Reinforced In-Context Black-Box Optimization

Lei Song[1,2] , **Chen-Xiao Gao**[1,2] , **Ke Xue**[1,2] , **Chenyang Wu**[1,2] , **Dong Li**[3] ,
Jianye Hao[3,4] , **Zongzhang Zhang**[1,2] and **Chao Qian**[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
[3]Huawei Noah's Ark Lab, China
[4]College of Intelligence and Computing, Tianjin University, China
{songl, gaocx, xuek, wucy, zzzhang, qianc}@lamda.nju.edu.cn, {lidong106, haojianye}@huawei.com

Abstract

Black-Box Optimization (BBO) has found successful applications in many fields of science and engineering. Recently, there has been a growing interest in meta-learning particular components of BBO algorithms to speed up optimization and get rid of tedious hand-crafted heuristics. As an extension, learning the entire algorithm from data requires the least labor from experts and can provide the most flexibility. In this paper, we propose RIBBO, a method to reinforce-learn a BBO algorithm from offline data in an end-to-end fashion. RIBBO employs expressive sequence models to learn the optimization histories produced by multiple behavior algorithms and tasks, leveraging the in-context learning ability of large models to extract task information and make decisions accordingly. Central to our method is to augment the optimization histories with *regret-to-go* tokens, which are designed to represent the performance of an algorithm based on cumulative regret over the future part of the histories. The integration of regret-to-go tokens enables RIBBO to automatically generate sequences of query points that are positively correlated to the user-desired regret, verified by its universally good empirical performance on diverse problems, including BBO benchmark, hyper-parameter optimization, and robot control problems.

1 Introduction

Black-Box Optimization (BBO) [Alarie *et al.*, 2021] refers to optimizing objective functions where neither analytic expressions nor derivatives of the objective are available. To solve BBO problems, we can only access the results of objective evaluations, which usually also incur a high computational cost. Many fundamental problems in science and engineering involve the optimization of expensive BBO functions, such as drug discovery [Terayama *et al.*, 2021], material design [Frazier and Wang, 2016], integrated circuit design [Shi *et al.*, 2023] and so on.

To date, a lot of BBO algorithms have been developed, among which the most prominent ones are Bayesian Optimization (BO) [Frazier, 2018] and Evolutionary Algorithms (EA) [Back, 1996]. Despite the advancements, these algorithms typically solve BBO problems from scratch and rely on expert-derived heuristics. Consequently, they are often hindered by slow convergence rates, and unable to leverage the inherent structures within the optimization problems [Astudillo and Frazier, 2021; Bai *et al.*, 2023].

Recently, there has been a growing interest in meta-learning particular components of algorithms using previously collected data [Arango *et al.*, 2021; Feurer *et al.*, 2021]. Learning these components not only alleviates the need for the laborious design process of the domain experts, but also specifies the components with domain-specific data to facilitate subsequent optimization. For example, several components in BO are proposed to be learned from data, including the surrogate model [Perrone *et al.*, 2018; Wang *et al.*, 2021; Wistuba and Grabocka, 2021; Müller *et al.*, 2023], the acquisition function [Volpp *et al.*, 2020; Hsieh *et al.*, 2021], the initialization strategy [Feurer *et al.*, 2015; Poloczek *et al.*, 2016], and the search space [Perrone and Shen, 2019; Wang *et al.*, 2024]. Some core evolutionary operations in EA have also been considered, e.g., learning the selection and mutation rate adaptation in genetic algorithm [Lange *et al.*, 2023a] or the update rules for evolution strategy [Lange *et al.*, 2023b]. Additionally, the configuration of an algorithm can also be learned and dynamically adjusted throughout the optimization process [Adriaensen *et al.*, 2022; Xue *et al.*, 2022].

There have also been some attempts to learn an entire algorithm in an End-to-End (E2E) fashion, which requires almost no expert knowledge at all and provides the most flexibility across a broad range of BBO problems. However, existing practices require additional knowledge regarding the objective function during the training stage, e.g., the gradient information (often impractical for BBO) [Chen *et al.*, 2017] or online sampling from the objective function (often very expensive) [Maraval *et al.*, 2023]. Chen *et al.* [2022] proposed the OptFormer method to imitate the behavior algorithms separately during training, presenting a challenge for the user to manually specify which algorithm to execute during testing. Thus, these methods are less ideal for practical scenarios where offline datasets are often available beforehand and a suitable algorithm for the given task has to be identified automatically without the involvement of domain experts.

In this paper, we introduce Reinforced In-context BBO

(RIBBO), which learns a reinforced BBO algorithm from offline datasets in an E2E fashion. RIBBO employs an expressive sequence model, i.e., causal transformer, to fit the optimization histories in the offline datasets generated by executing diverse behavior algorithms on multiple tasks. The sequence model is fed with previous query points and their function values, and trained to predict the distribution over the next query point. During testing, the sequence model itself serves as a BBO algorithm by generating the next query points auto-regressively. Apart from this, RIBBO augments the optimization histories with *regret-to-go* (RTG) tokens, which are calculated by summing up the regrets over the future part of the histories, representing the future performance of an algorithm. A novel Hindsight Regret Relabelling (HRR) strategy is proposed to update the RTG tokens during testing. By integrating the RTG tokens into the modeling, RIBBO can automatically identify different algorithms, and generate sequences of query points that are positively correlated to the user-desired regret. Such modeling enables RIBBO to circumvent the impact of inferior data and further reinforce its performance on top of the behavior algorithms.

We perform experiments on BBOB synthetic functions, hyper-parameter optimization and robot control problems by using some representatives of heuristic search, EA, and BO as behavior algorithms to generate the offline datasets. The results show that RIBBO can automatically generate sequences of query points related to the user-desired regret across diverse problems, and achieve good performance universally. Note that the best behavior algorithm depends on the problem at hand, and RIBBO can perform even better on some problems. Compared to the most related method OptFormer [Chen *et al.*, 2022], RIBBO also has clear advantage. In addition, we perform a series of experiments to analyze the influence of important components of RIBBO.

2 Background

2.1 Black-Box Optimization

Let $f : \mathcal{X} \to \mathbb{R}$ be a black-box function, where $\mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional search space. The goal of BBO is to find an optimal solution $\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$, with the only permission of querying the objective function value. Several classes of BBO algorithms have been proposed, e.g., BO [Frazier, 2018] and EA [Back, 1996]. The basic framework of BO contains two critical components: a surrogate model, typically formalized as Gaussian Process (GP), and an acquisition function [Balandat *et al.*, 2020], which are used to model f and decide the next query point, respectively. EA is a class of heuristic optimization algorithms inspired by natural evolution. It maintains a population of solutions and iterates through mutation, crossover, and selection operations to find better solutions.

