ConfigX: Modular Configuration for Evolutionary Algorithms via Multitask Reinforcement Learning

Hongshu Guo[1*], **Zeyuan Ma**[1*], **Jiacheng Chen**[1], **Yining Ma**[2], **Zhiguang Cao**[3], **Xinglin Zhang**[1], **Yue-Jiao Gong**[1†]

[1]South China University of Technology,
[2]Massachusetts Institute of Technology,
[3]Singapore Management University

Abstract

Recent advances in Meta-learning for Black-Box Optimization (MetaBBO) have shown the potential of using neural networks to dynamically configure evolutionary algorithms (EAs), enhancing their performance and adaptability across various BBO instances. However, they are often tailored to a specific EA, which limits their generalizability and necessitates retraining or redesigns for different EAs and optimization problems. To address this limitation, we introduce ConfigX, a new paradigm of the MetaBBO framework that is capable of learning a universal configuration agent (model) for boosting diverse EAs. To achieve so, our ConfigX first leverages a novel modularization system that enables the flexible combination of various optimization sub-modules to generate diverse EAs during training. Additionally, we propose a Transformer-based neural network to meta-learn a universal configuration policy through multitask reinforcement learning across a designed joint optimization task space. Extensive experiments verify that, our ConfigX, after large-scale pre-training, achieves robust zero-shot generalization to unseen tasks and outperforms state-of-the-art baselines. Moreover, ConfigX exhibits strong lifelong learning capabilities, allowing efficient adaptation to new tasks through fine-tuning. Our proposed ConfigX represents a significant step toward an automatic, all-purpose configuration agent for EAs.

1 Introduction

Over the decades, Evolutionary Algorithms (EAs) such as Genetic Algorithm (GA) (Holland 1992), Particle Swarm Optimization (PSO) (Kennedy and Eberhart 1995) and Differential Evolution (DE) (Storn and Price 1997) have been extensively researched to tackle challenging Black-Box Optimization (BBO) problems, where neither the mathematical formulation nor additional derivative information is accessible. On par with the development of EAs, one of the most crucial research avenues is the Automatic Configuration (AC) for EAs (Ansótegui, Sellmann, and Tierney 2009; Huang, Li, and Yao 2019). Generally speaking, AC for EAs aims at identifying the optimal configuration c^* from the configuration space $\mathcal{C}$ of an evolutionary algorithm A, across a set of BBO problem instances $\mathcal{I}$:

$$c^* = \arg\max_{c\in\mathcal{C}} \mathop{\mathbb{E}}_{p\in\mathcal{I}} [Perf(A, c, p)] \tag{1}$$

where $Perf()$ denotes the performance of a configuration for the algorithm under a given problem instance.

Traditionally, the primary research focus in AC for EAs has centered on human-crafted AC mechanisms. These mechanisms, including algorithm/operator selection (Fialho 2010) and parameter control (Aleti and Moser 2016), have demonstrated strong performance on well-known BBO benchmarks (Hansen et al. 2010), as well as in various eye-catching real-world scenarios such as Protein-Docking (Hwang et al. 2010), AutoML (Vanschoren et al. 2014), and Prompting Optimization of Large Language Models (Chen, Dohan, and So 2024). However, a major limitation of manual AC is its heavy reliance on deep expertise. To address a specific problem, one often needs to consult experts with the necessary experience to analyze the problem and then design appropriate AC mechanisms (as depicted in the top of Figure 1). This impedes the broader application of EAs in diverse scientific or industrial applications.

Recently, a novel paradigm called Meta-learning for Black-Box Optimization (MetaBBO) (Ma et al. 2023), has emerged in the learning-to-optimize community. MetaBBO aims to reduce the reliance on expert-level knowledge in designing more automated AC mechanisms. As shown in the middle of Figure 1, in MetaBBO, a neural network is meta-trained as a meta-level policy to maximize the expected performance (Eq. (1)) of a low-level algorithm by dictating suitable configuration for solving each problem instance. By leveraging the data-driven features of deep models and the generalization strengths of meta-learning (Finn, Abbeel, and Levine 2017) across a distribution of optimization problems, these MetaBBO methods (Ma et al. 2024c; Li et al. 2024; Song et al. 2024) have shown superior adaptability compared to traditional human-crafted AC baselines.

Despite these advancements, there remains significant potential to further reduce the expertise burden. Current MetaBBO methods often need custom neural network designs, specific learning objectives, and frequent retraining or even complete redesigns to fit different backbone EAs, overlooking the shared aspects of AC across multiple EAs. This

[*]These authors contributed equally.
[†]Corresponding author. (E-mail: gongyuejiao@gmail.com)

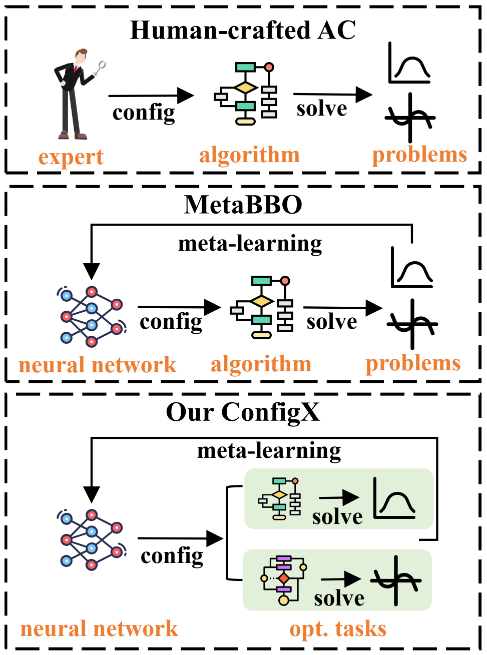

Figure 1: Conceptual overview of different AC paradigms.

leads to the core research question of this paper: *Is it possible to develop a MetaBBO paradigm that can meta-learn an automatic, all-purpose configuration agent for diverse EAs?* We outline the detailed research objective below:

$$c_k^* = \arg\max_{c \in \mathcal{C}_k} \mathop{\mathbb{E}}_{p \in \mathcal{I}} \left[Perf(A_k, c, p) \right], k = 1, 2, ..., K \quad (2)$$

where K is an exceedingly large number, potentially infinite. This objective is far more challenging since it can be regarded as the extension of Eq. (1). Concretely, it presents two key challenges: 1) **Constructing a comprehensive evolutionary algorithm space** is crucial for addressing Eq. (2), from which diverse EAs can be easily sampled for meta-training the MetaBBO; 2) **Ensuring the generalization capability** of the learned meta-level policy across not only optimization problems but also various EAs is imperative.

