Modeling All Response Surfaces in One for Conditional Search Spaces

Jiaxing Li[1,2*], Wei Liu[2], Chao Xue[2], Yibing Zhan[2†], Xiaoxing Wang[3], Weifeng Liu[1], Dacheng Tao[4]

[1]China University of Petroleum (East China)
[2]JD Explore Academy
[3]Shanghai Jiao Tong University
[4]Nanyang Technological University
xinghui0606@gmail.com, lwwd1996@163.com, xuechao19@jd.com, zybjy@mail.ustc.edu.cn, figure1_wxx@sjtu.edu.cn, liuwf@upc.edu.cn, dacheng.tao@gmail.com

Abstract

Bayesian Optimization (BO) is a sample-efficient black-box optimizer commonly used in search spaces where hyperparameters are independent. However, in many practical AutoML scenarios, there will be dependencies among hyperparameters, forming a conditional search space, which can be partitioned into structurally distinct subspaces. The structure and dimensionality of hyperparameter configurations vary across these subspaces, challenging the application of BO. Some previous BO works have proposed solutions to develop multiple Gaussian Process models in these subspaces. However, these approaches tend to be inefficient as they require a substantial number of observations to guarantee each GP's performance and cannot capture relationships between hyperparameters across different subspaces. To address these issues, this paper proposes a novel approach to model the response surfaces of all subspaces in one, which can model the relationships between hyperparameters elegantly via a self-attention mechanism. Concretely, we design a structure-aware hyperparameter embedding to preserve the structural information. Then, we introduce an attention-based deep feature extractor, capable of projecting configurations with different structures from various subspaces into a unified feature space, where the response surfaces can be formulated using a single standard Gaussian Process. The empirical results on a simulation function, various real-world tasks, and HPO-B benchmark demonstrate that our proposed approach improves the efficacy and efficiency of BO within conditional search spaces.

Introduction

Bayesian Optimization (BO) (Shahriari et al. 2016; Garnett 2023; Papenmeier, Nardi, and Poloczek 2022, 2023) is a powerful and efficient global optimizer for expensive black-box functions, which has gained increasing attention in AutoML systems and achieved great success in a number of practical application fields in recent years (Martinez-Cantin et al. 2007; González et al. 2015; Calandra et al. 2016; Roussel et al. 2024). Considering a black-box function f: $\chi \to \mathbb{R}$ defined on a search space χ, BO aims to find the global optimal configuration $x_* = \arg\min_{x \in \chi} f(x)$. The sequential Bayesian optimization procedure contains two key steps (Shahriari et al. 2016): (1) BO seeks a probabilistic surrogate model to capture the distribution of the black-box f given n noisy observations $y_i = f(x_i) + \epsilon, i \subset 1, ..., n, \epsilon \sim \mathcal{N}(0, \sigma)$. (2) Suggest the next query x_{n+1} by maximizing an exploit-explore trade-off acquisition function $\alpha(x)$. The most common choice of the surrogate models is Gaussian Process (GP) (Snoek, Larochelle, and Adams 2012; Seeger 2004) due to its generality and good uncertainty estimation. As to acquisition functions, the common choice for GP-based BO is the Expected Improvement (EI) (Mockus 1994; Ament et al. 2023).

In the traditional BO setting, the search space χ is flat where all configurations have the same dimensions and structure [1]: $x \in \chi \subset \mathbb{R}^d$, where d is the number of dimensions. However, in many practical AutoML scenarios, such as Combined Algorithm Selection and Hyperparameter optimization (CASH) (Swersky et al. 2014) and Neural Architecture Search (NAS) (Thornton et al. 2013; Tan et al. 2019), the search spaces are conditional where the configurations have different structures and number of dimension.

Such a conditional search space χ can be decomposed into a series of flat subspaces (Jenatton et al. 2017): $\chi = \chi^1 \cup \chi^2 \cup ... \cup \chi^n$ and configurations in the same subspace have the same structure: $x^i \in \chi^i \subset \mathbb{R}^{d^i}$. We give a typical CASH example in Fig. 1. The space consists of two machine-learning algorithms and their specific hyperparameters, which can be partitioned into 6 subspaces. Due to the unaligned features among configurations within different subspaces, the surrogate models commonly used, such as GP, are not directly applicable.

A straightforward strategy to solve this issue is to build separate surrogate models for each subspace independently (Levesque et al. 2017). Given the lack of information sharing during the training phase of the GPs, most of these approaches suffer from the requirement of a significant number of observations to ensure the predictive performance of

*This work was done while the author was an intern at JD Explore Academy.
†Corresponding Authors.

[1]The structure of a configuration in this paper is defined as containing two aspects: dependencies between pairs of hyperparameters and semantic information of each hyperparameter. Therefore, the hyperparameter "learning rate" has the same semantic information in XGBoost and DNN models.

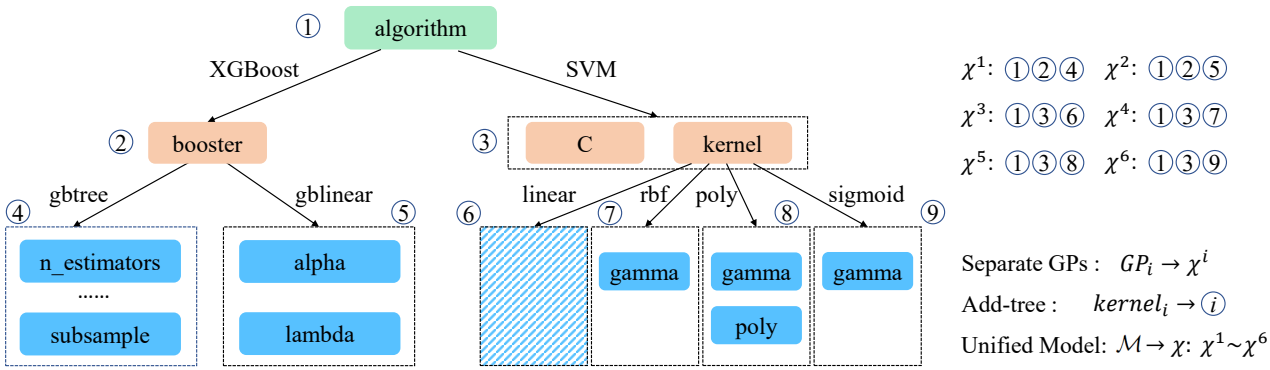


Figure 1: An example of the tree-structured search space for a CASH task. The space contains two popular algorithms and their distinct hyperparameters. According to the dependencies among hyperparameters, the search space can be formed as 9 nodes and partitioned into 6 flat subspaces. The approaches using separate GPs build a model GP_i for each subspace and conduct optimization in each subspace. Add-tree (Ma and Blaschko 2020a) builds a kernel on each node and integrates them using the additive assumption. Our method explores to build a unified surrogate model $\mathcal{M}$ for all subspaces $\chi^1 \sim \chi^6$.