To evaluate the performance of BBO algorithms, regrets are often used. The instantaneous regret $r_t = f(\boldsymbol{x}^*) - f(\boldsymbol{x}_t)$ measures the gap of function values between an optimal solution $\boldsymbol{x}^*$ and the currently selected point $\boldsymbol{x}_t$. The cumulative regret $\mathrm{Reg}_T = \sum_{i=1}^{T} r_i$ is the sum of instantaneous regrets in the first T iterations.

2.2 Meta-Learning in Black-Box Optimization

Hand-crafted BBO algorithms usually require an expert to analyze the algorithms' behavior across a wide range of problems, a process that is both tedious and time-consuming. One solution is meta-learning [Hospedales *et al.*, 2021], which aims to exploit knowledge to improve the performance of learning algorithms given data from a collection of tasks. By parameterizing a component of BBO algorithms or even an entire BBO algorithm that is traditionally manually designed, we can utilize historical data to incorporate domain knowledge into the optimization, which may bring speedup.

Meta-learning particular components has been studied with different BBO algorithms. Meta-learning in BO can be divided into four main categories according to "what to transfer" [Bai *et al.*, 2023], including the design of the surrogate model, acquisition function, initialization strategy, and search space. For surrogate model design, Wang *et al.* [2021] and Wistuba & Grabocka [2021] parameterized the mean or kernel function of the GP model with Multi-Layer Perceptron (MLP), while Perrone *et al.* [2018] and Muller *et al.* [2023] substituted GP with Bayesian linear regression or neural process [Garnelo *et al.*, 2018; Müller *et al.*, 2022]. For acquisition function design, MetaBO [Volpp *et al.*, 2020] uses Reinforcement Learning (RL) to meta-train an acquisition function on a set of related tasks, and FSAF [Hsieh *et al.*, 2021] employs a Bayesian variant of deep Q-network as a surrogate differentiable acquisition function trained by model-agnostic meta-learning [Finn *et al.*, 2017]. The remaining two categories focus on exploiting the previous good solutions to warm start the optimization [Feurer *et al.*, 2015; Poloczek *et al.*, 2016] or shrink the search space [Perrone and Shen, 2019; Wang *et al.*, 2024]. Meta-learning in EA usually focuses on learning specific evolutionary operations. For example, Lang *et al.* substituted core genetic operators, i.e., selection and mutation rate adaptation, with dot-product attention modules [Lange *et al.*, 2023a], and meta-learned a self-attention-based architecture to discover effective and order-invariant update rules [Lange *et al.*, 2023b]. Beyond that, dynamic algorithm configuration [Adriaensen *et al.*, 2022; Xue *et al.*, 2022] concentrates on learning the configurations of algorithms, employing RL to dynamically adjust the configurations during the optimization process.

Meta-learning entire algorithms has also been explored to obtain more flexible models. Early works [Chen *et al.*, 2017] use Recurrent Neural Network (RNN) to meta-learn a BBO algorithm by optimizing the summed objective functions of some iterations. RNN uses its memory state to store information about history and outputs the next query point. This work assumes access to gradient information during the training phase, which is, however, usually impractical in BBO problems. OptFormer [Chen *et al.*, 2022] uses a text-based transformer framework to learn an algorithm, providing a universal E2E interface for BBO problems. It is trained to imitate different BBO algorithms across a broad range of problems, which, however, presents a challenge for the user to manually specify an algorithm for inference. Neural Acquisition Processes (NAP) [Maraval *et al.*, 2023] uses transformer to meta-learn the surrogate model and acquisition function of BO jointly. Due to the lack of labeled acquisition data, NAP uses an online RL algorithm with a supervised auxiliary loss for training, which requires online sampling from the expensive objective function and lacks efficiency. Black-box Optimization NETworks (BONET) [Krishnamoorthy *et al.*, 2023] employ a transformer model to fit regret-augmented trajectories in an offline BBO scenario, where the training and testing data are from the same objective function, and a prefix sequence is required to warm up the optimization before testing. OPT-GAN [Lu *et al.*, 2023] utilizes Generative Adversarial Networks (GAN) to estimate the distribution of optimum gradually by exploration-exploitation trade-off. Compared to the above state-of-the-art E2E methods, we consider the meta-BBO setting, where the training datasets are generated from diverse algorithms across different functions. Our approach offers the advantage of automatically identifying (with RTG tokens) and deploying the best-performing algorithm without requiring the user to pre-specify which algorithm to use or to provide a prefix sequence during the testing phase. It utilizes a supervised learning loss for training on a fixed offline dataset without the need for further interaction with the objective function.

2.3 Decision Transformer

Transformer has emerged as a powerful architecture for sequence modeling tasks [Lin *et al.*, 2022]. A basic building block behind transformer is the self-attention mechanism [Vaswani *et al.*, 2017], which captures correlations between any token pairs across timesteps. As the scale of data and model increases, transformer has demonstrated the *in-context learning* ability [Brown *et al.*, 2020], referring to the capability of the model to infer the tasks at hand based on the input contexts. Decision Transformer (DT) [Chen *et al.*, 2021; Gao *et al.*, 2024] abstracts RL as a sequence modeling problem, and introduces return-to-go tokens, representing the cumulative future rewards. Conditioning on return-to-go tokens enables DT to correlate the trajectories with their corresponding returns and generate future actions to achieve a user-specified return. Inspired by DT, we will treat BBO tasks as a sequence modeling problem naturally, use a causal transformer for modeling, and train it by conditioning on future regrets. Such design is expected to enable the learned model to distinguish algorithms with different performance and achieve good results with a user-specified low regret. Further discussions on DT are available in Appendix I.

3 Method

This section presents Reinforced In-context Black-Box Optimization (RIBBO), which learns an enhanced BBO algorithm in an E2E fashion, as illustrated in Figure 1. We follow the task-distribution assumption, which is commonly adopted in meta-learning settings [Finn *et al.*, 2017; Hospedales *et al.*, 2021]. Our goal is to learn a generalizable model $\mathcal{M}$ capable of solving a wide range of BBO tasks, each associated with a BBO objective function f sampled from the task distribution $P(\mathcal{F})$, where $\mathcal{F}$ denotes the function space.

Let $[N]$ denote the integer set $\{1, 2, \dots, N\}$. During training, we usually access N source tasks and each task corresponds to an objective function $f_i \sim P(\mathcal{F})$, where $i \in [N]$. Hereafter, we use f_i to denote the task i if the context is clear. We assume that the information is available via offline datasets $\mathcal{D}_{i,j}$, which are produced by executing a behavior algorithm $\mathcal{A}_j$ on task f_i, where $j \in [K]$ and $i \in [N]$. Each dataset $\mathcal{D}_{i,j} = \{\boldsymbol{h}_T^{i,j,m}\}_{m=1}^M$ consists of M optimization histories $\boldsymbol{h}_T^{i,j,m} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$, where $\boldsymbol{x}_t$ is the query point selected by $\mathcal{A}_j$ at iteration t, and $y_t = f_i(\boldsymbol{x}_t)$ is its objective value. If the context is clear, we will omit i, j, m and simply use $\boldsymbol{h}_T$ to denote a history with length T. The initial history $\boldsymbol{h}_0$ is defined as $\emptyset$. We impose no additional assumptions about the behavior algorithms, allowing for a range of BBO algorithms, even random search.