To address these challenges, we introduce ConfigX, a pioneering MetaBBO framework capable of modularly configuring diverse EAs with a single model across different optimization problems (as shown at the bottom of Figure 1).

Specifically, to address the first challenge, we present a novel modularization system for EAs, termed Modular-BBO in Section 3.1. It leverages hierarchical polymorphism to efficiently encapsulate and maintain various algorithmic sub-modules within the EAs, such as mutation or crossover operators. By flexibly combining these sub-modules, Modular-BBO can generate a vast array of distinct EA structures, hence spanning a comprehensive algorithm space $\mathcal{A}$. To address the second challenge, we combine the problem instance space $\mathcal{I}$ and $\mathcal{A}$ to form a joint optimization task space $\mathcal{T} : \mathcal{A} \times \mathcal{I}$. We then consider meta-learning a Transformer based meta-level policy over moderate optimization tasks sampled from the joint space $\mathcal{T}$ to maximize the objective in Eq. (2), see Section 3.2 and 3.3. For each task $T = (A_m, I_n)$, the Transformer generates configurations by conditioning on a sequence of state tokens corresponding to the sub-modules in A_m. Through large-scale multitask reinforcement learning over the sampled tasks, it yields a universal meta-policy that exhibits robust generalization to unseen algorithm structures and problem instances.

We summarize our contributions in this paper in three folds:

- We introduce ConfigX, the first MetaBBO framework to learn a pre-trained universal AC agent via multitask reinforcement learning, enabling modular configuration of diverse EAs across various optimization problems.
- Technically, we present Modular-BBO as a novel system for EA modularization that simplifies the management of sub-modules and facilitates the sampling of diverse algorithm structures. We then propose a Transformer-based architecture to meta-learn a universal configuration policy over our defined joint optimization task space.
- Extensive benchmark results show that the configuration policy pre-trained by ConfigX not only achieves superior zero-shot performance against the state-of-the-art AC software SMAC3, but also exhibits favorable lifelong learning capability via efficient fine-tuning.

2 Related Works

2.1 Human-crafted AC

Human-crafted AC mechanisms enhance the optimization robustness of EAs through two main paradigms: Operator Selection (OS) and Parameter Control (PC). OS is geared towards selecting proper evolutionary operators (i.e., mutation in DE (Qin and Suganthan 2005)) for EAs to solve target optimization problems. To this end, such AC mechanism requires preparing a group of candidate operators with diverse searching behaviours. Besides, throughout the optimization progress, OS facilitates dynamic selection over the candidate operators, either by a roulette wheel upon the historical success rates (Lynn and Suganthan 2017) or random replacement upon the immediate performance improvement (Mallipeddi et al. 2011). PC, on the other hand, aims at configuring (hyper-) parameters for the operators in EAs, (e.g. the inertia weights in PSO (Amoshahy, Shamsi, and Sedaaghi 2016) and the scale factors in DE (Zhang and Sanderson 2009; Tanabe and Fukunaga 2013)), while embracing similar adaptive mechanisms as OS to achieve dynamic configuration. We note that OS and PC are complementary rather than conflicting. Recent outperforming EAs such as MadDE (Biswas et al. 2021), AMCDE (Ye et al. 2023) and SAHLPSO (Tao et al. 2021) integrate both to obtain maximal performance gain. However, the construction of the candidate operators pool, the parameter value range in PC, and the adaptive mechanism in both of them heavily rely on expertise. A more versatile and efficient alternative for human-crafted AC is Bayesian Optimization (BO) (Shahriari et al. 2015). By iteratively updating and sampling from a posterior distribution over the algorithm configuration space, a recent open-source BO software SMAC3 (Lindauer et al. 2022) achieves the state-of-the-art AC performance on many realistic scenarios.

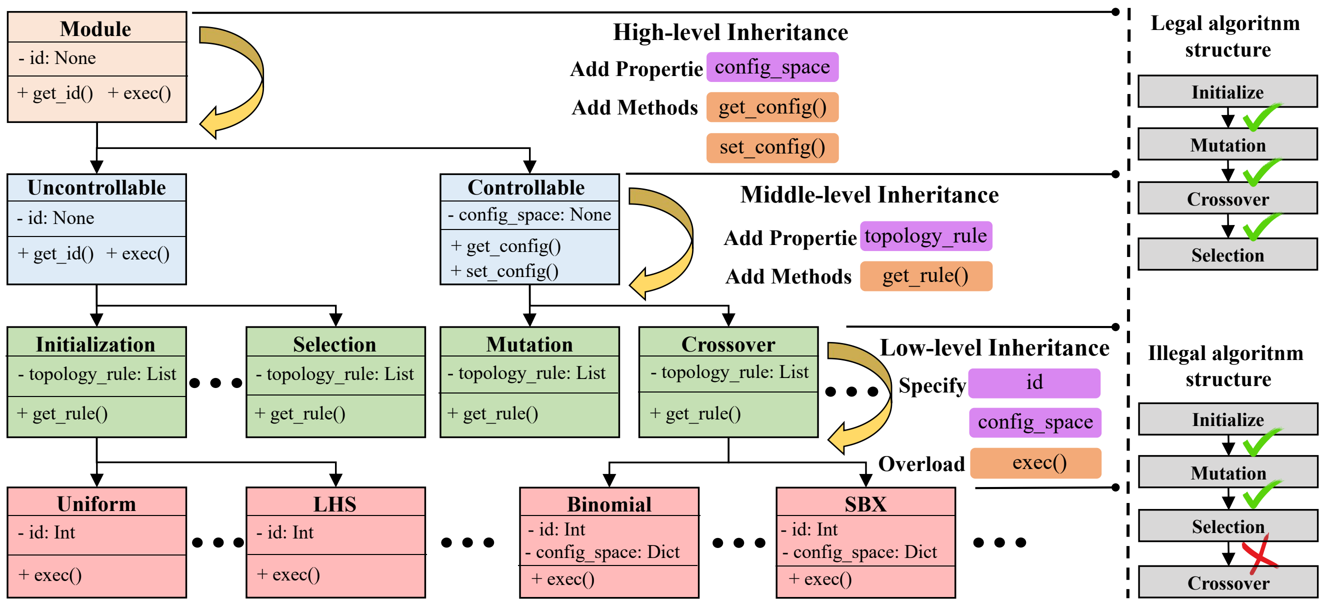

Figure 2: **Left**: The hierarchical polymorphism in Modular-BBO (we omit some properties and methods inherited from parent classes for better readability). **Right**: Legal/Illegal algorithm examples in Modular-BBO.