each GP (Ma and Blaschko 2020b). Recent work (Ma and Blaschko 2020a) proposed to set a covariance function on each node of the tree-structured space and integrate them in a GP through an additive assumption. However, the model is still essentially divided, with the parameters not shared among different nodes. As a consequence, the relationships between non-shared but semantically related hyperparameters, such as "gamma" of different kernels (in nodes 7-9 of Fig. 1), are totally ignored.

To overcome these limitations, we explore building a unified surrogate model for all hyperparameters within the tree-structured search space, by which we can capture the relationships among all hyperparameters and utilize the information of all observations even from different subspaces. Concretely, to obtain a better representation of the hyperparameter's features, we propose a hyperparameter embedding that preserves the structural information of the subspace each hyperparameter belongs to. In our framework, we treat the embedding of each hyperparameter as a token and each hyperparameter configuration as a sequence of tokens. Then, we introduce an attention-based deep encoder, which is capable of modeling the global relationships among tokens using self-attention blocks and projecting the sequences from different subspaces into a unified latent space by an average pooling operator. With the attention-based encoder, the features of these configurations become comparable and can be modeled by any standard kernel function. Different from the previous works, our approach considers the relationships among hyperparameters in different subspaces, and the parameters of the surrogate model are shared and completely consistent for all observations, which can improve the sample efficiency of BO in the tree-structured search spaces.

Some recent works have proposed using variational autoencoders (VAEs) (Kingma and Welling 2014) to transform high-dimensional and structured inputs into continuous, low-dimensional spaces that are more amenable to Bayesian optimization techniques (Kusner, Paige, and Hernández-Lobato 2017; Lu et al. 2018; Tripp, Daxberger, and Hernández-Lobato 2020; Grosnit et al. 2021; Maus et al. 2022). Structured or combinatorial inputs (Deshwal and Doppa 2021) refer to sequences, trees, or graphs organized by categorical variables. In contrast, the conditional search space we address has no such restrictions on variable types involving both numerical and categorical variables. Consequently, these existing methods do not readily extend to search spaces containing both categorical and numerical hyperparameters in a complex, structured relationship, and we don't consider these methods as our competitors.

In conclusion, our contributions can be summarized as follows:

1) We explore the relationships among all hyperparameters and integrate information from all observations with higher sample efficiency via a unified surrogate model.

2) We propose a novel attention-based Bayesian optimization framework, consisting of two key components: a hyperparameter embedding to preserve the structural feature of each hyperparameter in a configuration and an attention-based deep kernel Gaussian process to model the response surfaces of all subspaces in one. In our framework, all parameters are shared for any configurations, making it consider the relationships among all hyperparameters.

3) We conduct experiments on a standard tree-structured simulation function, a Neural Architecture Search (NAS) task, and several real-world OpenML tasks to demonstrate the efficiency and efficacy of our proposed approach. Besides, to validate the warm-starting capability of our method in scenarios with extensive historical data, we also conduct a meta-learning experiment on the HPO-B benchmark.

BO for Conditional Search Space

Sequential Model-based Algorithm Configuration (SMAC) is an early BO method that can deal with conditional search spaces (Hutter, Hoos, and Leyton-Brown 2011). SMAC involves utilizing all potentially active hyperparameters across the entire space as inputs to the surrogate model. Before being fed into the surrogate model, inactive hyperparameters

in configurations from different subspaces will be filled with default values to minimize their interference with the surrogate model. This approach will introduce redundant codes, resulting in higher-dimensional problems and diminishing the efficiency of fitting the surrogate model.

Compared to SMAC, GP-based BO gives better uncertainty estimation and shows higher sample efficiency in practical applications. The most straightforward way to leverage GP in the context of conditional search spaces is to build separate GPs in these subspaces (Levesque et al. 2017; Nguyen et al. 2020). Due to the lack of an information-sharing mechanism during the training stage, these approaches require a large number of observations in each subspace to fit these models, making it impractical when the number of subspaces becomes too large (Jenatton et al. 2017; Ma and Blaschko 2020b). Jenatton et al. (2017) proposed a semi-parametric GP method that captures the relationship of GPs via a weight vector. However, the linear relationship assumption among these GPs makes it less flexible. Following this work, Ma and Blaschko (2020b) proposed an Add-Tree covariance function to capture the response surfaces. It sets a covariance function on each node of the tree-structured space and integrates them using the additive assumption. However, it totally ignores the relationships between non-shared hyperparameters.

Preliminaries

Deep Kernel Learning for Gaussian Process The standard GPs rely on a suitable handcrafted kernel function. An inappropriate kernel will lead to sub-optimal performances due to the false assumptions (Cowen-Rivers et al. 2022). The idea of deep kernel learning (Wilson et al. 2016) is to introduce a neural network ϕ to transform the configuration x to a latent representation that serves as the input of the kernel, which facilitates learning the kernel in a suitable space. Concretely, the kernel function is shown as:

$$k_{deep}(x, x^{'}|\theta, \omega) = k(\phi(x, \omega), \phi(x^{'}, \omega)|\theta), \quad (1)$$

where ω represents the weights of the deep neural network ϕ and θ represents the parameters of the kernel function. All these parameters can be jointly estimated by maximizing the marginal likelihood (Wistuba and Grabocka 2021):

$$\log p(\mathbf{y}|\mathbf{X}, \theta, \omega) \propto -(\mathbf{y}^T \mathbf{K}_{deep}^{-1} \mathbf{y} + \log(|\mathbf{K}_{deep}|)), \quad (2)$$

where $\mathbf{X}$ and $\mathbf{y}$ represent the configurations and their noisy response, respectively, and the deep kernel matrix $\mathbf{K}_{deep}$ is equal to $k_{deep}(\mathbf{X}, \mathbf{X}|\theta, \omega) + \sigma^2 \mathbf{I}$ (Seeger 2004).

