire dataset. In our method, the introduction of special folds and evaluation methods that consider the sampling size and evaluation variance enhances the tolerance of the evaluation process and, consequently, improves the stability of the optimized configurations.

Proposition 1 (Sampling Stability)**.** *Suppose we have a dataset that is evenly distributed between two categories. When the subset obtained from sampling is small, we can use a binomial distribution to represent the process of random sampling. The probability function can be expressed as* $P(x; n, p) = C_n^x p^x (1-p)^{n-x}$*, where* x *represents the sampled number of positive instances,* n *represents the sampling size, and* p *represents the number of positive instances. For our method, assuming there are two groups with the same size* $n/2$*, the probabilities of the positive category are denoted as* $p_1 = p - \epsilon$ *and* $p + \epsilon$ *respectively. Then, the distribution of the subset can be represented as* $P_{our}(x; n, p_1, p_2) = \sum_{i=0}^{x} P(i; \frac{n}{2}, p - \epsilon) \cdot P(x - i; \frac{n}{2}, p + \epsilon)$*, where* $\epsilon \in [0, p]$*. We consider two extreme scenarios here,* $\epsilon = 0$ *and* $\epsilon = p$*. When* $\epsilon = 0$*, the distribution of the group is consistent with the distribution of the overall dataset. The sampling results of our method are consistent with those of random sampling, and the probability of being consistent with the distribution of the overall dataset is* $P(\frac{n}{2}, n, p)$*. On the other hand, when* $\epsilon = p$*, each group corresponds exactly to one category of instances, and the sampling results of our method always match the distribution of the overall dataset. Compared to random sampling and stratified sampling using only label information, our method is more likely to achieve an* ϵ *that is closer to the actual distribution* p*, thereby demonstrating better sampling stability.*

Time Complexity: In terms of time consumption, the main difference between our proposed method and the vanilla method is on the grouping operations before evaluation. The time consumption of grouping operations primarily depends on feature clustering. The time complexity of k-means clustering is $O(n * f * v * d)$, which is related to the number of instances, the dimension of features, the number of iterations in clustering, and the number of clustering centers. Among them, the number of clusters in our method is usually less than 5, and the number of iterations of k-means defaults to 10, so the time spent at this time is less than $50nf$. For the searched model, take the simplest three-layer binary classification BP neural network as an example, assuming that the number of neurons in each layer is $(f, h, 2)$, then its feedforward calculation and feedback for a single instance is $f * h + h * 2$, and the time to train the entire data set is $2(fh + 2h) = 2hnf + 4h$. In other words, the time required for clustering is equivalent to training a hidden layer with 25 neurons for one epoch, which can be ignored in hyperparameter optimization problems. Therefore, the method proposed in this paper does not bring significant time consumption compared with the vanilla method. On the contrary, an improved hyperparameter selection process reduces the time spent on repeated training attempts and on inefficient large-scale models, ultimately resulting in a decrease in overall optimization time.

Memory Complexity: In terms of memory cost, the memory usage of k-means is introduced during the construction of grouping, and its space complexity is $O((n + v)f)$. However, when the cluster labels are obtained, the cluster model does not need to be saved, and the memory can be freed. The final memory occupied is only the group tags of n instances. In addition, the clustering accuracy requirement in the method proposed in this paper is not high, which is only used to obtain certain feature information. When the amount of data is huge, it is sufficient to take only a part of the dataset for training the cluster. In other words, the memory cost of k-means can be further reduced.

IV. Experimental Studies

To investigate the rationality of the design of our method, we conduct experiments and present the experimental results in this section. We first introduce the datasets and experimental settings used in the study. Then, we exhibit two overall experiments to compare the proposed methods with some baselines in terms of hyperparameter optimization and cross-validation. Furthermore, we illustrate the significance of three main components in our method through corresponding independent experiments, including: i) the group construction based on feature and label; ii) the general and special folds in cross-validation; and iii) the evaluation metric with variance and sampling information. The corresponding code for our method can be found in the respective repository on GitHub https://github.com/JirehChan/EnhancingBHPO.