Algorithm 1: Algorithm Structure Generation.

Input: All accessible modules $\mathbb{M}$, all Initialization modules $\mathbb{M}_{\text{init}}$
Output: A legal algorithm structure A.

```
1:  Create an empty structure $A = \emptyset$, set index $j = 0$
2:  Randomly select an Initialization module from $\mathbb{M}_{\text{init}}$ as $a_j$
3:  $A \leftarrow A \bigcup a_j$
4:  while not COMPLETED do
5:      $j = j + 1$
6:      while VIOLATED do
7:          Randomly select a module from $\mathbb{M} \backslash \mathbb{M}_{\text{init}}$ as $a_j$
8:          Check the violation between $a_j$ and $a_{j-1}$
9:      end while
10:     $A \leftarrow A \bigcup a_j$
11: end while
```

2.2 MetaBBO

To relieve the expertise dependency of human-crafted AC, recent MetaBBO works introduce neural network-based control policy (typically denoted as the meta-level policy π_θ) to automatically dictate desired configuration for EAs (Ma et al. 2024d; Yang, Wang, and Li 2024). Generally speaking, the workflow of MetaBBO follows a bi-level optimization process: 1) At the meta level, the policy π_θ configures the low-level EA and assesses its performance, termed meta performance. The policy leverages this observed meta performance to refine its decision-making process, training itself through the maximization of accumulated meta performance, thereby advancing its meta objective. 2) At the lower level, the BBO algorithm receives a designated algorithmic configuration from the meta policy. With this configuration in hand, the low-level algorithm embarks on the task of optimizing the target objective. It observes the changes in the objective values and relays this information back to the meta optimizer, contributing to the meta performance signal. Similarly, existing MetaBBO works focus predominantly on OS and PC. Although a predefined operator group remains necessary, the selection decisions in works on OS (Sharma et al. 2019; Tan and Li 2021; Lian et al. 2024) are made by the meta policy π_θ which relieves the expert-level knowledge requirement. A notable example is RL-DAS (Guo et al. 2024) where advanced DE algorithms are switched entirely for complementary performance. In PC scenarios, initial works parameterize π with simple Multi-Layer Perceptron (MLP) (Wu and Wang 2022; Tan et al. 2022) and Long Short-Term Memory (LSTM) (Sun et al. 2021), whereas the latest work GLEET (Ma et al. 2024b) employs Transformer (Vaswani et al. 2017) architecture for more versatile configuration. Besides, works jointly configure both OS and PC such as MADAC (Xue et al. 2022) also show robust performance (Eimer et al. 2021).

3 Methodology

In this section, we elaborate on ConfigX step by step. We first explain in Section 3.1 the design of Modular-BBO and how to use it for efficient generation of diverse algorithm structures. We next provide a Markov Decision Process (MDP) definition of an optimization task and derive the corresponding multi-task learning objective in Section 3.2. At last, we introduce in Section 3.3 the details of each component in the MDP and the Transformer based architecture.

3.1 Modular-BBO

As illustrated in the left of Figure 2, the design philosophy of Modular-BBO adheres to a Hierarchical Polymorphism in *Python* which ensures the ease of maintaining different sub-modules (third-level sub-classes in Figure 2, labeled in green), as well as their practical variants (bottom-level sub-classes in Figure 2, labeled in red) in modern EAs. By facilitating the high-to-low level inheritances, Modular-BBO provides universal programming interfaces for the modularization of EAs, along with essential module-specific properties/methods to support diverse behaviours of various sub-modules. Further elaboration is provided below.

High-level. All sub-module classes in Modular-BBO stem from an abstract base class MODULE. It declares universal properties/interfaces shared among various sub-module variants, yet leave them void. At high-level inheritance, two sub-classes UNCONTROLLABLE and CONTROLLABLE inherit from MODULE. The two sub-classes divide all possible sub-modules in modern EAs into the ones with (hyper-) parameters and those without. For CONTROLLABLE modules, we declare its (hyper-) parameters by adding a *config_space* property. Additionally, we include the corresponding *get_config*() and *set_config*() methods for configuring the (hyper-) parameters. Currently, these properties and methods remain void until a specific EA sub-module is instantiated.

Middle-level. At this inheritance level, UNCONTROLLABLE and CONTROLLABLE are further divided into common sub-modules in EAs, e.g., initialization, mutation, selection and etc. To combine these sub-modules legally and generate legal algorithm structures, we introduce module-specific *topology_rule* as a guidance during the generating process (Algorithm 1), by invoking the added *get_rule*() method. We present a pair of examples in the left of Figure 2 to showcase one of the possible violation during the algorithm structure generation, where CROSSOVER is not allowed after SELECTION is a common sense in EAs.

Low-level. Within the low-level inheritance, we borrow from a large body of EA literature diverse practical sub-module variants (i.e., lots of initialization strategy have been proposed in literature such as Sobol sampling (Sobol 1967) and LHS sampling (McKay, Beckman, and Conover 2000)) and maintaining them by inheriting from the sub-module classes in middle-level inheritance. When inheriting from the parent class, a concrete sub-module variant has to instantiate void modules *id* and *config_space*, which detail its unique identifier in Modular-BBO and controllable parameters respectively. It also have to overload *exec*() method by which it operates the solution population. The unique module id of a sub-module variant is a 16-bit binary code of which: 1) the first bit is 0 or 1 to denote if this variant is UNCONTROLLABLE or CONTROLLABLE. 2) the 2-nd to 7-th bits denote the sub-module category (third-level sub-classes in Figure 2, labeled in green) to which the variant belongs. 3) the last 9 bits denotes its id within this sub-module category.

For now, Modular-BBO has included 11 common sub-module categories in EAs: INITIALIZATION (Kazimipour, Li, and Qin 2014), MUTATION (Das, Mullick, and Suganthan 2016), CROSSOVER (Spears 1995), PSO_UPDATE (Shami et al. 2022), BOUNDARY_CONTROL (Kadavy et al. 2023), SELECTION (Shukla, Pandey, and Mehrotra 2015), MULTI_STRATEGY (Gong et al. 2011), NICHING (Ma et al. 2019), INFORMATION_SHARING (Toulouse, Crainic, and Gendreau 1996), RESTART_STRATEGY (Jansen 2002), POPULATION_REDUCTION (Pool and Nielsen 2007). We construct a collection of over 100 variants for these sub-module categories from a large body of literature and denote this collection as module space $\mathbb{M}$. Theoretically, by using the algorithm generation process described in Algorithm 1, Modular-BBO spans a massive algorithm structure space $\mathcal{A}$ containing millions of algorithm structures. Due to the limitation of space, we provide the detail of each sub-module variant in $\mathbb{M}$ in Appendix A, Table 1, including the id, name, type, configuration space, topology rule and functional description. We also provide a detailed explanation for Algorithm 1 in Appendix B.