Self-Attention Mechanism In the field of Natural Language Processing (NLP), the transformer model (Vaswani et al. 2017) is a pioneering work, which utilizes the self-attention mechanism to model the global relationship between different words in a sequence, such as a sentence and paragraphs. In many later practices and papers, the effectiveness of the attention module was verified and applied in many fields (Lin et al. 2020, 2022). For an input sequence of N words, the d_k-dimensional embeddings of words plus the corresponding positional embeddings are fed into a stacked attention module. In the attention mechanism, the packed matrix representation of the query $Q \in \mathbb{R}^{N \times d_k}$, the key $K \in \mathbb{R}^{N \times d_k}$ and the value $V \in \mathbb{R}^{N \times d_k}$ are fused through $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} = \mathbf{A}\mathbf{V}$. The attention matrix A contains the similarity between each pair of words, which makes the output feature of a word a fusion of the feature of each word in the whole sequence. In this paper, this mechanism is employed to model the relationship among hyperparameters in a tree-structured search space, where the sampled configuration can be viewed as a sequence of hyperparameters.

Methodology

In this section, we present an attention-based Bayesian optimization framework (AttnBO) to model the relationships among all hyperparameters and build a unified response surface for all subspaces. First, we give the modeling of the conditional and tree-structured search space, and we analyze the differences between the unified surrogate model and multiple independent surrogate models. Then we provide the methodological details for each component of our approach.

Problem Formulation

A conditional search space χ can be decomposed into a series of flat subspaces (Jenatton et al. 2017; Ma and Blaschko 2020b): $\chi = \chi^1 \cup \chi^2 \cup ... \cup \chi^n$ and configurations in the same subspace have the same structure: $x^i \in \chi^i \subset \mathbb{R}^{d^i}$, d^i means the number of dimensions of the subspace χ^i.

Here, we give a loose assumption that such a conditional structure is a tree structure or a combination of several tree structures. To avoid multiple repetitions and simplify the notation, we use a single tree structure that refers to multiple trees since the embedding method and our method can handle both cases in a unified way. Inspired by Ma and Blaschko (2020b), we define the search space χ as a tree structure $\mathcal{T} = (V, E)$, where one node $v \in V$ refers to one set of hyperparameters, each with the associated range, type and value (if sampled), and $e \in E$ refers to the dependency relationship between a node v and the father node $v \uparrow \in V$, which is caused by the hyperparameter $p \in v$ and the dependent hyperparameter $p \uparrow \in v \uparrow$. We assign a virtual father vertex with a fixed embedding for those root nodes, which can also be applied to the case of multiple trees. The ancestor nodes and intermediate nodes have at least one decision variable. Each subspace χ^i is a path from the root to a leaf.

After sampling from the search space χ, we group the configurations $\mathbf{X}$ and their noisy responses by each subspace χ^i as $\mathbf{X}^i = \left\{x_j^i\right\}$, $\mathbf{y}^i = \left\{y_j^i\right\}$ where $i = 1, 2, ..., n$ and $j = 1, 2, ..., N^i$, N^i represents the number of sampled points belonging to the subspace χ^i. We denote the observations as $D = \left\{D^1, D^2, ..., D^n\right\}$, where D^i means all observations $\left\{(x_j^i, y_j^i)|j = 1, 2, ..., N^i\right\}$ in subspace χ^i. A configuration x_j^i is a sequence of hyperparameters $\left[p_k^{x_j^i}\right]$ with specific values, where $k = 1, 2, ..., d^i$, d^i means the dimension (the number of hyperparameters) of the subspace χ^i.

Most previous works (Jenatton et al. 2017; Nguyen et al. 2020; Levesque et al. 2017) proposed to build a surrogate model M^i for each subspace χ^i independently. These surrogate models can only be trained on the observations D^i and predict the posterior distribution $P(f_*^i|\mathbf{X}^i, \mathbf{y}^i, x_*^i)$, where x_*^i represents a configuration from subspace χ^i and f_*^i represents the objective funtion value on x_*^i. In contrast, a unified surrogate model M can be trained on all observations D and infer posterior probabilities for configuration x_* within any subspace: $P(f_*|\mathbf{X}, \mathbf{y}, x_*)$, which implies it must handle the input configurations that vary in both dimension and semantics while ensuring the parameters are shared across these inputs. Compared with the former, a unified surrogate model offers the advantage of utilizing all observations' information simultaneously without additional sharing mechanisms. This enhances training efficiency and enables flexible inference of posterior probabilities for all configurations.

Structure-aware Embeddings

The hyperparameters in different subspaces may have different semantics and relationships, which the surrogate model should be aware of when modeling the tree-structured response surface. As Fig.2 shows, there are two sampled configurations (p_1, p_2, p_4) and (p_1, p_3, p_6) from different subspaces. Obviously, the hyperparameters in the two subspaces have different semantics and relationships, making the value of the hyperparameter not enough to represent its feature.

Therefore, we assign each hyperparameter an embedding that contains semantic and dependency information, instead of only considering the value of the hyperparameter as in previous BO methods (Hutter, Hoos, and Leyton-Brown 2011; Jenatton et al. 2017; Ma and Blaschko 2020b). From the data structure view of this tree structure, since each node has only one parent node or not, we can serialize the tree by storing nodes and their corresponding father nodes. Thus, as Fig. 2 shows, we can encode a hyperparameter $p_k^{x_j^i}$ to a structure-aware embedding with four elements:

1) The identity embedding $id\_emb(p_k^{x_j^i})$. It works as an identifier and represents the semantic information of a hyperparameter. First, we adopt ordinal encoding to encode the identities of hyperparameters into numerical codes (1, 2, ...), represented by $id(p_k^{x_j^i})$. Similar to introducing vectorial meta features (Springenberg et al. 2016; Perrone et al. 2018), we use a trainable map to get the final identity embedding: $id\_emb(p_k^{x_j^i}) = \phi_{id}(id(p_k^{x_j^i}))$.

2) The index embedding $idx\_emb(p_k^{x_j^i})$. In some NAS problems, some hyperparameters, such as the number of hidden units in a multi-layer deep neural network, may be represented as vectors, the dimension of which depends on the number of layers. In this context, each element of this hyperparameter also needs an identifier. We use the index of each element in the vector $idx(p_k^{x_j^i})$ to identify it and a trainable map to get the final index embedding: $idx\_emb(p_k^{x_j^i}) = \phi_{idx}(idx(p_k^{x_j^i}))$. For the hyperparameter $p_k^{x_j^i}$ containing only a scalar, we set $idx(p_k^{x_j^i}) = 0$.