A. Datasets and Experimental Setup

We used 12 public datasets from LibSVM [14], UCI [36] and Kaggle [1] in the experiments, including eight binary classification datasets and two multi-category datasets and two regression datasets. Most of the datasets utilized in the experiments exhibit a balanced distribution, while a few are imbalanced (i.e., *machine*, *a9a*, *fraud*, and *satimage*). Table II shows the summary of these datasets. For the datasets that do not contain a test set, we used the 80/20 rule to construct

the train set and test set. In this paper, each experiment was repeated five times with different random seeds, and the average value was taken as the result for presentation.

TABLE II
INFORMATION OF THE DATASETS USED.

type	dataset	#classes	#train	#test	#features
binary classification	australian	2	690	-	14
	splice	2	1,000	2,175	60
	gisette	2	6,000	1,000	5,000
	machine [28]	2	10,000	-	9
	NTICUSdroid [37]	2	29,332	-	86
	a9a	2	32,561	16,281	123
	fraud [44]	2	284,807	-	86
	credit2023 [39]	2	568,630	-	29
multi-category classification	satimage	6	4,435	2,000	36
	usps	10	7,291	2,007	256
regression	molecules [13]	/	16,242	-	1275
	kc-house [21]	/	21,613	-	18

In the experiment, we used a neural network as the model for optimization. The configuration space for hyperparameter search is shown in Table III, which includes 8 different hyperparameters involving model structure and training hyperparameters. In this study, we utilized the implementation of MLPClassifier and MLPRegressor from the scikit-learn library [40], and the specific meanings of each hyperparameter can be found in its documentation [12]. The experiments were conducted on a machine with an Intel(R) Xeon(R) Silver 4210 CPU of 126GB main memory running on a Linux OS.

TABLE III
INFORMATION OF THE HYPERPARAMETER SEARCH SPACE.

name	range
hidden layer sizes	[(30), (30,30), (40), (40,40), (50), (50,50)]
activation	['logistic', 'tanh', 'relu']
solver	['lbfgs', 'sgd', 'adam']
learning rate init	[0.1, 0.05, 0.01]
batch size	[32, 64, 128]
learning rate	['constant', 'invscaling', 'adaptive']
momentum	[0.7, 0.8, 0.9]
early stopping	[True, False]

B. Hyperparameter Optimization Experiment

In this section, we present our experiments on HPO. Initially, we applied the proposed method to three bandit-based methods (SHA, HyperBand, and BOHB) and compared them with its vanilla version. Secondly, we evaluated the proposed method and vanilla method under varying settings of configurations and analyzed the impact of the hyperparamer types and model complexity for different methods.

Compare with Different HPO Methods:

Firstly, we conducted a comparison between our proposed method and vanilla methods in terms of hyperparameter optimization with 4 hyperparameters (i.e., "hidden layer sizes", "activation", "solver" and "learning rate init") and $6 \times 3^3 = 162$ configurations. The vanilla methods included the random search and three bandit-based methods, as detailed below,

- random: randomly select 10 configurations for evaluation.
- SHA: the vanilla Successive Halving algorithm, which is implemented in the scikit-learn library.
- HB: an improved algorithm for SHA, this paper uses the implementation in the HpBandSter-sklearn library [6].
- BOHB: an improved bandit-based method that combines the Bayesian optimization. In the experiments, we use the HpBandSter-sklearn's implementation for experiments.

In addition to the above baselines, we also compared our method with other optimization methods (i.e., SMAC3 [34] and Optuna [2]). We found that these methods performed similarly to random search when the time budget was similar to Successive Halving which was the ground truth we compared with. For example, in *NTICUSdroid*, SMAC3 achieved a test accuracy of 96.62% in 1880 seconds, Optuna achieved 96.42% in 1776 seconds, and the random approach achieved 96.73% in 1798 seconds. Therefore, we only kept the random search to showcase the performance of our method in the paper. As for our method, we applied it to three bandit-based optimization algorithms, denoted as "SHA$^+$", "HB$^+$", and "BOHB$^+$", respectively. All the methods were evaluated using 5-fold cross-validation. We set the number of general folds k_{gen} to 3 and the number of special folds k_{spe} to 2 in our method. For other settings in our method, we set the r_{group} as 0.8, α as 0.1 and $\beta_{\max}$ as 10. The remaining parameters are consistent with the vanilla bandit-based methods. For more details, please refer to the scikit-learn and HpBandSter-sklearn.