3.2 Multi-task Learning in ConfigX

Optimization Task Space We first define an optimization task space $\mathcal{T}$ as a synergy of an algorithm space $\mathcal{A}$ and an optimization problem set $\mathcal{I}$. Then an optimization task $T \in \mathcal{T}$ can be defined as $T : \{A \in \mathcal{A}, p \in \mathcal{I}\}$. In this paper, we adopt the algorithm space spanned by Modular-BBO as $\mathcal{A}$, the problem instances from well-known CoCo-BBOB (Hansen et al. 2010), Protein-docking (Hwang et al. 2010) and HPO-B benchmark (Arango et al. 2021) as $\mathcal{I}$.

AC as an MDP For an optimization task $T : \{A, p\}$, we facilitate a Transformer based policy π_θ (detailed in Section 3.3) to dynamically dictate desired configuration for A to solve p. This configuration process can be formulated as an Markov Decision Process (MDP) $\mathcal{M} := (S, C, \Gamma, R, H, \gamma)$, where S denotes the state space that reflect optimization status, C denotes the action space which is exactly the configuration space of algorithm A, $\Gamma(s_{t+1}|s_t, c_t)$ denotes the optimization transition dynamics, $R(s_t, c_t)$ measures the single step optimization improvement obtained by using configuration c_t for optimizing p. H and γ are the number of optimization iterations and discount factor respectively. At each optimization step t, the policy π_θ receives a state s_t and then outputs a configuration $c_t = \pi_\theta(s_t)$ for A. Using c_t, algorithm A optimizes the optimization problem p for a single step. The goal is to find an optimal policy π_{θ^*} which maximizes the accumulated optimization improvement during the optimization process: $G = \sum_{t=1}^{H} \gamma^{t-1} R(s_t, c_t)$. Recall that our ConfigX aims at addressing a more challenging objective in Eq. (2), where the goal is to maximize the accumulated optimization improvement G of all tasks $T \in \mathcal{T}$. We use s_t^i and c_t^i to denote the input state and the outputted configuration of the policy π_θ for the i-th task in $\mathcal{T}$. Then the objective in Eq. (2) can be rewritten as a multi-task RL problem:

$$\mathbb{J}(\theta) = \frac{1}{K \cdot N} \sum_{i=1}^{K \cdot N} \sum_{t=1}^{H} \gamma^{t-1} R(s_t^i, c_t^i) \tag{3}$$

where we sample $K \cdot N$ tasks from $\mathcal{T}$ to train π_θ since the number of tasks in $\mathcal{T}$ is massive. These tasks is sampled first by calling Algorithm 1 K times to obtain K algorithm structures, and then combine these algorithm with the N problem instances in $\mathcal{I}$. In this paper we use Proximal Policy Opti-

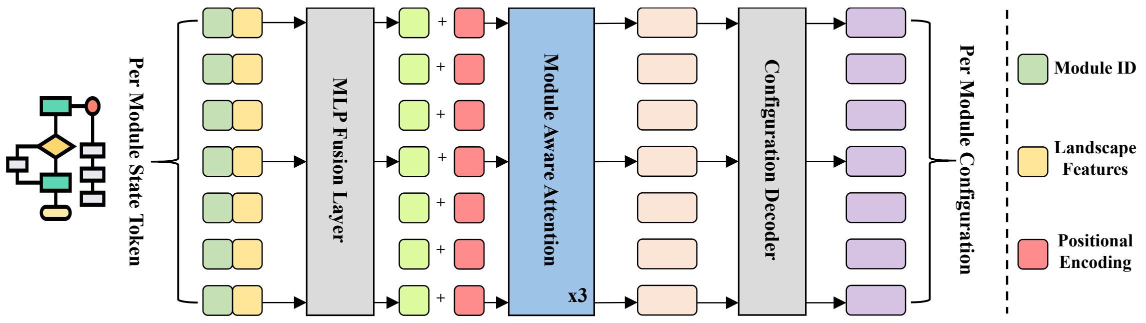

Figure 3: The workflow of the Transformer based configuration policy in ConfigX.

mization (PPO) (Schulman et al. 2017), a popular policy gradient (Williams 1992) method for optimizing this objective in a joint policy optimization (Gupta et al. 2022) fashion. We include the pseudocode of the RL training in Appendix D.

3.3 ConfigX

Progress in MetaBBO has made it possible to meta-learn neural network-based control policies for configuring the backbone EAs to solve optimization problems. However, existing MetaBBO methods are not suitable for the massive algorithm structure space $\mathcal{A}$ spanned by the module space $\mathbb{M}$ in our proposed Modular-BBO, since learning a separate policy for each algorithm structure is impractical. However, the modular nature of EAs implies that while each structure is unique, they are still constructed from the same module space and potentially shares sub-modules and workflows with other algorithm structures. We now describe how ConfigX exploits this insight to address the challenge of learning a universal controller for different algorithm structures.

State Design In ConfigX, we encode not only the algorithm structure information but also the optimization status information into the state representation to ensure the generalization across optimization tasks. Concretely, as illustrated in the left of Figure 3, for i-th tasks $T_i : \{A_i, p_i\}$ in the sampled $K \cdot N$ tasks, we encode a information pair for each sub-module in A_i, e.g., $s_i : \{s^{\text{id}}_{i,j}, s^{\text{opt}}_i\}_{j=1}^{L_i}$, where $s^{\text{id}}_{i,j} \in \{0,1\}^{16}$ denotes the unique module id for j-th sub-module in A_i, $s^{\text{opt}}_i \in \mathbb{R}^9$ denotes the algorithm performance information which we borrow the idea from recent MetaBBO methods (Guo et al. 2024; Ma et al. 2024a), L_i denotes the number of sub-modules in A_i. We provide details of these information pairs in Appendix C.