With the datasets, we seek to learn a model $\mathcal{M}_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{h}_{t-1})$, which is parameterized by $\boldsymbol{\theta}$ and generates the next query point $\boldsymbol{x}_t$ by conditioning on the previous history $\boldsymbol{h}_{t-1}$. As introduced in Section 2.1, with a given budget T and the history $\boldsymbol{h}_T$ produced by an algorithm $\mathcal{A}$, we use the cumulative regret to evaluate performance:

$$\mathrm{Reg}_T = \sum\nolimits_{t=1}^{T} (y^* - y_t) \tag{1}$$

as the evaluation metric, where y^* is the optimum value and $\{y_t\}_{t=1}^T$ are the function values in $\boldsymbol{h}_T$.

3.1 Method Outline

Given the current history $\boldsymbol{h}_{t-1}$ at iteration t, a BBO algorithm usually selects the next query point $\boldsymbol{x}_t$, observes the function value $y_t = f_i(\boldsymbol{x}_t)$, and updates the history $\boldsymbol{h}_t = \boldsymbol{h}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$. Similar to previous work [Chen *et al.*, 2017], we take this framework as a starting point and treat the learning of a universal BBO algorithm as learning a model $\mathcal{M}_{\boldsymbol{\theta}}$, which takes the preceding history $\boldsymbol{h}_{t-1}$ as input and outputs a distribution of the next query point $\boldsymbol{x}_t$. The optimization histories in offline datasets provide natural supervision for the learning process.

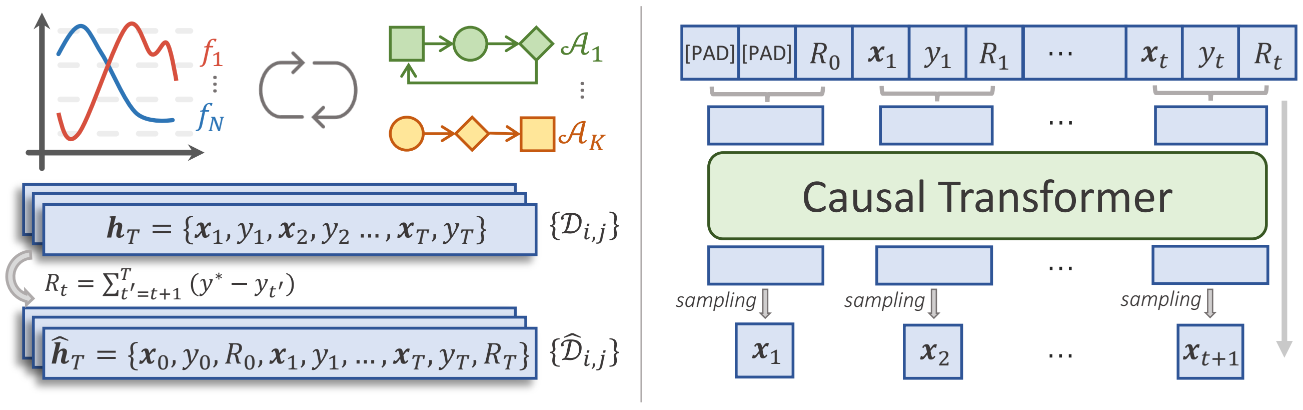
Suppose we have a set of histories $\{\boldsymbol{h}_T\}$, generated by a single behavior algorithm $\mathcal{A}$ on a single task f. By employing a causal transformer model $\mathcal{M}_{\boldsymbol{\theta}}$, we expect $\mathcal{M}_{\boldsymbol{\theta}}$ to imitate $\mathcal{A}$ and produce similar optimization history on f. In practice, we usually have datasets containing histories from multiple behavior algorithms $\{\mathcal{A}_j\}_{j=1}^K$ on multiple tasks $\{f_i\}_{i=1}^N$. To fit $\mathcal{M}_{\boldsymbol{\theta}}$, we use the negative log-likelihood loss

$$\mathcal{L}_{\mathrm{BC}}(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{h}_T \sim \mathcal{D}_{i,j}} \left[\sum\nolimits_{t=1}^{T} \log \mathcal{M}_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{h}_{t-1}) \right]. \tag{2}$$

To effectively minimize this loss, $\mathcal{M}_{\boldsymbol{\theta}}$ needs to recognize both the task and the behavior algorithm in-context, and then imitate the optimization behavior of the corresponding behavior algorithm.

Nevertheless, naively imitating the offline datasets hinders the model since some inferior behavior algorithms may severely degenerate the model's performance. Inspired by DT [Chen *et al.*, 2021], we propose to augment the optimization history with Regret-To-Go (RTG) tokens R_t, defined as the sum of instantaneous regrets over the future history:

$$\hat{\boldsymbol{h}}_T = \{(\boldsymbol{x}_t, y_t, R_t)\}_{t=0}^T,\ R_t = \sum\nolimits_{t'=t+1}^{T} (y^* - y_{t'}), \tag{3}$$



f_1
f_N
$\mathcal{A}_1$
$\mathcal{A}_K$
$\boldsymbol{h}_T = \{\boldsymbol{x}_1, y_1, \boldsymbol{x}_2, y_2 \dots, \boldsymbol{x}_T, y_T\}$
$\{\mathcal{D}_{i,j}\}$
$R_t = \sum_{t'=t+1}^{T} (y^* - y_{t'})$
$\hat{\boldsymbol{h}}_T = \{\boldsymbol{x}_0, y_0, R_0, \boldsymbol{x}_1, y_1, \dots, \boldsymbol{x}_T, y_T, R_T\}$
$\{\widehat{\mathcal{D}}_{i,j}\}$
[PAD] [PAD] R_0 $\boldsymbol{x}_1$ y_1 R_1 $\cdots$ $\boldsymbol{x}_t$ y_t R_t
Causal Transformer
sampling
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_{t+1}$