Table IV presents the final results of performance evaluation, including accuracy or F1-score for classification datasets and R2 score for regression datasets, along with the corresponding search times for different methods. It can be observed our improved version outperforms the vanilla method in all bandit-based methods by finding configurations with higher test accuracy and lower variance. This highlights the importance of a more refined sampling strategy and metric evaluation for optimizing results and stability. In general, the proposed method can achieve an improvement of more than 1% in test accuracy than the vanilla method. Furthermore, our improvements can achieve good results in cases where vanilla methods perform poorly. For instance, our BOHB$^+$ method achieved a 14% increase in test accuracy compared to the vanilla method in the *usps* dataset. In terms of search time, the proposed method did not result in a significant increase in search time. On the contrary, in most cases, the search time of the proposed method is less than the vanilla method. This is because the proposed method accurately evaluates and selects hyperparameter configurations, avoiding the time-consuming training of large-scale models but having poor performance. In some cases, this reduction in search time can be as much as nearly 50%, such as the *a9a* dataset with HB. The proposed method may take longer than the vanilla method for a few datasets, but its accuracy improvements are also quite significant (e.g., HB$^+$ improves 15% accuracy in

TABLE IV
TRAIN RESULT (%), TEST RESULT (%) AND SEARCH TIME (SEC.) OF DIFFERENT HYPERPARAMETER OPTIMIZATION METHODS.

dataset	metric	**random**	bandit-based methods					
			SHA	**SHA$^+$**	**HB**	**HB$^+$**	**BOHB**	**BOHB$^+$**
gisette	trainAcc. (%)	99.97±0.06	100.00±0.00	99.96±0.07	82.67±28	99.79±0.32	99.27±1.24	99.99±0.01
	testAcc. (%)	96.87±0.35	97.00±0.36	**97.43±0.25** ✔	81.43±27	96.87±0.35 ✔	96.10±0.36	97.27±0.25 ✔
	time (sec.)	3613±71	1927±145	2098±290 ✘	**365±179**	816±286 ✘	408±285	454±130 ✘
NTICUSdroid	trainAcc. (%)	97.69±0.02	97.72±0.57	97.80±0.26	97.42±0.76	97.68±0.95	98.02±0.38	98.05±0.17
	testAcc. (%)	96.73±0.06	96.78±0.07	**96.92±0.07** ✔	96.61±0.36	96.64±0.28 ✔	96.39±0.06	96.43±0.03 ✔
	time (sec.)	1798±246	359±57	**344±87** ✔	1782±126	1102±37 ✔	732±596	424±586 ✔
credit2023	trainAcc. (%)	95.83±0.32	94.93±0.30	96.01±0.38	77.72±24.09	80.35±26.21	89.91±5.82	89.50±2.75
	testAcc. (%)	95.02±0.35	94.81±0.38	**95.92±0.37** ✔	77.76±24.11	80.36±23.23 ✔	84.91±5.82	89.50±2.76 ✔
	time (sec.)	3761±389	1897±295	**1780±332** ✔	5274±360	7510±439 ✘	11851±1234	6132±149 ✔
machine	trainF1. (%)	98.18±0.17	98.37±0.15	98.33±0.14	98.20±0.01	98.37±0.19	98.20±0.00	98.30±0.16
	testF1. (%)	98.09±0.33	98.30±0.01	98.39±0.16 ✔	98.24±0.02	**98.44±0.20** ✔	98.25±0.00	98.32±0.19 ✔
	time (sec.)	46±3	**16±1**	18±4 ✘	81±21	38±39 ✔	52±29	35±14 ✔
a9a	trainF1. (%)	90.18±0.21	90.84±0.49	90.33±0.02	91.08±1.30	90.76±0.70	92.21±1.40	91.06±1.32
	testF1. (%)	90.21±0.20	90.12±0.18	**90.50±0.08** ✔	89.51±0.52	90.33±0.18 ✔	89.06±1.18	90.00±0.49 ✔
	time (sec.)	12500±1398	1702±241	**1562±226** ✔	3590±2798	1795±622 ✔	1802±2389	3508±488 ✘
fraud	trainF1. (%)	99.91±0.02	99.88±0.04	99.92±0.00	99.92±0.00	99.92±0.00	99.92±0.00	99.92±0.00
	testF1. (%)	99.90±0.02	99.88±0.04	**99.91±0.00** ✔	**99.91±0.00**	**99.91±0.00** ✔	**99.91±0.00**	**99.91±0.00** ✔
	time (sec.)	2191±202	294±84	**225±69** ✔	7197±8385	2688±38 ✔	8849±8913	2958±56 ✔
usps	trainAcc. (%)	99.98±0.00	99.87±0.16	99.97±0.01	99.94±0.06	99.91±0.13	83.28±28.95	98.69±2.22
	testAcc. (%)	92.91±0.25	92.89±1.00	**93.74±0.32** ✔	92.01±1.01	93.11±0.67 ✔	78.39±25.65	92.31±0.78 ✔
	time (sec.)	2242±126	962±268	1031±68 ✘	63±23	41±4 ✔	165±147	**34±9** ✔
satimage	trainF1. (%)	98.75±1.09	94.19±0.68	96.18±2.44	87.80±7.92	98.32±1.47	88.22±7.19	98.58±1.39
	testF1. (%)	86.68±0.20	86.62±0.46	**87.88±0.13** ✔	82.77±3.64	86.22±1.52 ✔	84.26±4.84	86.52±0.64 ✔
	time (sec.)	949±117	339±102	273±26 ✔	40±40	60±35 ✘	**17±11**	23±16 ✘
molecules	trainR2 (%)	99.00±0.20	98.85±0.16	98.90±0.20	98.04±1.15	98.80±0.07	98.98±0.05	98.74±0.07
	testR2 (%)	98.55±0.17	98.51±0.17	**98.75±0.13** ✔	97.97±1.03	98.68±0.02 ✔	98.23±0.74	98.84±0.11 ✔
	time (sec.)	1235±235	1058±50	960±213 ✔	1932±1856	1534±1374 ✔	984±477	**639±158** ✔
kc-house	trainR2 (%)	93.53±0.16	92.05±0.58	92.45±0.63	52.75±11.19	84.54±7.57	70.05±25.23	85.41±12.33
	testR2 (%)	88.95±0.36	88.27±0.55	**89.24±0.49** ✔	52.17±11.12	82.56±6.34 ✔	70.64±26.97	81.97±12.06 ✔
	time (sec.)	735±35	238±35	222±42 ✔	1038±986	1032±1321 ✔	556±610	**197±208** ✔