State Encode We apply an MLP fusion layer to preprocess the state representation s_i. This fusion process ensures the information within the information pair $\{s^{\text{id}}_{i,j}, s^{\text{opt}}_i\}$ join each other smoothly (as illustrated in left part of Figure 3).

$$\begin{aligned} \hat{e}_{i,j} &= \text{hstack}(\phi(s^{\text{id}}_{i,j}; \mathbf{W}^{\text{id}}_e); \phi(s^{\text{opt}}_i; \mathbf{W}^{\text{opt}}_e)) \\ e_{i,j} &= \phi(\hat{e}_{i,j}; \mathbf{W}_e), \quad j = 1, ..., L_i \end{aligned} \tag{4}$$

Where $\phi(\cdot; \mathbf{W}^{\text{id}}_e)$, $\phi(\cdot; \mathbf{W}^{\text{opt}}_e)$ and $\phi(\cdot; \mathbf{W}_e)$ denotes MLP layers with the shape of 16×16, 9×16 and 32×64 respectively, $e_{i,j}$ denotes the fused information for each sub-module. Then we add *Sin* Positional Encoding (Vaswani et al. 2017) $\mathbf{W}_{\text{pos}}$ to each sub-module, which represents the relative position information among all sub-modules in the algorithm structure.

$$\mathbf{h}^{(0)}_i = \text{vstack}(e_{i,j}; \cdots; e_{i,L_i}) + \mathbf{W}_{\text{pos}} \tag{5}$$

where $\mathbf{h}^{(0)}_i \in \mathbb{R}^{L_{\max} \times 64}$ denotes the module embedding for each sub-module. We note that since the number of sub-modules (L_i) may vary between different algorithm structures, we zero pad $\mathbf{h}^{(0)}_i$ to a pre-defined maximum length $L_{\max}$ to ensure input size invariant among tasks.

Module Aware Attention From the module embeddings $\mathbf{h}^{(0)}_i$ described above, we obtains the output features for all sub-modules as:

$$\begin{aligned} \hat{\mathbf{h}}^{(l)}_i &= \text{LN}(\text{MSA}(\mathbf{h}^{(l-1)}_i) + \mathbf{h}^{(l-1)}_i), l = 1, 2, 3 \\ \mathbf{h}^{(l)}_i &= \text{LN}(\phi(\hat{\mathbf{h}}_i; \mathbf{W}^{(l)}_F)) + \hat{\mathbf{h}}^{(l)}_i), l = 1, 2, 3 \end{aligned} \tag{6}$$

where LN is Layernorm (Ba, Kiros, and Hinton 2016), MSA is Multi-head Self-Attention (Vaswani et al. 2017) and $\phi(\cdot; \mathbf{W}^{(l)}_F)$ are MLP layers with the shape of 64×64. In this paper we use $l = 3$ MSA blocks to process the module embeddings (as illustrated in the middle of Figure 3).

Configuration Decoder In ConfigX, the policy $\pi_\theta(c_i|s_i)$ models the conditional distribution of A_i's configuration c_i given the state s_i. As illustrated in the right of Figure 3, for each sub-module a_j in an algorithm $A_i = \{a_1, a_2, ...\}$, we output distribution parameters μ and Σ as:

$$\begin{aligned} \mu_j = \phi(h^{(3)}_{i,j}; \mathbf{W}_\mu), \quad \Sigma_j = \text{Diag}\phi(h^{(3)}_{i,j}; \mathbf{W}_\Sigma) \\ c_{i,j} \sim \mathcal{N}(\mu_j; \Sigma_j) \end{aligned} \tag{7}$$

where $\phi(\cdot; \mathbf{W}_\mu)$ and $\phi(\cdot; \mathbf{W}_\Sigma)$ are two MLP layers with the same shape of $64 \times C_{\max}$, $c_{i,j} \in \mathbb{R}^{C_{\max}}$ denotes the configurations for sub-module a_j in algorithm structure A_i. Since the size of the configuration spaces may vary between different sub-modules, we pre-defined a maximum configuration space size $C_{\max}$ to cover the sizes of all sub-modules. If the size of a sub-module is less than $C_{\max}$, we use the first few configurations in $c_{i,j}$ and ignore the rest.

For the critic, we calculate the value of a sub-module as $V(s_{i,j}) = \phi(\mathbf{h}^{(3)}_{i,j}; \mathbf{W}_c)$ using a MLP with the shape of $\mathbf{W}_c \in \mathbb{R}^{64 \times 16 \times 1}$. The value of the algorithm structure is the averaged value per sub-module $V(s_i) = \frac{1}{L_{\max}} \sum_{j=1}^{L_{\max}} V(s_{i,j})$.

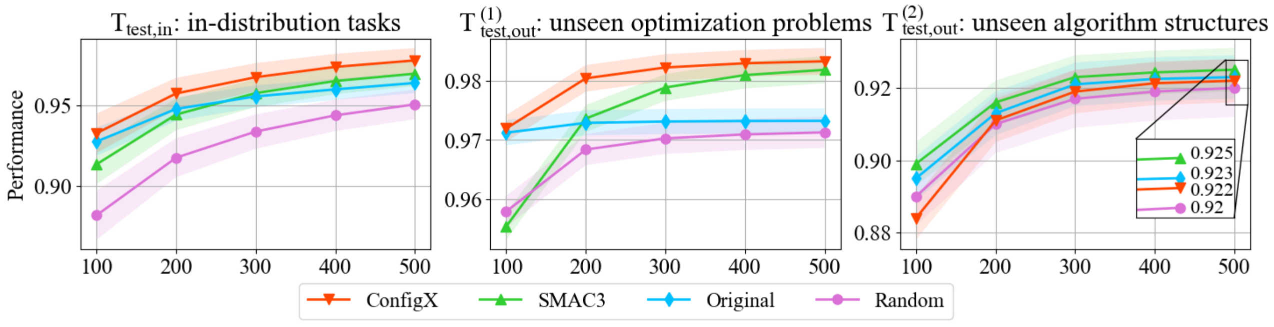

Figure 4: Optimization curves of the pre-trained ConfigX model and the baselines, over three different zero-shot scenarios.

Reward Function The objective value scales across different problem instances can vary. To ensure the accumulated performance improvement across tasks approximately share the same numerical level, we propose a task agnostic reward function. At optimization step t, the reward function on any problem instance $p \in \mathcal{I}$ is formulated as:

$$r_t = \delta \times \frac{f^*_{p,t-1} - f^*_{p,t}}{f^*_{p,0} - f^*_p} \tag{8}$$

where $f^*_{p,t}$ is the found best objective value of problem instance p at step t, f^*_p is the global optimal objective value of p and $\delta = 10$ is a scale factor. In this way, we make the scales of the accumulated improvement in all tasks similar and hence stabilize the training.

4 Experiment

In this section, we discuss the following research questions: **RQ1**: Can pre-trained ConfigX model zero-shots to unseen tasks with unseen algorithm structures and/or unseen problem instances? **RQ2**: If the zero-shot performance is not as expected, is it possible to fine-tune ConfigX to address novel algorithm structures in future? **RQ3**: How do the concrete designs in ConfigX contribute to the learning effectiveness? Below, we first introduce the experimental settings and then address RQ1$\sim$RQ3 respectively.