3) The value embedding $value\_emb(p_k^{x_j^i})$. Besides structural information, the value of a hyperparameter is a crucial feature. We apply a trainable linear map ϕ_{value} to construct the value embedding.

4) The identity embedding of the father (dependent) hyperparameter $id\_emb(p_k^{x_j^i} \uparrow) = \phi_{id}(id(p_k^{x_j^i} \uparrow))$. It provides the identifier of a hyperparameter's ancestor, which helps preserve the structural information. For the hyperparameters in the root nodes, we set $id(p_k^{x_j^i} \uparrow) = 0$.

We concatenate these four embeddings as the representation of a hyperparameter $p_k^{x_j^i}$. We refer to all operations involved in obtaining this embedding collectively as the embedding block, denoted by $emb(p_k^{x_j^i})$:

$$emb(p_k^{x_j^i}) = concat(id\_emb(p_k^{x_j^i}), idx\_emb(p_k^{x_j^i}), value\_emb(p_k^{x_j^i}), id\_emb(p_k^{x_j^i} \uparrow)), k = 1, 2, ..., d^i, \quad (3)$$

where d^i represents the dimension of the flat subspace χ^i. Using such embeddings, we transform each hyperparameter into a vector representation that incorporates structural information, thereby enhancing the effectiveness and distinctiveness of the hyperparameter features. The full embedding of a configuration is represented as:

$$emb(x_j^i) = \left[emb(p_1^{x_j^i}), emb(p_2^{x_j^i}), ..., emb(p_{d^i}^{x_j^i})\right]. \quad (4)$$

To demonstrate the effectiveness of the structure-aware embedding, we conducted an ablation study on these embeddings, and the experimental results can be found in Fig. 8.

BO with Attention-based DKGP

Compared to other surrogate models (Hutter, Hoos, and Leyton-Brown 2011; Bergstra et al. 2011), Gaussian Processes (GPs) offer better uncertainty estimation and higher sample efficiency in practical applications. However, handling configurations that vary in length and contain different hyperparameters is still a challenge for GP. To address this issue, we introduce an attention-based encoder to adapt the GP for this context. This encoder works as a deep feature extractor within the deep kernel framework (Wilson et al. 2016), allowing us to capture global relationships among hyperparameters and project variable-length configurations into a unified latent space $\mathcal{Z} \subset \mathbb{R}^d$. A GP can then be built on this latent space $\mathcal{Z}$ using a standard kernel function.

Consider a black-box function with noisy observations $y_i = f(x_i) + \epsilon, i \subset 1, ..., n, \epsilon \sim \mathcal{N}(0, \sigma)$, we have a dataset D of N noisy observations in a conditional space χ that has n flat subspaces $\left\{\chi^1 \cup \chi^2 \cup ... \cup \chi^n\right\}$, $N = \sum_{i=1}^n N^i$, $D = \left\{D^1, D^2, ..., D^n\right\}$, where D^i means all observations $\left\{(x_j^i, y_j^i)|j = 1, 2, ..., N^i\right\}$ in subspace χ^i. We adopt the deep kernel learning framework (Wilson et al. 2016) to learn the weights of the embedding block, attention-based encoder, and the parameters of the kernel function jointly

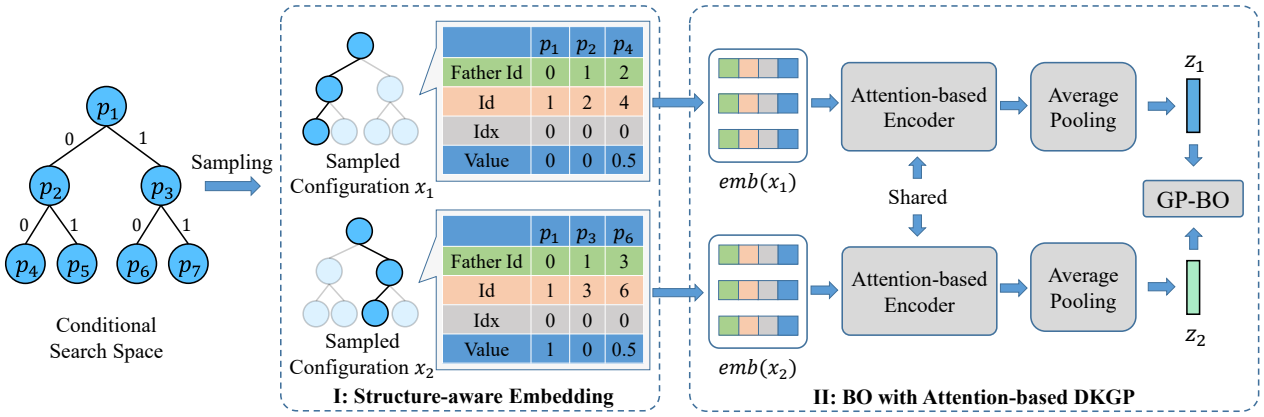


Figure 2: The framework of AttnBO. We introduce three elements—the identity, index, and father's identity of a hyperparameter, which preserve structural features for each hyperparameter. Each hyperparameter embedding can be considered a token, and a configuration can therefore be viewed as a sequence of tokens. Then we employ an attention-based encoder to capture the relationships of these tokens and project all sequences into a unified latent space where a GP-based BO can work directly.

by maximizing the log marginal likelihood in eq. 2. In our framework, the deep kernel matrix is as follows:

$$\begin{aligned}\mathbf{K}_{deep} &= k_{deep}(\mathbf{X}, \mathbf{X}|\theta, \omega) + \sigma^2\mathbf{I} \\ &= k(\phi(emb(\mathbf{X}, \omega_1), \omega_2), \\ &\quad \phi(emb(\mathbf{X}, \omega_1), \omega_2)|\theta) + \sigma^2\mathbf{I}, \end{aligned} \tag{5}$$

where ω_1, ω_2 are two subsets of ω representing the weights of the embeddings and the attention-based encoder.