(a) Data Generation (b) Training and Inference

Figure 1: Illustration of RIBBO. *Left: Data Generation.* K existing BBO algorithms $\{\mathcal{A}_j\}_{j=1}^K$ and N BBO tasks $\{f_i\}_{i=1}^N$ are used to serve as the behavior algorithms and the training tasks, respectively. The offline datasets $\{\mathcal{D}_{i,j}\}$ consist of the optimization histories $\boldsymbol{h}_T = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$ collected by executing each behavior algorithm $\mathcal{A}_j$ on each task f_i for T evaluation steps, which are then augmented with the regret-to-go tokens R_t (calculated as the cumulative regret $\sum_{t'=t+1}^T (y^* - y_{t'})$ over the future optimization history) to generate the final dataset $\{\widehat{\mathcal{D}}_{i,j}\}$ for training. *Right: Training and Inference.* Our model takes in triplets of $(\boldsymbol{x}_t, y_t, R_t)$, embeds them into one token, and outputs the distribution over the next query point $\boldsymbol{x}_{t+1}$. During training, the ground-truth next query point is used to minimize the loss in Eq. (4). During inference, the next query point $\boldsymbol{x}_{t+1}$ is generated auto-regressively based on the current history $\hat{\boldsymbol{h}}_t$.

where $\boldsymbol{x}_0$ and y_0 are placeholders for padding, denoted as [PAD] in Figure 1(b), and $R_T = 0$. The augmented histories compose the augmented dataset $\widehat{\mathcal{D}}_{i,j}$, and the training objective of $\mathcal{M}_{\boldsymbol{\theta}}$ becomes

$$\mathcal{L}_{\text{RIBBO}}(\boldsymbol{\theta}) = -\mathbb{E}_{\hat{\boldsymbol{h}}_T \sim \widehat{\mathcal{D}}_{i,j}} \left[\sum_{t=1}^{T} \log \mathcal{M}_{\boldsymbol{\theta}}(\boldsymbol{x}_t | \hat{\boldsymbol{h}}_{t-1}) \right]. \quad (4)$$

The integration of RTG tokens in the context brings identifiability of behavior algorithms, and the model $\mathcal{M}_{\boldsymbol{\theta}}$ can effectively utilize them to make appropriate decisions. Furthermore, RTG tokens have a direct correlation with the metric of interest, i.e., cumulative regret Reg_T in Eq. (1). Conditioning on a lower RTG token provides a guidance to our model and reinforces $\mathcal{M}_{\boldsymbol{\theta}}$ to exhibit superior performance. These advantages will be clearly shown by experiments in Section 4.4.

The resulting method RIBBO has implicitly utilized the in-context learning capacity of transformer to guide the optimization with previous histories and the desired future regret as context. The in-context learning capacity of inferring the tasks at hand based on the input contexts has been observed as the scale of data and model increases [Kaplan *et al.*, 2020]. It has been explored to infer general functional relationships as supervised learning or RL algorithms. For example, the model is expected to behave as a supervised learning algorithm to accurately predict the query input $\boldsymbol{x}_t$ by feeding the training dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{t-1}$ as the context [Li *et al.*, 2023]; Laskin *et al.* [2023] learned RL algorithms using causal transformers. Here, we use it for BBO.

3.2 Practical Implementation

Next, we detail the model architecture, training and inference processes of RIBBO.

Model Architecture. For the formalization of the model $\mathcal{M}_{\boldsymbol{\theta}}$, we adopt the commonly used GPT architecture [Radford *et al.*, 2018], which comprises a stack of causal attention blocks. Each block is composed of an attention mechanism and a feed-forward network. We aggregate each triplet $(\boldsymbol{x}_i, y_i, R_i)$ using a two-layer MLP network. The output of $\mathcal{M}_{\boldsymbol{\theta}}$ is a diagonal Gaussian distribution of the next query point. Note that previous works that adopt the sequence model as surrogate models [Müller *et al.*, 2022; Nguyen and Grover, 2022; Müller *et al.*, 2023] typically remove the positional encoding because the surrogate model should be invariant to the history order. On the contrary, our implementation preserves the positional encoding, naturally following the behavior of certain algorithms (e.g., BO or EA) and making it easier to learn from algorithms. Additionally, the positional encoding can help maintain the monotonically decreasing order of RTG tokens. More details about the architecture can be found in Appendix A.

Model Training. RTG tokens are calculated as outlined in Eq. (3) for the offline datasets before training. Since the calculation of regret requires the optimum value of task i, we use the best-observed value $y_{\max}^i$ as a proxy for the optimum. Let $\{\widehat{\mathcal{D}}_{i,j}\}_{i\in[N],j\in[K]}$ denote the RTG augmented datasets with N tasks and K algorithms. During training, we sample a minibatch of consecutive subsequences of length $\tau < T$ uniformly from the augmented datasets. The training objective is to minimize the RIBBO loss in Eq. (4).

Model Inference. The model $\mathcal{M}_{\boldsymbol{\theta}}$ generates the query points $\boldsymbol{x}_t$ auto-regressively during inference, which involves iteratively selecting a new query point $\boldsymbol{x}_t$ based on the current augmented history $\hat{\boldsymbol{h}}_{t-1}$, evaluating the query as $y_t = f(\boldsymbol{x}_t)$, and updating the history by $\hat{\boldsymbol{h}}_t = \hat{\boldsymbol{h}}_{t-1} \cup \{(\boldsymbol{x}_t, y_t, R_t)\}$. A critical aspect of this procedure is specifying the value of RTG (i.e., R_t) at each iteration t. Inspired by DT, a naive approach is to set a desired performance as the initial RTG R_0, and decrease it as $R_t = R_{t-1} - (y^* - y_t)$. However, this strat-

Algorithm 1 Model Inference with HRR

```
Input: trained model $\mathcal{M}_{\boldsymbol{\theta}}$, budget $T$, optimum value $y^*$
Process:
1: Initialize $\hat{\boldsymbol{h}}_0 = \{(\boldsymbol{x}_0, y_0, R_0)\}$, where $\boldsymbol{x}_0$ and $y_0$ are placeholders for padding and $R_0 = 0$;
2: for $t = 1, 2, \dots, T$ do
3:     Generate the next query point $\boldsymbol{x}_t \sim \mathcal{M}_{\boldsymbol{\theta}}(\cdot|\hat{\boldsymbol{h}}_{t-1})$;
4:     Evaluate $\boldsymbol{x}_t$ to obtain $y_t = f(\boldsymbol{x}_t)$;
5:     Calculate the instantaneous regret $r = y^* - y_t$;
6:     Relabel $R_i \leftarrow R_i + r$, for each $(\boldsymbol{x}_i, y_i, R_i)$ in $\hat{\boldsymbol{h}}_{t-1}$;
7:     $\hat{\boldsymbol{h}}_t = \hat{\boldsymbol{h}}_{t-1} \cup \{(\boldsymbol{x}_t, y_t, 0)\}$
8: end for
```

egy has the risk of producing out-of-distribution RTGs, since the values can fall below 0 due to an improperly selected R_0.