4.1 Experimental Setup

Training setup. We have prepared several task sets from different sub-task-spaces of the overall task space $\mathcal{T}$ (defined at Section 3.2) to aid for the following experimental validation. Concretely, denote $\mathcal{I}_{\text{syn}}$ as the problems in CoCo-BBOB suite, $\mathcal{I}_{\text{real}}$ as all realistic problems in Protein-docking benchmark and HPO-B benchmark, $\mathcal{A}_{\text{DE}}$ as the algorithm structure space only including DE variants, $\mathcal{A}_{\text{PSO,GA}}$ as the algorithm structure space including PSO and GA variants, we have prepared 256 optimization tasks as training task set $T_{\text{train}} \subset \mathcal{A}_{\text{DE}} \times \mathcal{I}_{\text{syn}}$, another 512 optimization tasks as in-distribution testing task set $T_{\text{test,in}} \subset \mathcal{A}_{\text{DE}} \times \mathcal{I}_{\text{syn}}$. For out-of-distribution tasks, we have prepared two task sets: $T^{(1)}_{\text{test,out}} \subset \mathcal{A}_{\text{DE}} \times \mathcal{I}_{\text{real}}$ and $T^{(2)}_{\text{test,out}} \subset \mathcal{A}_{\text{PSO,GA}} \times \mathcal{I}_{\text{syn}}$, each with 512 task instances. During the training, for a batch of $batch\_size = 32$ tasks, PPO (Schulman et al. 2017) method is used to update the policy net and critic $\kappa = 3$ times for every 10 rollout optimization steps. All of the tasks are allowed to be optimized for $H = 500$ optimization steps. The training lasts for 50 epochs with a fixed learning rate 0.001. All experiments are run on an Intel(R) Xeon(R) 6348 CPU with 504G RAM. Refer to Appendix E.1 for more details.

Baselines and Performance Metric. In the following comparisons, we consider three baselines: **SMAC3**, which is the state-of-the-art AC software based on Bayesian Optimization and aggressive racing mechanism; **Original**, which denotes using the suggested configurations in sub-modules' original paper (see Appendix A for one-to-one correspondence); **Random**, which randomly sample the configurations for the algorithm from the algorithm's configuration space. For the pre-trained model in ConfigX and the above baselines, we calculate the performance of them on tested task set by applying them to configure each tested task for 51 independent runs and then aggregate a normalized accumulated optimization improvement across all tasks and all runs, we provide more detailed calculation steps in Appendix E.2.

4.2 Zero-shot Performance (RQ1)

We validate the zero-shot performance of ConfigX by first pre-training a model on T_{train}. Then the pre-trained model is directly used to facilitate AC process for tasks in tested set, without any fine-tuning. Concretely, we aims at validating the zero-shot generalization performance in three different scenarios: 1) $T_{\text{test,in}}$, where the optimization tasks come from the same task space on which ConfigX is pre-trained. 2) $T^{(1)}_{\text{test,out}}$, where the optimization tasks locate beyond the optimization problem scope of the training task space. 3) $T^{(2)}_{\text{test,out}}$, where the optimization tasks locate beyond the algorithm structure scope of the training task space. We present the optimization curves of our pre-trained model and the baselines in Figure 4, where the x-axis denotes the optimization horizon and y-axis denotes the performance metric we defined previously. The results in Figure 4 reveal several key observations: 1) In all zero-shot scenarios, ConfigX presents significantly superior performance to the Random baseline, which randomly configures the algorithms in the tested tasks. This underscores the effectiveness of the multi-task reinforcement learning in ConfigX. 2) The results on $T_{\text{test,in}}$ demonstrate that pre-training ConfigX on some task samples of the given task space is enough to ensure the gen-

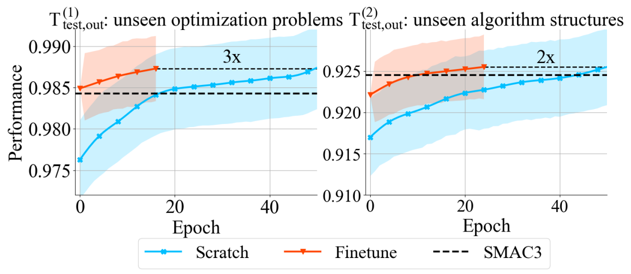

Figure 5: The learning curves of fine-tuning and re-training ConfigX on novel optimization problems or algorithm structures. The fine-tuning saves 3x and 2x learning steps than the re-training on $T^{(1)}_{\text{test,out}}$ and $T^{(2)}_{\text{test,out}}$ respectively.

	$T_{\text{test,in}}$	$T^{(1)}_{\text{test,out}}$	$T^{(2)}_{\text{test,out}}$
ConfigX	9.81E-01 ±7.33E-03	**9.86E-01 ±2.64E-03**	**9.22E-01 ±6.94E-03**
ConfigX-MLP	9.70E-01 ±8.13E-03	9.80E-01 ±2.54E-03	9.16E-01 ±6.57E-03
ConfigX-LPE	**9.82E-01 ±7.62E-03**	9.84E-01 ±2.58E-03	9.20E-01 ±6.89E-03
ConfigX-NPE	9.74E-01 ±7.75E-03	9.81E-01 ±2.67E-03	9.19E-01 ±6.73E-03
MLP-NPE	9.51E-01 ±9.27E-03	9.73E-01 ±2.71E-03	9.06E-01 ±7.29E-03

Table 1: Performance of different ablated baselines.

eralization to the other tasks within this space, surpassing the state-of-the-art AC baseline SMAC3. 3) The results on $T^{(1)}_{\text{test,out}}$ show that ConfigX is capable of adapting itself to totally unseen optimization problem scope. This observation attributes to our state representation design, where the optimization status borrowed from recent MetaBBO works are claimed to be generic across different problem scopes. 4) Though promising, we find that the zero-shot performance of ConfigX on $T^{(2)}_{\text{test,out}}$ is not as expected. It is not surprising as the sub-modules and structures in GA/PSO are significantly different from those in DE, which hinders ConfigX from applying its DE configuration experience on PSO/GA tasks. We explore whether this generalization gap could be addressed through further fine-tuning in the next section.