With a well-fitted DKGP, the predictive posterior distribution of the objective function f at x_* is as follows:

$$f_*|\mathbf{X}, \mathbf{y}, x_* \sim \mathcal{N}\left(\overline{f}_*, var(f_*)\right), \tag{6}$$

$$\overline{f}_* = \mathbf{k}_{deep_*}^\top \mathbf{K}_{deep}^{-1}\mathbf{y}, \tag{7}$$

$$var(f_*) = k_{deep}(x_*, x_*) - \mathbf{k}_{deep_*}^\top \mathbf{K}_{deep}^{-1}\mathbf{k}_{deep_*}. \tag{8}$$

The deep kernel matrix $\mathbf{K}_{deep}$ can be found in eq. 5, and we write $\mathbf{k}_{deep_*} = k_{deep}(x_*, \mathbf{X})$ to donate the vector of covariances between the test point x_* and the training points $\mathbf{X}$. In this paper, we use the Matérn 5/2 kernel function to accommodate the DKGP model and adopt EI acquisition function to choose the next query. During the acquisition stage, we utilize lbfgs optimizer to optimize EI on each subspace and find the most valuable configurations to query in each subspace, enabling parallel Bayesian optimization on the objective functions. For those more complex search spaces that have a large number of subspaces, optimizing the acquisition function within each subspace is computationally expensive. In such cases, we can directly employ random search to optimize the acquisition function over the entire space to give batch queries. Under the sequential BO setting, we choose the configuration that maximizes the acquisition function among all subspaces. The detailed procedure of the algorithm is shown in Algorithm 1.

Experiments

Experimental Setting

Tasks. To demonstrate the efficiency and efficacy of AttnBO, we conduct experiments on multiple tasks: For the simulation task, we follow the setting of Ma and Blaschko (2020b), and the details of the search space can be found in our supplementary material. We adopt log_{10} distance between the best minimum achieved so far and the practical minimum value as the y-axis.

Following the solid work Tan et al. (2019) in the NAS field, we set an optimization problem in a complex search space which includes a minimum of 29 and a maximum of 47 hyperparameters depending on different conditions —- the number of the blocks ranging from 4 to 7. There are both categorical and continuous hyperparameters in this NAS space, and the candidate will be evaluated on CIFAR-10 dataset after 100 training epochs. The details of the settings of this search space can be found in the supplementary material.

For the OpenML tasks, we designed two conditional search spaces, one for SVM and one for XGBoost, both of which are widely used machine learning models for tabular data. Moreover, we also combine the two search spaces via an algorithm variable as Fig. 1 shows, leading to a CASH problem and making the search space more complex having six subspaces and 15 hyperparameters. The details of these search spaces are shown in our supplementary material. Supported by OpenML (Vanschoren et al. 2013), we consider 6 most evaluated datasets: [10101, 37, 9967, 9946, 10093, 3494].

Benefiting from the capability to handle configurations across various subspaces, our surrogate model enables large-scale meta-learning on multiple source tasks from various search spaces. We verify this feature on HPO-B-v3 (Pineda-Arango et al. 2021), a large-scale hyperparameter optimization benchmark that contains a collection of 935 black-box tasks for 16 hyperparameter search spaces evaluated on 101 datasets. We set the ID of a search space as the father node of its hyperparameters, resulting in a tree-structured search space. We meta-train our model on all training data points of the 16 search spaces and fine-tune it on the test tasks to get the final performance.

Algorithm 1: AttnBO: An Attention-based Bayesian Optimization Method for Conditional Search Spaces.

Inputs: A black-box function f defined on a conditional search space $\chi = \chi^1 \cup \chi^2 \cup ... \cup \chi^n$;
The batch size $B (B <= n)$;
The number of total training iterations T.

Randomly sample two initial points ($N^i = 2$) to evaluate from each subspace, resulting in $N = 2n$ initial points in total.

Get the initial dataset: $D_0 = \{(x_j^i, y_j^i) | i = 1, 2, ..., n, j = 1, 2, ..., N^i\}$

for $t := from\ 1\ to\ T$ **do**

  Fit the Deep Kernel Gaussian Process by maximizing the log marginal likelihood (eq. 2) with Adam optimizer.

  Optimize the acquisition function in each subspace: $x_*^i = \arg\max_{x \in \chi^i} \alpha(x), i = 1, 2, ..., n.$

  Get the next queries: $X_* = \{x_b | b = 1, 2, ..., B\} = TopB(\{\alpha(x_*^i) | i = 1, 2, ..., n\})$

  Query f and get new observations: $D_* = \{(x_b, y_b) | b = 1, 2, ..., B, y_b = f(x_b)\}$.

  Update dataset: $D_t = D_{t-1} \cup D_*$

end

Output: The best point x_{opt} in history.

‡: $TopB$ is a function that returns the top B configurations ranked by the acquisition function.

Baselines. We compare AttnBO with Random Search (Bergstra and Bengio 2012) and four BO baselines for the conditional space on all tasks, including two GP-based methods (Bandits-BO (Nguyen et al. 2020), AddTree (Ma and Blaschko 2020b,a)) and two non-GP methods (SMAC (Hutter, Hoos, and Leyton-Brown 2011), TPE (Bergstra et al. 2011)). Compared to the naive independent GPs, Bandits-BO is more advanced and can easily degrade to the latter. Therefore, we choose it as our baseline rather than the naive independent GPs. Moreover, we also compare with Bandits-BO under a parallel setting on real-world OpenML tasks to demonstrate our ability of batch optimization. The implementation details of these baselines can be found in our supplementary material.

Implementation details. We adopt the Transformer encoder as the deep kernel network to project the configurations in different subspaces into a unified latent space $\mathcal{Z}$. Specifically, we employ 6 attention blocks with 2 parallel attention heads. The dimensionality of input and output is dmodel = 256 (4×64), and the inner layer also has a dimensionality of 512. We adopt average pooling to integrate the output of the transformer encoder and utilize a multi-layer perceptron (MLP) with 4 hidden layers, which has [128, 128, 128, 32] units of each hidden layer, to project the features of the configurations into 32-dim vectors. The parameters of the encoder are selected based on their performance on the SVM and XGBoost tasks. We train the embedding layer and attention-based encoder jointly for 100 epochs using Adam (Kingma and Ba 2015). We set the initial learning rate to 0.001 and reduce it by half every 30 epochs. More details of our implementation can be found in our supplementary material.