Given the fact that RTGs are lower bounded by 0 and a value of 0 implies a good BBO algorithm with low regret, we propose to set the immediate RTG as 0. Furthermore, we introduce a strategy called **Hindsight Regret Relabelling (HRR)** to update previous RTGs based on the current sample evaluations. The inference procedure with HRR is detailed in Algorithm 1. In line 1, the history $\hat{\boldsymbol{h}}_0$ is initialized with padding placeholders $\boldsymbol{x}_0, y_0$ and RTG $R_0 = 0$. At iteration t (i.e., lines 3–7), the model $\mathcal{M}_{\boldsymbol{\theta}}$ is fed with the augmented history $\hat{\boldsymbol{h}}_{t-1}$ to generate the next query point $\boldsymbol{x}_t$ in line 3, followed by the evaluation procedure to obtain y_t in line 4. Then, the immediate RTG R_t is set to 0, and we employ HRR to update previous RTG tokens in $\hat{\boldsymbol{h}}_{t-1}$, i.e., calculate the instantaneous regret $r = y^* - y_t$ (line 5) and add r to every RTG token within $\hat{\boldsymbol{h}}_{t-1}$ (line 6):

$$\forall 0 \leq i < t, R_i \leftarrow R_i + (y^* - y_t). \tag{5}$$

Note that this relabelling process guarantees that $\forall 0 \leq i < t$, the RTG token $R_i = \sum_{t'=i+1}^{t}(y^* - y_{t'})$, which can also be written as $\sum_{t'=i+1}^{T}(y^* - y_{t'})$, consistent with the definition in Eq. (3), because the immediate RTG $R_t = \sum_{t'=t+1}^{T}(y^* - y_{t'})$ is set to 0. In line 7, the history $\hat{\boldsymbol{h}}_t$ is updated by expanding $\hat{\boldsymbol{h}}_{t-1}$ with $\{(\boldsymbol{x}_t, y_t, 0)\}$, i.e., the current sampling and its immediate RTG $R_t = 0$. The above process is repeated until reaching the budget T. Thus, we can find that HRR not only exploits the full potential of $\mathcal{M}_{\boldsymbol{\theta}}$ through using 0 as the immediate RTG and thereby demands the model to generate the most advantageous decisions, but also preserves the calculation of RTG tokens following the same way as the training data, i.e., representing the cumulative regret over future optimization history. More discussions about the RTG tokens can be found in Appendix F and G.

3.3 Data Generation

Finally, we give some guidelines about data generation for using the proposed RIBBO method.

Data Collection. Given a set of tasks $\{f_i\}_{i=1}^N$ sampled from the task distribution $P(\mathcal{F})$, we can employ a diverse set of behavior algorithms for data collection. For example, we can select some representatives from different types of BBO algorithms, e.g., BO and EA. Datasets $\mathcal{D}_{i,j}$ are obtained by using each behavior algorithm to optimize each task with different random seeds. Each optimization history $\boldsymbol{h}_T = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$ in $\mathcal{D}_{i,j}$ is then augmented with RTG tokens R_t, which is computed as in Eq. (3). The resulting histories $\hat{\boldsymbol{h}}_T = \{(\boldsymbol{x}_t, y_t, R_t)\}_{t=0}^T$ compose the final datasets $\widehat{\mathcal{D}}_{i,j}$ for model training.

Data Normalization. To provide a unified interface and balance the statistic scales across tasks, it is important to apply normalization to the inputs to our model. We normalize the point $\boldsymbol{x}$ by $(\boldsymbol{x} - \boldsymbol{x}_{\min})/(\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min})$, with $\boldsymbol{x}_{\max}$ and $\boldsymbol{x}_{\min}$ being the upper and lower bounds of the search space, respectively. For the function value y, we apply random scaling akin to previous works [Wistuba and Grabocka, 2021; Chen *et al.*, 2022]. That is, when sampling a history $\boldsymbol{h}_\tau$ from the datasets $\mathcal{D}_{i,j}$, we randomly sample the lower bound $l \sim \mathcal{U}(y^i_{\min} - \frac{s}{2}, y^i_{\min} + \frac{s}{2})$ and the upper bound $u \sim \mathcal{U}(y^i_{\max} - \frac{s}{2}, y^i_{\max} + \frac{s}{2})$, where $\mathcal{U}$ stands for uniform distribution, $y^i_{\min}, y^i_{\max}$ denote the observed minimum and maximum values for f_i, and $s = y^i_{\max} - y^i_{\min}$; the values y_t in $\boldsymbol{h}_\tau$ are then normalized by $(y_t - l)/(u - l)$ for training. The RTG tokens are calculated accordingly with the normalized values. The random normalization can make a model exhibit invariance across various scales of y. For inference, the average values of the best-observed and worst-observed values across the training tasks are used to normalize y.

4 Experiments

In this section, we examine the performance of RIBBO across various tasks, such as synthetic functions, Hyper-Parameter Optimization (HPO) and robot control problems. The model architecture and hyper-parameters are consistent across these problems, with the average performance and standard deviation being reported after execution using distinct random seeds. Details of the model are given in Appendix A. Our code is available at https://github.com/lamda-bbo/RIBBO.

4.1 Experimental Setup

Benchmarks. We use BBO Benchmarks (BBOB) [Elhara *et al.*, 2019], HPO-B [Arango *et al.*, 2021], and rover trajectory planning task [Wang *et al.*, 2018]. The BBOB suite, a comprehensive and widely used benchmark in the continuous domain, consists of 24 synthetic functions. For each function, a series of linear and non-linear transformations are implemented on the search space to obtain a distribution of functions with similar properties. HPO-B is a commonly used HPO benchmark and consists of a series of HPO problems. Each problem is to optimize a machine learning model across various datasets, and an XGBoost model is provided as the objective function for evaluation in a continuous space. We conduct experiments on two widely used models, SVM and XGBoost, in the continue domain. For robot control optimization, we perform experiments on rover trajectory planning task, which is a trajectory optimization problem to emulate rover navigation. Similar to Elhara *et al.* [2019] and Volpp *et al.* [2020], we implement random translations and scalings to the search space to construct a distribution of functions. For BBOB and rover problems, we sample a set of functions from the task distribution as training and test tasks, while for HPO-B, we use the meta-training/test task splits provided by the authors. Detailed explanations of the benchmarks can be found in Appendix B.1.