4.3 Lifelong Learning in ConfigX (RQ2)

The booming algorithm designs in EAs, together with the increasingly diverse optimization problems pose non-negligible challenges to universal algorithm configuration methods such as our ConfigX. On the one hand, although our pre-trained model shows uncommon AC performance when encountered with novel optimization problems (middle of Figure 4), further performance boost is still needed especially in industrial scenarios. On the other hand, as shown in the left of Figure 4, the pre-trained model can not cover those algorithm sub-modules which have not been included within its training algorithm structure space. Both situations above underline the importance of lifelong learning in ConfigX. We hence investigate the fine-tuning efficiency of the pre-trained model in this section. Concretely, we compare the learning curves of 1) fine-tuning the pre-trained model, and 2) re-training a new model from scratch in Figure 5, where the x-axis denotes the learning epochs and the y-axis denotes the aforementioned performance metric over the tested task set. The results reveal that ConfigX supports efficient fine-tuning for adapting out-of-distribution tasks, which in turn provides operable guidance for lifelong learning in ConfigX: (a) One can configure an algorithm included in the algorithm space of Modular-BBO, yet on different problem scope, by directly using the pre-trained model. (b) One can also integrate novel algorithm designs into Modular-BBO and facilitate efficient fine-tuning to enhance the performance of the pre-trained model on these algorithm structures.

4.4 Ablation Study (RQ3)

In Section 3.3, we proposed a Transformer based architecture to encode and process the state information of all sub-modules within an algorithm structure. In particular, we added Sin positional embeddings (PE) to each sub-module token as additional topology structure information for learning. We further apply Multi-head Self-Attention (MSA) to enhance the module-aware information sharing. In this section we investigate on what extent these designs influence ConfigX's learning effectiveness. Concretely, for the positional embeddings, we introduce two ablations 1) ConfigX-NPE: remove the Sin PE from ConfigX. 2) ConfigX-LPE, replace the Sin PE by Learnable PE (Gehring et al. 2017). For the MSA, we introduce one ablation ConfigX-MLP: cancel the information sharing between the sub-modules by replacing the MSA blocks by an MLP layer. We present the final performance of these baselines and ConfigX on the tested task sets in Table 1. The results underscores the importance of these special designs: (a) Without the MSA block, ConfigX struggles in learning the configuration policy in an informative way. (b) Without the positional embdeddings, the configuration policy in ConfigX becomes agnostic to the structure information of the controlled algorithm. (c) Learnable PE shows similar performance with Sin PE, while introducing additional parameters for ConfigX to learn.

5 Conclusion

In this paper, we propose ConfigX as a pioneer research exploring the possibility of learning a universal MetaBBO agent for automatically configuring diverse EAs across optimization problems. To this end, we first introduce a novel EA modularization system Modular-BBO that is capable of maintaining various sub-modules in EAs and spanning a massive algorithm structure space. We then formulate the universal AC over this algorithm space as an MTRL problem and hence propose meta-learning a Transformer based configuration policy to maximize the overall optimization performance across task samples. Extensive experiments demonstrate that a pre-trained ConfigX model could achieve superior AC performance to the state-of-the-art manual AC method SMAC3. Furthermore, we verify that ConfigX holds promising lifelong learning ability when being fine-tuned to adapt out-of-scope algorithm structures and optimization problems. We hope this work could serve as a pivotal step towards automatic and all-purpose AC base model.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62276100), in part by the Guangdong Natural Science Funds for Distinguished Young Scholars (Grant No. 2022B1515020049), in part by the Guangdong Provincial Natural Science Foundation for Outstanding Youth Team Project (Grant No. 2024B1515040010), in part by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No. AISG3-RP-2022-031), and in part by the TCL Young Scholars Program.

References

Aleti, A.; and Moser, I. 2016. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Comput. Surv.*

Amoshahy, M. J.; Shamsi, M.; and Sedaaghi, M. H. 2016. A novel flexible inertia weight particle swarm optimization algorithm. *PloS one*, 11(8): e0161558.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *CP*.

Arango, S. P.; 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*.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. In *NeurIPS*.

Biswas, S.; Saha, D.; De, S.; Cobb, A. D.; Das, S.; and Jalaian, B. A. 2021. Improving differential evolution through Bayesian hyperparameter optimization. In *CEC*, 832–840.

Chen, A.; Dohan, D.; and So, D. 2024. EvoPrompting: language models for code-level neural architecture search. *NeurIPS*, 36.

Das, S.; Mullick, S. S.; and Suganthan, P. N. 2016. Recent advances in differential evolution–an updated survey. *Swarm Evol. Comput.*, 27: 1–30.

Eimer, T.; Biedenkapp, A.; Reimer, M.; Adriaensen, S.; Hutter, F.; and Lindauer, M. 2021. DACBench: A benchmark library for dynamic algorithm configuration. *arXiv preprint arXiv:2105.08541*.

Fialho, Á. 2010. *Adaptive operator selection for optimization*. Ph.D. thesis, Université Paris Sud-Paris XI.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; and Dauphin, Y. N. 2017. Convolutional sequence to sequence learning. In *ICML*, 1243–1252. PMLR.

Gong, W.; Fialho, A.; Cai, Z.; and Li, H. 2011. Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. *Inf. Sci.*, 181(24): 5364–5386.

Guo, H.; Ma, Y.; Ma, Z.; Chen, J.; Zhang, X.; Cao, Z.; Zhang, J.; and Gong, Y.-J. 2024. Deep Reinforcement Learning for Dynamic Algorithm Selection: A Proof-of-Principle Study on Differential Evolution. *TSMC*.

Gupta, A.; Fan, L.; Ganguli, S.; and Fei-Fei, L. 2022. Metamorph: learning universal controllers with transformers. In *ICLR*. ICLR.

Hansen, N.; Auger, A.; Finck, S.; and Ros, R. 2010. *Real-parameter black-box optimization benchmarking 2010: Experimental setup*. Ph.D. thesis, INRIA.

Holland, J. H. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Huang, C.; Li, Y.; and Yao, X. 2019. A survey of automatic parameter tuning methods for metaheuristics. *TEC*.

Hwang, H.; Vreven, T.; Janin, J.; and Weng, Z. 2010. Protein–protein docking benchmark version 4.0. *Proteins: Structure, Function, and Bioinformatics*.

Jansen, T. 2002. On the analysis of dynamic restart strategies for evolutionary algorithms. In *PPSN*, 33–43.

Kadavy, T.; Viktorin, A.; Kazikova, A.; Pluhacek, M.; and Senkerik, R. 2023. Impact of boundary control methods on bound-constrained optimization benchmarking. In *GECCO*, 25–26.

Kazimipour, B.; Li, X.; and Qin, A. K. 2014. A review of population initialization techniques for evolutionary algorithms. In *CEC*, 2585–2592. IEEE.