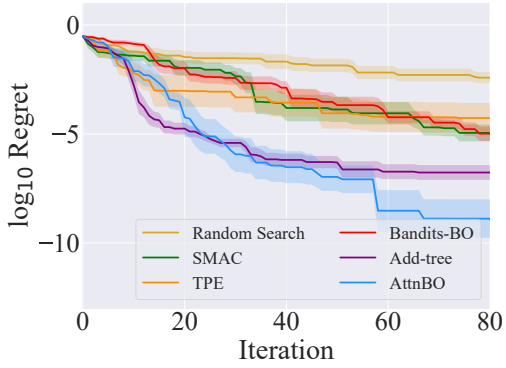


Figure 3: Performance of our AttnBO and baselines on the conditional simulation objective function.

Other settings. Following the settings of Bandits-BO, we give $2n$ random points to initialize BO methods. Then, we run BO on the simulation and OpenML tasks until 80 observations (without initial points) are collected and repeat the experiment 10 times to reduce the impact of random seeds. For the NAS tasks, we train each candidate on CIFAR-10 training set for 100 epochs and evaluate on the testing set. Because the evaluation of a configuration in this task is very expensive, we only repeat the experiment 3 times.

Experiment Analysis

Simulation Function. Following the setting of Ma and Blaschko (2020b), we compare our AttnBO with other baselines on this additive structure objective function. As shown in Fig. 3, our method performs best on this simulation function. Here, for a fair comparison, we reimplement the experiment and set the same random seeds as all other algorithms. (Probably, we did not get the same results as shown in their paper due to the different number of initial points.)

Neural Architecture Search. Considering the evaluation of a deep neural network is expensive, parallelization becomes especially important and necessary, which could improve the efficiency of the optimization process. However, the state-of-the-art method AddTree (Ma and Blaschko 2020b) doesn't support parallelization, which will still give one query per BO iteration in this experiment. We show the optimal accuracy after each BO iteration for all methods in Fig. 4.

With more hyperparameters and more complex condition settings, the ability to explore becomes crucial. Compared to other methods, Add-Tree is limited in its capability to explore or exploit various configurations within a BO loop due to the inability to provide batch queries. As a result,

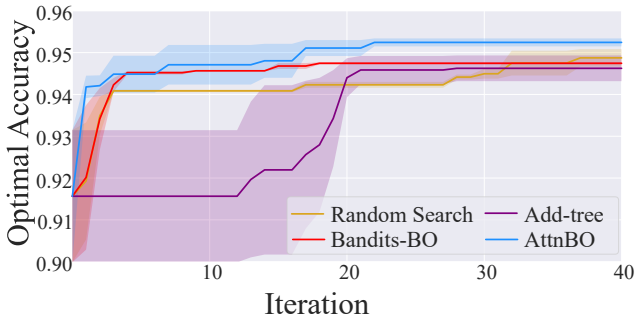


Figure 4: Performance of baselines and AttnBO on the complex NAS space.

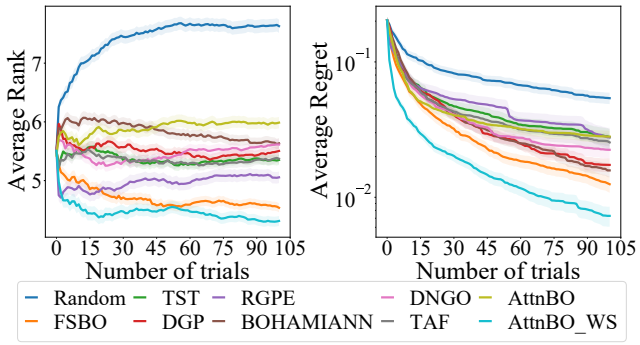


Figure 5: Performance of various black-box optimization methods on HPO-B benchmark.

Time Cost (minutes)		
AttnBO	*Bandits − BO*	*Add − tree*
17.05	11.96	12.73

Table 1: The total time cost over 100 iterations on the simulation task.

the opportunity for observation is reduced, leading to a failure in finding optimal configurations during the early stages. On the other hand, Random Search demonstrates better performance on this task because of its strong ability to explore across each dimension in a larger space with parallelism (Bergstra and Bengio 2012). In contrast to existing methods, our AttnBO has the advantage of exploiting the relationships between hyperparameters, which allows us to learn better representations of configurations. Additionally, AttnBO also enables parallel optimization by selecting the best candidate in each subspace, leading to higher efficiency throughout the BO process.

Real-world tasks on OpenML. Fig. 7 reports the average ranking of performance on three hierarchical search spaces of two machine-learning models, which were evaluated on 6 real-world datasets randomly selected from OpenML (Vanschoren et al. 2013; Feurer et al. 2021). Our proposed method achieves the best performance on all three tasks and, in particular, outperforms the start-of-the-art BO method AddTree (Ma and Blaschko 2020a) for conditional search spaces. We also report the performance of all baselines on each dataset, which can be found in our supplementary material.

HPO-B Benchmark. Fig. 5 displays the performance of our method compared to other default baselines as reported in (Pineda-Arango et al. 2021). We also compare AttnBO with and without warm-starting, denoted as AttnBO_WS and AttnBO, respectively. The results demonstrate the effectiveness of the warm-starting phase on large-scale datasets. Besides, with warm-starting on HPO-B, our proposed AttnBO can achieve competitive performance with state-of-the-art transfer learning methods.

Computational Cost Compared to other baselines, our method incurs slightly higher computational costs due to the training of the deep encoder. Table 1 summarizes the total time (in minutes) required by our method and two other BO methods to complete 100 iterations on the simulation task. Since the simulation function involves no evaluation cost, the recorded time is effectively equivalent to the algorithm's runtime. The results indicate that our method takes only about 5 minutes longer than the other two methods. For objective functions with significantly high evaluation costs, this additional overhead becomes negligible.

Sample Efficiency. Compared to separate GP models, the unified model exhibits higher sample efficiency. To illustrate the higher sample efficiency of the unified model compared to separate GPs, we conducted a comparison based on the simulation function. The results show that our method can achieve the same performance within 100 observation points that the separate models require 200 points to reach. Thus, our method will become more advantageous when the observation cost of the black box is exceedingly expensive.