Data. Similar to OptFormer [Chen *et al.*, 2022], we employ 7 behavior algorithms, i.e., Random Search, Shuffled Grid Search, Hill Climbing, Regularized Evolution [Real *et al.*, 2019], Eagle Strategy [Yang and Deb, 2010], CMA-ES [Hansen, 2016], and GP-EI [Balandat *et al.*, 2020], which are representatives of heuristic search, EA, and BO, respectively. Datasets are generated by employing each behavior algorithm to optimize various training functions sampled from the task distribution, using different random seeds. For inference, new test functions are sampled from the distribution to serve as the test set. Specifically, for the HPO-B problem, the meta-training/test splits have been predefined by the authors and we adhere to this standard setup. Additional information about the behavior algorithms and datasets can be found in Appendix B.2 and B.3, respectively.

4.2 Baselines

As RIBBO is an in-context E2E model, the most related baselines are those also training an E2E model with offline datasets, including Behavior Cloning (BC) [Bain and Sammut, 1995] and OptFormer [Chen *et al.*, 2022]. Their hyperparameters are set as same as that of our model for fairness. Note that the seven behavior algorithms used to generate datasets are also important baselines, and included for comparison as well.

BC uses the same transformer architecture as RIBBO. The only difference is that we do not feed RTG tokens into the model of BC and train to minimize the BC loss in Eq. (2). When the solutions are generated auto-regressively, BC tends to imitate the average behavior of various behavior algorithms. Consequently, the inclusion of underperforming algorithms, e.g., Random Search and Shuffled Grid Search, may significantly degrade the performance. To mitigate this issue, we have also trained the model by excluding these underperforming algorithms, denoted as **BC Filter**.

OptFormer employs a transformer to imitate the behaviors of a set of algorithms and an algorithm identifier usually needs to be specified manually during inference for superior performance. Its original implementation is built upon a text-based transformer with a large training scale. In this paper, we re-implement a simplified version of OptFormer where we only retain the algorithm identifier within the metadata. The initial states, denoted as $\boldsymbol{x}_0$ and y_0, are used to distinguish between algorithms. They are obtained by indexing the algorithm type through an embedding layer, thereby aligning the initial states with the specific imitated algorithm. This enables the identification of distinct behavior algorithms within the simplified OptFormer. Further details about the re-implementation can be found in Appendix C.

4.3 Main Results

Synthetic Functions. We use the BBOB suite as the synthetic benchmark functions, consisting of 24 synthetic functions. The model is trained on all BBOB synthetic functions simultaneously with the results shown in the leftmost subfigure in Figure 2. To aggregate results across functions with different output scaling, we normalize all the functions adhering to previous literature [Arango *et al.*, 2021; Chen *et al.*, 2022]. For the sake of clarity in visualization, we have omitted the inclusion of Random Search and Shuffled Grid Search due to their poor performance from start to finish. We also provide the detailed results on each BBOB function. Please see Appendix D.

The results suggest that RIBBO achieves superior average performance compared to the best behavior algorithm, BC, BC Filter and OptFormer. A single trained model across all BBOB functions behaves well, demonstrating the versatility and generalization capabilities of RIBBO. We can observe that RIBBO performs well in the early stages, drawing advantage from the HRR strategy, i.e., employing 0 as the immediate RTG to generate the optimal potential solutions. Compared with BC and BC Filter, RIBBO exhibits better performance. BC tends to imitate the average behavior of various algorithms, and its poor performance is due to the aggregation of behavior algorithms with inferior performance. BC Filter is generally better than BC, because the data from the two underperforming behavior algorithms, i.e., Random Search and Shuffled Grid Search, are manually excluded from the training of BC Filter. As introduced before, OptFormer requires manual specification of which behavior algorithm to execute. We have chosen Eagle Strategy, which obtains good overall performance on these problems. It can be observed that OptFormer displays a close performance to Eagle Strategy, while RIBBO performs better. More details about the imitation capacity of OptFormer can be found in Appendix C.

Note that the good performance of RIBBO is not due to the memorization of optimal solutions, as the search space is transformed randomly, resulting in variations in optimal solutions across different functions from the same distribution. It is due to the in-context capacity. The context data consists of the trajectory collected from new problems and the RTG tokens specified by the user. The collected trajectory provides an understanding of the problems, while the RTG tokens facilitate the identification of algorithms and reinforce the performance on top of the behavior algorithms, which will be clearly shown later. These elements are integrated as inputs, thereby influencing the resulting sampled points.

Real-World Problems. We further conduct experiments on real-world problems, including the HPO-B and rover trajectory planning task. The results are shown in Figure 2. We can observe that RIBBO achieves good performance on XGBoost and rover problems. For the SVM problem, RIBBO does not perform well, which may be due to the problem's low-dimensional nature (only three parameters) and its relative simplicity for optimization. Behavior algorithms can achieve good performance easily, while the complexity of RIBBO's training and inference processes could instead result in the performance degradation.

Cross-Distribution Generalization. We also conduct experiments to examine the cross-distribution generalization capacity of RIBBO to unseen function distributions during training. According to the mathematical properties of the functions, the BBOB suite can be divided into 5 categories (detailed in Appendix B.1), and we select one from each cat-

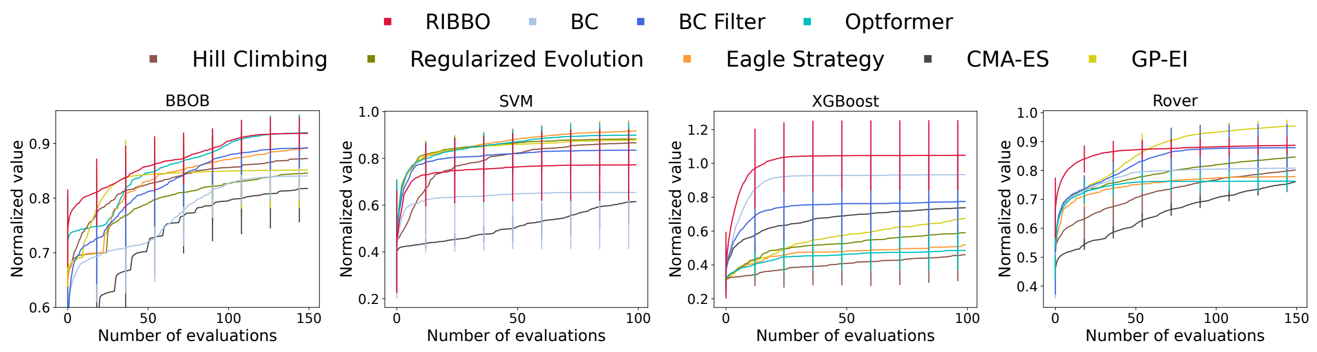
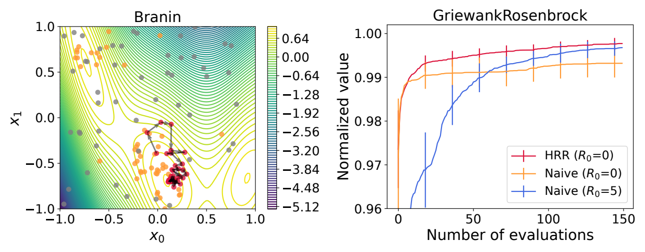


Figure 2: Performance comparison among RIBBO, BC, BC Filter, OptFormer, and behavior algorithms on synthetic functions, HPO, and robot control problems. The y-axis is the normalized average objective value, and the length of vertical bars represents the standard deviation.