Kennedy, J.; and Eberhart, R. 1995. Particle swarm optimization. In *ICNN*, volume 4, 1942–1948. IEEE.

Li, X.; Wu, K.; Li, Y. B.; Zhang, X.; Wang, H.; and Liu, J. 2024. GLHF: General Learned Evolutionary Algorithm Via Hyper Functions. *arXiv preprint arXiv:2405.03728*.

Lian, H.; Ma, Z.; Guo, H.; Huang, T.; and Gong, Y.-J. 2024. RLEMMO: Evolutionary Multimodal Optimization Assisted By Deep Reinforcement Learning. In *GECCO*, 683–693.

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhkopf, T.; Sass, R.; and Hutter, F. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *JMLR*, 23(54): 1–9.

Lynn, N.; and Suganthan, P. N. 2017. Ensemble particle swarm optimizer. *Appl. Soft Comput.*, 55: 533–548.

Ma, H.; Shen, S.; Yu, M.; Yang, Z.; Fei, M.; and Zhou, H. 2019. Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey. *Swarm Evol. Comput.*, 44: 365–387.

Ma, Z.; Chen, J.; Guo, H.; and Gong, Y.-J. 2024a. Neural exploratory landscape analysis. *arXiv preprint arXiv:2408.10672*.

Ma, Z.; Chen, J.; Guo, H.; Ma, Y.; and Gong, Y.-J. 2024b. Auto-configuring Exploration-Exploitation Tradeoff in Evolutionary Computation via Deep Reinforcement Learning. In *GECCO*, 1497–1505.

Ma, Z.; Guo, H.; Chen, J.; Li, Z.; Peng, G.; Gong, Y.-J.; Ma, Y.; and Cao, Z. 2023. MetaBox: A Benchmark Platform for Meta-Black-Box Optimization with Reinforcement Learning. In *NeurIPS*, volume 36.

Ma, Z.; Guo, H.; Chen, J.; Peng, G.; Cao, Z.; Ma, Y.; and Gong, Y.-J. 2024c. LLaMoCo: Instruction Tuning of Large Language Models for Optimization Code Generation. *arXiv preprint arXiv:2403.01131*.

Ma, Z.; Guo, H.; Gong, Y.-J.; Zhang, J.; and Tan, K. C. 2024d. Toward Automated Algorithm Design: A Survey and Practical Guide to Meta-Black-Box-Optimization. *arXiv preprint arXiv:2411.00625*.

Mallipeddi, R.; Suganthan, P. N.; Pan, Q.-K.; and Tasgetiren, M. F. 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Appl. Soft Comput.*, 11(2): 1679–1696.

McKay, M. D.; Beckman, R. J.; and Conover, W. J. 2000. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1): 55–61.

Pool, J. E.; and Nielsen, R. 2007. Population size changes reshape genomic patterns of diversity. *Evolution*, 61(12): 3001–3006.

Qin, A. K.; and Suganthan, P. N. 2005. Self-adaptive differential evolution algorithm for numerical optimization. In *CEC*, volume 2, 1785–1791. IEEE.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175.

Shami, T. M.; El-Saleh, A. A.; Alswaitti, M.; Al-Tashi, Q.; Summakieh, M. A.; and Mirjalili, S. 2022. Particle swarm optimization: A comprehensive survey. *IEEE Access*, 10: 10031–10061.

Sharma, M.; Komninos, A.; López-Ibáñez, M.; and Kazakov, D. 2019. Deep reinforcement learning based parameter control in differential evolution. In *GECCO*, 709–717.

Shukla, A.; Pandey, H. M.; and Mehrotra, D. 2015. Comparative review of selection techniques in genetic algorithm. In *ABLAZE*, 515–519. IEEE.

Sobol, I. 1967. The distribution of points in a cube and the accurate evaluation of integrals (in Russian) Zh. *Vychisl. Mat. i Mater. Phys*, 7: 784–802.

Song, L.; Gao, C.; Xue, K.; Wu, C.; Li, D.; Hao, J.; Zhang, Z.; and Qian, C. 2024. Reinforced In-Context Black-Box Optimization. *arXiv preprint arXiv:2402.17423*.

Spears, W. M. 1995. Adapting Crossover in Evolutionary Algorithms. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 367.

Storn, R.; and Price, K. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.*, 11: 341–359.

Sun, J.; Liu, X.; Bäck, T.; and Xu, Z. 2021. Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. *TEC*, 25(4): 666–680.

Tan, Z.; and Li, K. 2021. Differential evolution with mixed mutation strategy based on deep reinforcement learning. *Appl. Soft Comput.*, 111: 107678.

Tan, Z.; Tang, Y.; Li, K.; Huang, H.; and Luo, S. 2022. Differential evolution with hybrid parameters and mutation strategies based on reinforcement learning. *Swarm Evol. Comput.*

Tanabe, R.; and Fukunaga, A. 2013. Success-history based parameter adaptation for differential evolution. In *CEC*, 71–78. IEEE.

Tao, X.; Li, X.; Chen, W.; Liang, T.; Li, Y.; Guo, J.; and Qi, L. 2021. Self-Adaptive two roles hybrid learning strategies-based particle swarm optimization. *Inf. Sci.*, 578: 457–481.

Toulouse, M.; Crainic, T. G.; and Gendreau, M. 1996. *Communication issues in designing cooperative multi-thread parallel searches*. Springer.

Vanschoren, J.; Van Rijn, J. N.; Bischl, B.; and Torgo, L. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2): 49–60.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *NeurIPS*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Wu, D.; and Wang, G. G. 2022. Employing reinforcement learning to enhance particle swarm optimization methods. *Engineering Optimization*, 54(2): 329–348.

Xue, K.; Xu, J.; Yuan, L.; Li, M.; Qian, C.; Zhang, Z.; and Yu, Y. 2022. Multi-agent dynamic algorithm configuration. *NeurIPS*, 35: 20147–20161.

Yang, X.; Wang, R.; and Li, K. 2024. Meta-Black-Box Optimization for Evolutionary Algorithms: Review and Perspective. *Available at SSRN 4956956*.

Ye, C.; Li, C.; Li, Y.; Sun, Y.; Yang, W.; Bai, M.; Zhu, X.; Hu, J.; Chi, T.; Zhu, H.; et al. 2023. Differential evolution with alternation between steady monopoly and transient competition of mutation strategies. *Swarm Evol. Comput.*, 83: 101403.

Zhang, J.; and Sanderson, A. C. 2009. JADE: adaptive differential evolution with optional external archive. *TEC*, 13(5): 945–958.