gisette). Additionally, although our method's grouping strategy is primarily designed for classification problems, its simple transfer to regression problems can also yield improvements in accuracy and efficiency.

Compare with Method under Different Configurations:

The performance of bandit-based methods is greatly influenced by the number of configurations, which is one of the main motivations behind our work. When dealing with a large number of configurations, each configuration may only have a small number of instances assigned to it, which can result in poor optimization performance. Our method improves bandit-based methods in various aspects including subset sampling, cross-validation, and metric design, which is theoretically better than the vanilla method for multiple configurations. To verify this claim, we conducted experiments with "SHA" and "SHA$^+$" in the *australian* dataset from two perspectives: i) the number of hyperparameters and ii) model complexity.

Regarding the number of hyperparameters, we sequentially added new hyperparameters to the configuration space according to the order in Table III. As for model complexity, we selected the number of neurons per layer from $[10, 20, 30, 40, 50]$, and analyzed the performance of different methods by increasing the number of layers to observe the changes in model complexity. Figure 4 illustrates the accuracy and time cost variations of different methods with varying numbers of configurations. The increase in the number of configurations brings greater performance potential, which raises the upper limit of accuracy. This is the reason why both methods' accuracy improved when the number of settings initially increased from 1 to 4. However, as the number of configurations continued to increase, the instability of optimization led to fluctuations and even a decrease in accuracy. However, compared with the vanilla methods, our method exhibits superior accuracy as the number of configurations increases, especially for the increasing of layers. As for the search time, our method exceeds the vanilla method in terms of time efficiency, and this advantage becomes more noticeable as the number of settings increases. The experimental results indicate that our method performs well and efficiently under numerous hyperparameters and complex models.