(a) Visualization (b) RTG update strategy

Figure 3: *(a) Visualization* of the contour lines of 2D Branin function and sampling points of RIBBO (red), Eagle Strategy (orange), and Random Search (gray), where the arrows represent the optimization trajectory of RIBBO. *(b) RTG update strategy* comparison between HRR and the naive strategy with various initial RTG R_0.

egory due to resource constraints, including Greiwank Rosenbrock, Lunacek, Rastrigin, Rosenbrock, and Sharp Ridge. The training is performed on 4 of the 5 chosen synthetic functions, with the remaining one used for testing. Note that each function here actually represents a distribution of functions with similar properties, and a set of functions is sampled from each distribution as introduced before. The results demonstrate RIBBO's strong generalizing ability to unseen function distributions with different properties. Due to space limitation, comprehensive experimental results and analyses are provided in Appendix E.

Why Does RIBBO Behave Well? To better understand RIBBO, we train the model using only two behavior algorithms, Eagle Strategy and Random Search, which represent a good algorithm and an underperforming one, respectively. Figure 3(a) illustrates the contour lines of the 2D Branin function along with the sampling points of RIBBO, Eagle Strategy, and Random Search, which are represented by red, orange, and gray points, respectively. Arrows are used to depict the optimization trajectory of RIBBO. Note that the two parameters of the Branin function have been scaled to the range of $[-1, 1]$ for better visualization. It can be observed that RIBBO tends to favor Eagle Strategy over Random Search, indicating its ability to automatically identify the quality of training data. Additionally, RIBBO achieves a balance between exploration and exploitation based on its knowledge acquired during training, thereby achieving superior solutions than those in the training dataset.

4.4 Ablation Studies

Effectiveness of HRR. A key point of the inference procedure is how to update the value of RTGs at each iteration. To assess the effectiveness of the proposed strategy, HRR, as outlined in Eq. (5), we compare it with the naive strategy, that sets an initial RTG token R_0 and decreases it by the one-step regret after each iteration. This method employs the same updating mechanism as DT. The results are shown in Figure 3(b). The naive strategy shows distinct behaviors depending on the initial setting of R_0. Specifically, when $R_0 = 0$, i.e., the lower bound of regret, the model performs well initially. However, as the optimization progresses, the RTG tokens gradually decrease to negative values, leading to poor performance since negative RTGs are out-of-distribution values. Using $R_0 = 5$ compromises the initial performance, as the model may not select the most aggressive solutions under a high R_0. However, a higher initial R_0 yields better convergence value since it prevents out-of-distribution RTGs in later stage. The proposed HRR strategy consistently outperforms across the entire optimization stage, because setting the immediate RTG to 0 encourages the model to make the most advantageous decisions at every iteration, while hindsight relabeling of previous RTG tokens, as specified in Eq. (5), ensures that these values remain meaningful and feasible.

Further Studies. We also study the effects of the initial RTGs R_0, the immediate RTG R_t, the method to aggregate $(\boldsymbol{x}_i, y_i, R_i)$ tokens, the normalization method for y, the model size, and the sampled subsequence length τ. For more details, please see Appendix G. Additional visualizations illustrating the effects of random transformations on the search space are detailed in Appendix H.

5 Conclusion

This paper proposes RIBBO, which employs a transformer architecture to learn a reinforced BBO algorithm from offline datasets in an E2E fashion. By incorporating RTG tokens into the optimization histories, RIBBO can automatically generate optimization trajectories satisfying the user-desired regret. Comprehensive experiments on synthetic functions, HPO and robot control problems show the versatility of RIBBO. This work is a preliminary attempt towards universal BBO, and we hope it can encourage more explorations in this direction. For more potential future works, please refer to Appendix J.

Acknowledgements

This work was supported by the National Science and Technology Major Project (2022ZD0116600), the National Science Foundation of China (62422605, 92370132, 62276126), and the Fundamental Research Funds for the Central Universities (14380020). The version with Appendix is available at https://arxiv.org/abs/2402.17423.

Contribution Statement

Lei Song and Chen-Xiao Gao share equal contribution. Chao Qian is the corresponding author.

References

[Adriaensen *et al.*, 2022] S. Adriaensen, A. Biedenkapp, G. Shala, N. H. Awad, T. Eimer, M. Lindauer, and F. Hutter. Automated dynamic algorithm configuration. *JAIR*, 75:1633–1699, 2022.

[Alarie *et al.*, 2021] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel. Two decades of blackbox optimization applications. *EJCO*, 9:100011, 2021.

[Arango *et al.*, 2021] S. P. Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. *arXiv:2106.06257*, 2021.

[Astudillo and Frazier, 2021] R. Astudillo and P. I. Frazier. Thinking inside the box: A tutorial on grey-box Bayesian optimization. In *WSC*, pages 1–15, Phoenix, AZ, 2021.

[Back, 1996] T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Bai *et al.*, 2023] T. Bai, Y. Li, Y. Shen, X. Zhang, W. Zhang, and B. Cui. Transfer learning for Bayesian optimization: A survey. *arXiv:2302.05927*, 2023.

[Bain and Sammut, 1995] M. Bain and C. Sammut. A framework for behavioural cloning. *Machine Intelligence*, 15:103–129, 1995.

[Balandat *et al.*, 2020] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *NeurIPS*, pages 10113–10124, Virtual, 2020.

[Brown *et al.*, 2020] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *NeurIPS*, pages 1877–1901, Virtual, 2020.

[Chen *et al.*, 2017] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas. Learning to learn without gradient descent by gradient descent. In *ICML*, pages 748–756, Sydney, Australia, 2017.

[Chen *et al.*, 2021] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, pages 15084–15097, Virtual, 2021.

[Chen *et al.*, 2022] Y. Chen, X. Song, C. Lee, Z. Wang, R. Zhang, D. Dohan, K. Kawakami, G. Kochanski, A. Doucet, M. Ranzato, S. Perel, and N. de Freitas. Towards learning universal hyperparameter optimizers with transformers. In *NeurIPS*, pages 32053–32068, New Orleans, LA, 2022.

[Elhara *et al.*, 2019] O. Elhara, K. Varelas, D. Nguyen, T. Tusar, D. Brockhoff, N. Hansen, and A. Auger. COCO: The large scale black-box optimization benchmarking (BBOB-largescale) test suite. *arXiv:1903.06396*, 2019.