Ablation Study

Structure-aware Embedding. In this experiment, we compare our method with two variants, AttnBO-no-emb and AttnBO-token-mixer, on two machine-learning tasks. On the

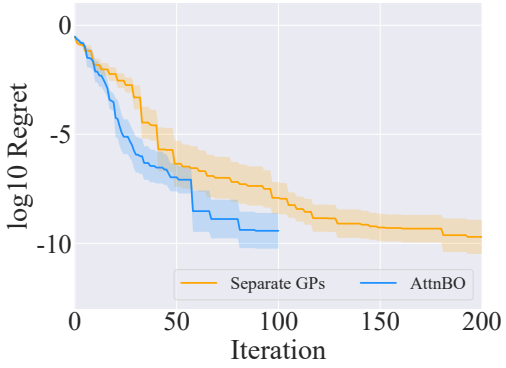
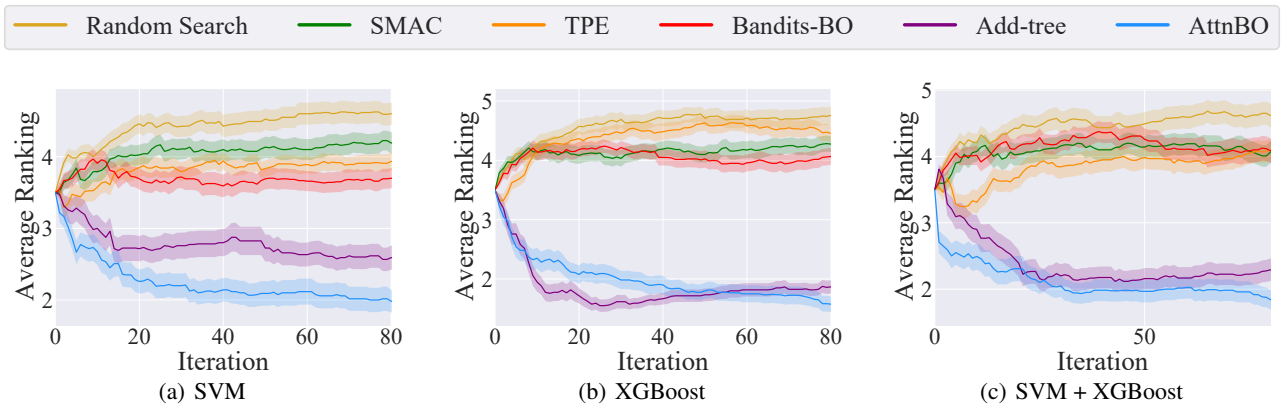
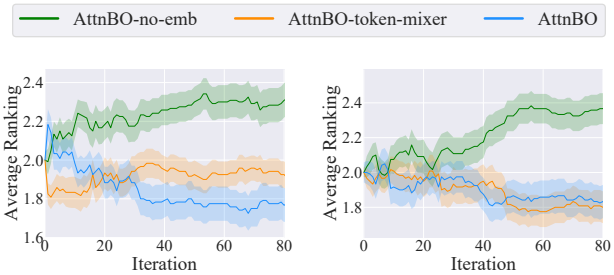


Figure 6: Comparison of AttnBO with the Separate GP Approach on the conditional simulation objective function.



(a) SVM (b) XGBoost (c) SVM + XGBoost

Figure 7: Average rankings of various methods on three machine-learning tasks evaluated on real-world OpenML datasets.



(a) SVM (b) XGBoost

Figure 8: Performance of our AttnBO and two variants on the SVM and XGBoost task.

one hand, to verify the effectiveness of the embedding we proposed, we drop the structure-aware embedding and only use the values of the hyperparameters as their features. We denote this method as AttnBO-no-emb. On the other hand, instead of using the average pooling operator for feature fusion, we introduce an additional token to represent the global feature of the sequence, which is often used in the Computer Vision field (Dosovitskiy et al. 2021) and denoted as AttnBO-token-mixer. Fig. 8 shows the average ranking of these methods in each search space. Obviously, our proposed embedding method helps to capture the relationships between hyperparameters and leads to higher effectiveness.

Architecture of the Attention-based Encoder. This experiment is to explore the architecture of the attention-based encoder. The number of the self-attention blocks and parallel heads are denoted as n_a and n_b. Table 2 shows all candidates' average regret and ranking on these two tasks. The results show that changes in the network structure had minimal impact on the performance, with the final average rankings of different structures being very similar. This indicates that our method is not sensitive to slight variations in network architecture. We selected the configuration with the lowest average ranking across the two tasks ($n_a = 6, n_b = 2$, with MLP) as our default network structure.

Architecture			SVM		XGBoost	
n_a	n_b	MLP	accuracy	ranking	accuracy	ranking
3	8	×	0.84±0.05	4.73±0.24	0.88±0.04	4.71±0.28
3	8	✓	0.85±0.05	4.81±0.25	0.89±0.04	5.12±0.27
4	6	×	0.84±0.05	4.80±0.25	0.89±0.04	4.21±0.28
4	6	✓	0.85±0.05	4.41±0.26	0.89±0.04	4.68±0.24
5	4	×	0.85±0.05	4.45±0.24	0.90±0.04	3.99±0.28
5	4	✓	0.85±0.05	4.07±0.25	0.89±0.04	4.48±0.29
6	2	×	0.85±0.05	4.53±0.25	0.88±0.04	4.72±0.28
6	2	✓	0.84±0.05	4.20±0.22	0.90±0.04	4.10±0.28

Table 2: The average accuracy and ranking of candidate network architectures on the SVM and XGBoost tasks. The signs "×" and "✓" represent the architecture with and without a MLP after the attention block, respectively.

Conclusion

In this paper, we explore how to model the response surfaces within conditional search spaces in one for efficient optimization in these spaces. We proposed a novel attention-based BO framework AttnBO. Concretely, we proposed a hyperparameter embedding method that can introduce the semantic and dependency information into the feature of each hyperparameter. Then we utilize an attention-based encoder to model the relationships among hyperparameters and project the configurations from different subspaces into a unified latent space. With the powerful attention-based encoder, we build a single standard GP model in a united latent space. Moreover, our proposed method can give a batch of queries in a BO iteration, which improves efficiency when dealing with expensive objective functions. We conduct the experiments on multiple tasks and give sufficient experimental results to demonstrate the effectiveness of our method.

Acknowledgments

This work was partially supported by the Major Science and Technology Innovation 2030 "New Generation Artificial Intelligence" key project (No. 2021ZD0111700). The authors are grateful to the anonymous reviewers for their insightful comments and careful proofreading.

References

Ament, S.; Daulton, S.; Eriksson, D.; Balandat, M.; and Bakshy, E. 2023. Unexpected improvements to expected improvement for bayesian optimization. In *NeurIPS*.

Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for Hyper-Parameter Optimization. In *NeurIPS*.

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.*

Calandra, R.; Seyfarth, A.; Peters, J.; and Deisenroth, M. P. 2016. Bayesian optimization for learning gaits under uncertainty - An experimental comparison on a dynamic bipedal walker. *Ann Math Artif Intell*.

Cowen-Rivers, A. I.; Lyu, W.; Tutunov, R.; Wang, Z.; Grosnit, A.; Griffiths, R.; Maraval, A. M.; Hao, J.; Wang, J.; Peters, J.; and Bou-Ammar, H. 2022. HEBO: An Empirical Study of Assumptions in Bayesian Optimisation. *J. Artif. Intell. Res.*

Deshwal, A.; and Doppa, J. R. 2021. Combining Latent Space and Structured Kernels for Bayesian Optimization over Combinatorial Spaces. In *NeurIPS*.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

Feurer, M.; van Rijn, J. N.; Kadra, A.; Gijsbers, P.; Mallik, N.; Ravi, S.; Müller, A.; Vanschoren, J.; and Hutter, F. 2021. OpenML-Python: an extensible Python API for OpenML. *Journal of Machine Learning Research*.

Garnett, R. 2023. *Bayesian Optimization*. Cambridge University Press.

González, J.; Longworth, J.; James, D. C.; and Lawrence, N. D. 2015. Bayesian Optimization for Synthetic Gene Design. arXiv:1505.01627.

Grosnit, A.; Tutunov, R.; Maraval, A. M.; Griffiths, R.-R.; Cowen-Rivers, A. I.; Yang, L.; Zhu, L.; Lyu, W.; Chen, Z.; Wang, J.; Peters, J.; and Bou-Ammar, H. 2021. High-Dimensional Bayesian Optimisation with Variational Autoencoders and Deep Metric Learning. arXiv:2106.03609.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*.

Jenatton, R.; Archambeau, C.; González, J.; and Seeger, M. W. 2017. Bayesian Optimization with Tree-structured Dependencies. In *ICML*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kusner, M. J.; Paige, B.; and Hernández-Lobato, J. M. 2017. Grammar Variational Autoencoder. In *ICML*.

Levesque, J.; Durand, A.; Gagné, C.; and Sabourin, R. 2017. Bayesian optimization for conditional hyperparameter spaces. In *IJCNN*.

Lin, T.; Wang, Y.; Liu, X.; and Qiu, X. 2022. A survey of transformers. *AI Open*.

Lin, X.; Ding, C.; Zeng, J.; and Tao, D. 2020. Gps-net: Graph property sensing network for scene graph generation. In *CVPR*.

Lu, X.; Gonzalez, J.; Dai, Z.; and Lawrence, N. D. 2018. Structured Variationally Auto-encoded Optimization. In *ICML*.

Ma, X.; and Blaschko, M. B. 2020a. Additive Tree-Structured Conditional Parameter Spaces in Bayesian Optimization: A Novel Covariance Function and a Fast Implementation. *TPAMI*.

Ma, X.; and Blaschko, M. B. 2020b. Additive Tree-Structured Covariance Function for Conditional Parameter Spaces in Bayesian Optimization. In *AISTATS*.

Martinez-Cantin, R.; de Freitas, N.; Doucet, A.; and Castellanos, J. A. 2007. Active Policy Learning for Robot Planning and Exploration under Uncertainty. In *RSS III*.

Maus, N.; Jones, H.; Moore, J.; Kusner, M. J.; Bradshaw, J.; and Gardner, J. R. 2022. Local Latent Space Bayesian Optimization over Structured Inputs. In *NeurIPS*.

Mockus, J. 1994. Application of Bayesian approach to numerical methods of global and stochastic optimization. *J Glob Optim*.

Nguyen, D.; Gupta, S.; Rana, S.; Shilton, A.; and Venkatesh, S. 2020. Bayesian Optimization for Categorical and Category-Specific Continuous Inputs. In *AAAI*.

Papenmeier, L.; Nardi, L.; and Poloczek, M. 2022. Increasing the Scope as You Learn: Adaptive Bayesian Optimization in Nested Subspaces. In *NeurIPS*.

Papenmeier, L.; Nardi, L.; and Poloczek, M. 2023. Bounce: Reliable High-Dimensional Bayesian Optimization for Combinatorial and Mixed Spaces. In *NeurIPS*.

Perrone, V.; Jenatton, R.; Seeger, M. W.; and Archambeau, C. 2018. Scalable Hyperparameter Transfer Learning. In *NeurIPS*.

Pineda-Arango, S.; Jomaa, H. S.; Wistuba, M.; and Grabocka, J. 2021. HPO-B: A Large-Scale Reproducible Benchmark for Black-Box HPO based on OpenML. In *NeurIPS*.

Roussel, R.; Edelen, A. L.; Boltz, T.; Kennedy, D.; Zhang, Z.; Ji, F.; Huang, X.; Ratner, D.; Garcia, A. S.; Xu, C.; et al. 2024. Bayesian optimization algorithms for accelerator physics. *Physical Review Accelerators and Beams*.

Seeger, M. W. 2004. Gaussian Processes For Machine Learning. *Int. J. Neural Syst.*

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and de Freitas, N. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NeurIPS*.

Springenberg, J. T.; Klein, A.; Falkner, S.; and Hutter, F. 2016. Bayesian Optimization with Robust Bayesian Neural Networks. In *NeurIPS*.

Swersky, K.; Duvenaud, D.; Snoek, J.; Hutter, F.; and Osborne, M. A. 2014. Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces. arXiv:1409.4011.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *SIGKDD*.

Tripp, A.; Daxberger, E. A.; and Hernández-Lobato, J. M. 2020. Sample-Efficient Optimization in the Latent Space of Deep Generative Models via Weighted Retraining. In *NeurIPS*.

Vanschoren, J.; van Rijn, J. N.; Bischl, B.; and Torgo, L. 2013. OpenML: Networked Science in Machine Learning. In *SIGKDD*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NeurIPS*.

Wilson, A. G.; Hu, Z.; Salakhutdinov, R.; and Xing, E. P. 2016. Deep Kernel Learning. In *AISTATS*.

Wistuba, M.; and Grabocka, J. 2021. Few-Shot Bayesian Optimization with Deep Kernel Surrogates. In *ICLR*.