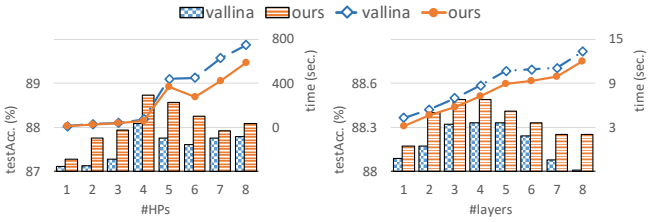

Fig. 4. Performance changes as the increase of HPs and model size.

C. Cross-Validation Experiment

In addition to optimizing hyperparameters, our approach can also be directly applied to k-fold cross-validation. To

better analyze the performance of the proposed method in the configuration evaluation, we conducted separate experiments on cross-validation. In this experiment, we improved the fold generation method and metrics in our proposed approach and compared them with several commonly used cross-validation methods, including random KFold and stratified KFold. The details about the baselines are shown below.

- random: the RandomKFold, which uses the method of random sampling to get different folds. We used the KFold implementation of the scikit-learn library.
- stratified: the StratifiedKFold, in which different folds are obtained by uniform sampling according to the distribution of labels, which is implemented in scikit-learn library.
- ours: the implementation of the proposed method on k-fold cross-validation. The folds are constructed using a group-based sampling method, which constructs general and special folds, and the quality of configurations is evaluated based on variance and sampling information.

Due to the time-consuming nature of training all models on the dataset to assess the quality of the recommended results, we limited our experimentation to modifying two hyperparameters: hidden layer sizes (in [(30), (30,30), (40), (40,40), (50), (50,50)]) and activation (in ['logistic', 'tanh', 'relu']]). There are a total of $6 \times 3 = 18$ configurations. We maintained the same 5-fold cross-validation setup as in the previous experiment, while keeping the other training parameters at their default values. In the experiment, we performed cross-validation on each configuration with different proportion subsets to obtain validation scores. Based on the validation scores, we recommended a configuration and calculated its actual accuracy. At the same time, we ranked the configurations based on the evaluation scores and compared the predicted ranking with the actual ranking to analyze their evaluation ability.

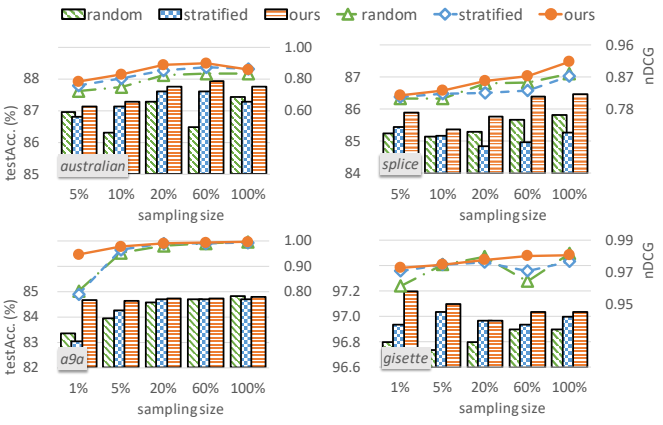

Fig. 5. Test accuracy (%) and nDCG score under different subset sizes for our method and other k-fold cross-validation methods.

From the experimental results shown in Figure 5, it can be observed that our method is capable of recommending configurations with better test accuracy across all six datasets. Moreover, the increased nDCG scores suggest that our enhancement is not restricted to suggesting a single superior configuration, but instead, it enables us to more effectively assess the ranking of various configurations. This enhanced evaluation capability expands the potential applications of our method, which can help the comparison and ranking for different configurations. In addition, our method outperforms other methods significantly in small subset sizes, which highlights the importance of variance in evaluating performance when the subset is small. Meanwhile, although different methods yield similar results for large subsets, our method shows significant advantages on some datasets (e.g., *splice* and *a9a*), which are due to the fine construction of the folds.

D. Independent Experiments

In addition to the two overall experiments, several independent experiments were conducted to verify the characteristics of the main designs in our method. In the experiment, we made adjustments to the three main parts of the proposed method, including instance grouping, fold construction, and metric design. In this section, we provide a detailed explanation of the experimental details and results.