[Feurer *et al.*, 2015] M. Feurer, J. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *AAAI*, pages 1128–1135, Austin, TX, 2015.

[Feurer *et al.*, 2021] M. Feurer, J. N. Van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter. OpenML-Python: An extensible Python API for OpenML. *JMLR*, 22(1):4573–4577, 2021.

[Finn *et al.*, 2017] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, Sydney, Australia, 2017.

[Frazier and Wang, 2016] P. I. Frazier and J. Wang. *Bayesian Optimization for Materials Design*. Springer, 2016.

[Frazier, 2018] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811*, 2018.

[Gao *et al.*, 2024] C. Gao, C. Wu, M. Cao, R. Kong, Z. Zhang, and Y. Yu. ACT: Empowering decision transformer with dynamic programming via advantage conditioning. In *AAAI*, pages 12127–12135, Vancouver, Canada, 2024.

[Garnelo *et al.*, 2018] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami. Conditional neural processes. In *ICML*, pages 1704–1713, Stockholm, Sweden, 2018.

[Hansen, 2016] N. Hansen. The CMA evolution strategy: A tutorial. *arXiv:1604.00772*, 2016.

[Hospedales *et al.*, 2021] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE TPAMI*, 44(9):5149–5169, 2021.

[Hsieh *et al.*, 2021] B. Hsieh, P. Hsieh, and X. Liu. Reinforced few-shot acquisition function learning for Bayesian optimization. In *NeurIPS*, pages 7718–7731, Virtual, 2021.

[Kaplan *et al.*, 2020] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray,

A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

[Krishnamoorthy *et al.*, 2023] S. Krishnamoorthy, S. Mashkaria, and A. Grover. Generative pretraining for black-box optimization. In *ICML*, pages 24173–24197, Honolulu, HI, 2023.

[Lange *et al.*, 2023a] R. T. Lange, T. Schaul, Y. Chen, C. Lu, T. Zahavy, V. Dalibard, and S. Flennerhag. Discovering attention-based genetic algorithms via meta-black-box optimization. In *GECCO*, pages 929–937, Lisbon, Portugal, 2023.

[Lange *et al.*, 2023b] R. T. Lange, T. Schaul, Y. Chen, T. Zahavy, V. Dalibard, C. Lu, S. Singh, and S. Flennerhag. Discovering evolution strategies via meta-black-box optimization. In *ICLR*, Kigali, Rwanda, 2023.

[Laskin *et al.*, 2023] M. Laskin, L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. S. Hansen, A. Filos, E. Brooks, maxime gazeau, H. Sahni, S. Singh, and V. Mnih. In-context reinforcement learning with algorithm distillation. In *ICLR*, Kigali, Rwanda, 2023.

[Li *et al.*, 2023] Y. Li, M. E. Ildiz, D. Papailiopoulos, and S. Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In *ICML*, pages 19565–19594, Honolulu, HI, 2023.

[Lin *et al.*, 2022] T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. *AI open*, 3:111–132, 2022.

[Lu *et al.*, 2023] M. Lu, S. Ning, S. Liu, F. Sun, B. Zhang, B. Yang, and L. Wang. OPT-GAN: A broad-spectrum global optimizer for black-box problems by learning distribution. In *AAAI*, pages 12462–12472, Washington, DC, 2023.

[Maraval *et al.*, 2023] A. M. Maraval, M. Zimmer, A. Grosnit, and H. B. Ammar. End-to-end meta-Bayesian optimisation with transformer neural processes. In *NeurIPS*, pages 11246–11260, New Orleans, LA, 2023.

[Müller *et al.*, 2022] S. Müller, N. Hollmann, S. P. Arango, J. Grabocka, and F. Hutter. Transformers can do Bayesian inference. In *ICLR*, Virtual, 2022.

[Müller *et al.*, 2023] S. Müller, M. Feurer, N. Hollmann, and F. Hutter. PFNs4BO: In-context learning for Bayesian optimization. In *ICML*, pages 25444–25470, Honolulu, HI, 2023.

[Nguyen and Grover, 2022] T. Nguyen and A. Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In *ICML*, pages 16569–16594, Baltimore, MD, 2022.

[Perrone and Shen, 2019] V. Perrone and H. Shen. Learning search spaces for Bayesian optimization: Another view of hyperparameter transfer learning. In *NeurIPS*, pages 12751–12761, Vancouver, Canada, 2019.

[Perrone *et al.*, 2018] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In *NeurIPS*, pages 6846–6856, Montreal, Canada, 2018.

[Poloczek *et al.*, 2016] M. Poloczek, J. Wang, and P. I. Frazier. Warm starting Bayesian optimization. In *WSC*, pages 770–781, Washington, DC, 2016.

[Radford *et al.*, 2018] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018.

[Real *et al.*, 2019] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, pages 4780–4789, Honolulu, HI, 2019.

[Shi *et al.*, 2023] Y. Shi, K. Xue, L. Song, and C. Qian. Macro placement by wire-mask-guided black-box optimization. In *NeurIPS*, pages 6825–6843, New Orleans, LA, 2023.

[Terayama *et al.*, 2021] K. Terayama, M. Sumita, R. Tamura, and K. Tsuda. Black-box optimization for automated discovery. *Accounts of Chemical Research*, 54(6):1334–1346, 2021.

[Vaswani *et al.*, 2017] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, Long Beach, Canada, 2017.

[Volpp *et al.*, 2020] M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. In *ICLR*, Addis Ababa, Ethiopia, 2020.

[Wang *et al.*, 2018] Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *AISTATS*, pages 745–754, Playa Blanca, Spain, 2018.

[Wang *et al.*, 2021] Z. Wang, G. E. Dahl, K. Swersky, C. Lee, Z. Nado, J. Gilmer, J. Snoek, and Z. Ghahramani. Pre-trained Gaussian processes for Bayesian optimization. *arXiv:2109.08215*, 2021.

[Wang *et al.*, 2024] S. Wang, K. Xue, L. Song, X. Huang, and C. Qian. Monte Carlo tree search based space transfer for black box optimization. In *NeurIPS*, pages 49591–49624, Vancouver, Canada, 2024.

[Wistuba and Grabocka, 2021] M. Wistuba and J. Grabocka. Few-shot Bayesian optimization with deep kernel surrogates. In *ICLR*, Virtual, 2021.

[Xue *et al.*, 2022] K. Xue, J. Xu, L. Yuan, M. Li, C. Qian, Z. Zhang, and Y. Yu. Multi-agent dynamic algorithm configuration. In *NeurIPS*, New Orleans, LA, 2022.

[Yang and Deb, 2010] X. Yang and S. Deb. Eagle strategy using Lévy walk and firefly algorithms for stochastic optimization. In *NICSO*, pages 101–111, Granada, Spain, 2010.