Feature and Label based Instance Grouping:

First, we examined the impact of group construction in our method. We employed the same training settings as Section IV-C, which used cross-validation to make a direct comparison. To highlight the impact of grouping, both the vanilla and our method use stratified sampling in this experiment. The vanilla method divides the data equally based on the labels, while our method divides the data based on the groups. Meanwhile, the two methods both use mean accuracy as the evaluation metric. To observe the performance difference between the two methods on the subset size, we conducted experiments under two sampling ratios of 10% and 100%.

The experimental results are presented in Table V. Although the improvements in accuracy and nDCG achieved by our proposed methods are not significant, it can be observed that without considering the design of metrics and folds, the fine-grained grouping method proposed in this paper can still improve the ranking and recommendation effects. Besides, the proposed methods generally have smaller variances in most cases, which also confirms the advantage of our method in terms of stability. By comparing the results obtained under different subset sizes, it is apparent that the benefits of groups become more prominent when the subset size is small, which is consistent with the characteristics of the complete method.

General and Special in Fold Construction:

Our method not only incorporates group construction but also considers two types of folds: the general fold and the special fold. Traditional methods only consider general folds that conform to the overall distribution, neglecting the potential benefits of special folds on a limited budget. To analyze the impact of these two types of folds, we conducted experiments with different allocations for the two types of folds while ensuring that the total number of folds remained at 5.

Figure 6 illustrates the results of different fold allocations in the 5-fold cross-validation experiment. Through experimental results, it can be observed that cross-validation with all general

TABLE V
TEST ACCURACY (%) AND NDCG SCORE FOR CROSS-VALIDATION WITH OUR PROPOSED BLOCKING METHOD AND VANILLA STRATIFIED METHOD.

data	ratio	method	testAcc. (%)	nDCG	data	ratio	method	testAcc. (%)	nDCG	data	ratio	method	testAcc. (%)	nDCG
australian	10%	vanilla	85.02±0.49	0.786	*splice*	10%	vanilla	85.16±1.31	0.809	*a9a*	10%	vanilla	84.65±0.09	0.985
		ours	**85.83±0.28**	**0.845**			ours	**85.39±0.40**	**0.818**			ours	**84.70±0.08**	**0.989**
	100%	vanilla	85.18±0.56	0.764		100%	vanilla	85.27±1.15	0.870		100%	vanilla	84.70±0.87	0.992
		ours	**85.51±0.00**	**0.811**			ours	**86.05±0.50**	**0.874**			ours	**84.70±0.16**	**0.992**
gisette	10%	vanilla	96.73±0.23	0.975	*satimage*	10%	vanilla	88.49±0.69	0.951	*usps*	10%	vanilla	93.37±0.26	0.803
		ours	**96.87±0.25**	**0.980**			ours	**88.73±0.24**	**0.962**			ours	**93.49±0.37**	**0.834**
	100%	vanilla	96.90±0.17	0.976		100%	vanilla	88.88±0.20	0.966		100%	vanilla	93.42±0.28	0.869
		ours	**97.03±0.12**	**0.988**			ours	**88.95±0.52**	**0.974**			ours	**93.42±0.18**	**0.874**

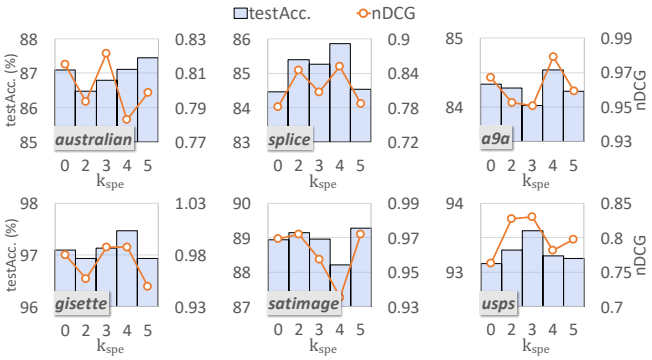

Fig. 6. Test accuracy (%) and nDCG scores with different allocation of folds.

folds or all special folds usually yields similar results in terms of test accuracy and nDCG score. This phenomenon suggests that the average performance of each subset with special folds is comparable to that of each subset with general folds, which is an interesting aspect of multi-subset evaluation. Furthermore, results from multiple datasets indicate that cross-validation with a mixture of both types of folds has better evaluation capabilities compared to validation with a single type of folds, as demonstrated in datasets such as *splice*, *usps*, and *gisette*. This is also the reason why we introduce special folds in the evaluation with limited resources. However, the effectiveness of mixed folds validation is not consistently stable in other datasets, but with appropriate settings, it can still lead to significant improvements.

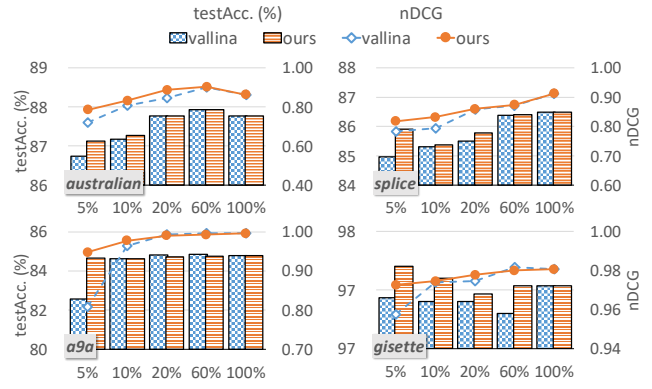

Fig. 7. Test accuracy (%) and nDCG score for cross-validation with vanilla metrics and our metrics in subsets with different sizes.

Variance and Sampling in Metric Design:

As one of the main designs in our work, we have considered both variance and sampling size information in the evaluation metric. In this section, we compared the differences in evaluation performance between the improved metric and the original metric through experiments. Similar to other independent experiments, we conducted the cross-validation experiment on different datasets. We maintained the instance grouping and fold construction unchanged and only altered the selection of the metric to evaluate the effectiveness of cross-validation under different subset sizes. Figure 7 shows the results of the experiment in terms of test accuracy and nDCG score. The experimental results clearly demonstrate the advantages of the metric proposed in this paper. When the sampling size is small, considering the variance and the metric of sampling, both the test accuracy and nDCG score can be higher than the vanilla method on all datasets.

V. CONCLUSION

Bandit-based optimization allocates budgets for different configurations to evaluate and select high-quality hyperparameters with limited resources, which has been widely applied in academia and industry. However, sampling randomness has a notable effect on bandit-based optimization, resulting in unstable training, poor performance, and high time consumption, which affects the performance of bandit-based optimization especially when handling a large number of configurations with high-dimensional and large problems. To address these issues, we have proposed a new bandit-based method based on sampling and variance information. The method effectively addresses the above-mentioned shortcomings by using more careful subset sampling, considering both general and special folds, and designing metrics that consider sampling and variance information. The proposed method achieves significant improvements in both optimization performance and efficiency, by accuracy improvement of 1% to 10% while reducing execution time by half.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No. 62376099 & 62072186) and the Guangzhou Municipal Science and Technology Project (No. 2023A03J0143).

References

[1] Kaggle: Your machine learning and data science community, Year Published.

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

[3] Davide Anguita, Luca Ghelardoni, Alessandro Ghio, Luca Oneto, Sandro Ridella, et al. The'k'in k-fold cross validation. In *ESANN*, pages 441–446, 2012.

[4] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. 2010.

[5] Noor Awad, Neeratyoy Mallik, and Frank Hutter. Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*, pages 2147–2153. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[6] Antoni Baum. hpbandster-sklearn. https://github.com/Yard1/hpbandster-sklearn.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 24, 2011.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 13(2), 2012.

[9] Daniel Berrar. Cross-validation, 2019.

[10] Ondrej Bohdal, Lukas Balles, Beyza Ermis, Cédric Archambeau, and Giovanni Zappella. Pasha: Efficient hpo and nas with progressive resource allocation, 2023.

[11] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[12] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[13] Burakh. (2016). Ground State Energies of 16,242 Molecules, Version 1. Retrieved October 25, 2023 from https://www.kaggle.com/datasets/burakhmmtgl/energy-molecule.

[14] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–27, 2011.

[15] Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. Easy hyperparameter search using optunity. *arXiv preprint arXiv:1412.1114*, 2014.

[16] Radwa ElShawi, Hudson Lekunze, and Sherif Sakr. csmartml: A meta learning-based framework for automated selection and hyperparameter tuning for clustering. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1119–1126. IEEE, 2021.

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

[18] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *The Journal of Machine Learning Research (JMLR)*, 23(1):11936–11996, 2022.

[19] Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

[20] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1487–1495, 2017.

[21] Harlfoxem. (2016). House Sales in King County, USA, Version 1. Retrieved October 25, 2023 from https://www.kaggle.com/datasets/harlfoxem/housesalesprediction.

[22] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: Data mining, inference, and prediction*, volume 2. Springer, 2009.

[23] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade*, pages 599–619. Springer, 2012.

[24] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI / PAMI)*, 44(9):5149–5169, 2021.

[25] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

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

[27] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics (AISTATS)*, pages 240–248. PMLR, 2016.

[28] Kihome. (2023, October). machine-predictive-maintenance-classification, Version 1. Retrieved October 25, 2023 from https://www.kaggle.com/code/kihome/machine-predictive-maintenance-classification.

[29] PM Lerman. Fitting segmented regression models by grid search. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(1):77–84, 1980.

[30] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *arXiv preprint arXiv:1810.05934*, 5, 2018.

[31] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research (JMLR)*, 18(1):6765–6816, 2017.

[32] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Zhi Yang, Ce Zhang, and Bin Cui. Transbo: Hyperparameter optimization via two-phase transfer learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 956–966, 2022.

[33] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[34] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *The Journal of Machine Learning Research (JMLR)*, 23(54):1–9, 2022.

[35] Mohamed Mohamed Maher Zenhom Abdelrahman Maher and Sherif Sakr. Smartmml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In *22nd International Conference on Extending Database Technology (EDBT)*, 2019.

[36] Kolby Nottingham Markelle Kelly, Rachel Longjohn. The uci machine learning repository. https://archive.ics.uci.edu.

[37] Akshay Mathur. NATICUSdroid (Android Permissions) Dataset. UCI Machine Learning Repository, 2022. DOI: https://doi.org/10.24432/C5FS64.

[38] Yinfeng Meng, Jiye Liang, Fuyuan Cao, and Yijun He. A new distance with derivative information for functional k-means clustering algorithm. *Information Sciences*, 463:166–185, 2018.

[39] Elgiriyewithana Nidula. (2023, October). Credit Card Fraud Detection Dataset 2023, Version 1. Retrieved October 25, 2023 from https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023.

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research (JMLR)*, 12:2825–2830, 2011.

[41] Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[42] Danny Matthew Saputra, Daniel Saputra, and Liniyanti D Oswari. Effect of distance metrics in determining k-value in k-means clustering using elbow and silhouette method. In *Sriwijaya International Conference on*

Information Technology and Its Applications (SICONIAN 2019), pages 341–346. Atlantis Press, 2020.

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

[44] Machine Learning Group ULB. (2017). Credit Card Fraud Detection, Version 3. Retrieved October 25, 2023 from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/.

[45] Chunnan Wang, Hongzhi Wang, Tianyu Mu, Jianzhong Li, and Hong Gao. Auto-model: utilizing research papers and hpo techniques to deal with the cash problem. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1906–1909. IEEE, 2020.

[46] Ananto Setyo Wicaksono and Ahmad Afif Supianto. Hyper parameter optimization using genetic algorithm on machine learning methods for online news popularity prediction. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(12), 2018.

[47] Xue Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, 2019.

[48] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-performance Computing Environments (MLHPC)*, pages 1–5, 2015.

[49] Yongqi Zhang, Quanming Yao, Wenyuan Dai, and Lei Chen. Autosf: Searching scoring functions for knowledge graph embedding. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 433–444. IEEE, 2020.

[50] Jihua Zhu, Zutao Jiang, Georgios D Evangelidis, Changqing Zhang, Shanmin Pang, and Zhongyu Li. Efficient registration of multi-view point sets by k-means clustering. *Information Sciences*, 488:205–218, 2019.