META-BLACK-BOX-OPTIMIZATION THROUGH OFFLINE Q-FUNCTION LEARNING WITH MAMBA ARCHITECTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent progress in Meta-Black-Box-Optimization (MetaBBO) has demonstrated that meta-training a neural network based meta-level control policy over an optimization task distribution could significantly enhance the optimization performance of the low-level black-box optimizers. However, achieving such performance enhancement requires effective policy optimization/search method to locate optimal control policy within a massive joint-action space. The online learning fashion of existing works further makes the efficiency of MetaBBO problematic. To address these technical challenges, we propose an offline learning framework in this paper, termed Q-Mamba. Concretely, our method uses a Mamba neural network architecture to meta-learn decomposed Q-functions for each configurable component in the low-level optimizer. By decomposing the Q-function of the configuration decisions of all components in an optimizer, we can apply effective sequence modelling to avoid searching the control policy in the massive joint-action space. Furthermore, by leveraging the long-sequence modelling advantage of Mamba and moderate offline trajectory samples, Q-Mamba can be efficiently trained through a synergy of offline Temporal-Difference update and Conservative Q-Learning regularization to achieve competitive performance against the online learning paradigms. Through extensive benchmarking, we observe that Q-Mamba achieves competitive or even superior optimization performance to prior online/offline learning baselines, while significantly improving the training efficiency of existing online learning baselines. Additional ablation studies show that each of the proposed key designs contributes to this good performance.

1 INTRODUCTION

Optimization is everywhere. When it comes to the Black-Box Optimization (BBO), where neither the problem formulation nor the gradient information is accessible, global optimization algorithms in Evolutionary Computation (EC) show superiority for addressing these through better exploration and exploitation tradeoff (Zhan et al., 2022). For decades, a broad family of evolutionary algorithms and swarm intelligence algorithms have been extensively studied and the corresponding application scenarios range from basic engineering problems (Slowik & Kwasnicka, 2020) to advanced scientific discovery (Chen et al., 2023; Guo et al., 2024b). Despite the good performance observed in various BBO problems, one particular technical bottleneck shared by these BBO optimizers is the generalization across different problems (Eiben & Smit, 2011). Typically, to solve a particular optimization problem, deep expertise is required to configure an existing optimizer or redesign a new one. This impedes the further spread of EC towards wider application range.

Recent research efforts in Meta-Black-Box-Optimization (MetaBBO) address the aforementioned generalization gap by introducing a bi-level learning to optimize paradigm (Ma et al., 2023), where a neural network-based control policy is maintained at the meta level and meta-trained to serve as experts for tuning the low-level BBO optimizers (as shown in the top left of Figure 1). However, achieving such generalization performance through meta-learning comes with certain challenges. On the one hand, controlling/configuring all components within the low-level optimizer requires effective policy optimization paradigm such as Reinforcement Learning (RL) (Sutton, 2018) to search for the optimal control policy in a massive joint-action space. On the other hand, to ensure the effec-

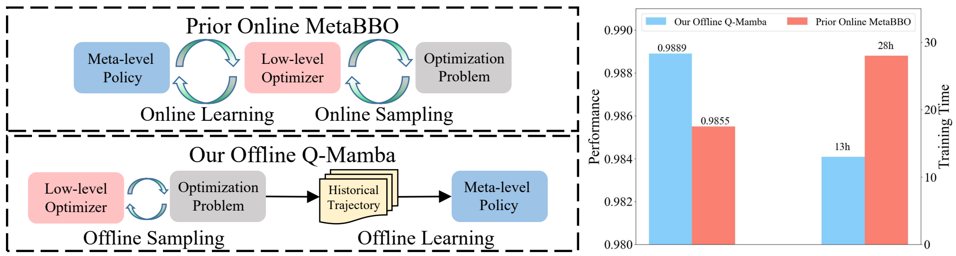


Figure 1: **Top left**: The workflow of existing MetaBBO methods, online learning fashion. **Bottom left**: The workflow of our offline Q-Mamba. **Right**: The normalized performance and training wall time comparison between our offline learning Q-Mamba and online learning MetaBBO.

tiveness of the learning, existing MetaBBO methods primarily facilitate online learning to meta-train their meta-level policies, which is inefficient particularly for the black-box optimization process which typically involves at least hundreds of optimization iterations.

Given a such dilemma in-between the effectiveness and efficiency, we in this paper propose an offline learning MetaBBO framework (as shown in the bottom left of Figure 1), termed **Q-Mamba**, to break the tie and hence ensure both the learning effectiveness and efficiency (as shown in the right of Figure 1). Concretely, to reduce the difficulty of learning optimal control policy from the entire configuration space of a black-box optimizer, we introduce a *Decomposed Q-function Representation* (DQR) which allows sequence modelling-based representation for each component's Q-function. Such decomposition has been studied in some pioneer offline RL researches (Janner et al., 2021; Chebotar et al., 2023) and demonstrated effectiveness in control problems. With the DQR, we further design a *Mamba* (Gu & Dao, 2023) *neural network-based RL agent* (Q-Learner), which treats the configuration of each component in the optimizer as a separate time step and auto-regressively predicts the corresponding decomposed Q-function by conditioning on the current optimization status and configurations of components selected before. To improve the efficiency of training the Q-Learner, we refer to offline RL (Levine et al., 2020), which learns the optimal control policy from demonstration. However, the offline RL training is vulnerable due to the endemic distributional shift issue (Wang et al., 2021) and demonstration quality issue (Ball et al., 2023). To succeed the training, we additionally integrate a *Conservative Q-Learning* (Kumar et al., 2020) regularization (CQL) into the original Bellman backup to relieve the potential distribution shift. Besides, we construct an *Exploration&Exploitation Trajectory Collection* (E&E Dataset) from a mix of randomly generated trajectories and well-performing MetaBBO trajectories. Such a combination enables better offline learning effectiveness. Accordingly, we summarize our contributions in this paper as follows:

- **Novel Framework.** Our main contribution in this paper is Q-Mamba, a novel offline RL MetaBBO framework which shows better learning effectiveness and efficiency than prior online/offline learning baselines.

- **Key Designs.** From the perspective of deep learning, we have defined the *problem formulation* (Section 4.1) as optimizing the Decomposed Q-function (DQR) for each components in a black-box optimizer to reduce the difficulty hence improve the effectiveness of learning from joint-action space, and proposed the *model* (Section 4.2) as a Mamba-based Q-Learner to enhance long-sequence modelling and learning efficiency for configuring the optimizer within the optimization process. To further stabilize the offline training, we have introduced a CQL regularization into the original *training objective* (Section 4.3) and constructed an E&E Dataset as the offline *training data* (Section 4.4), to relieve the distributional shift and reinforce the data quality during the training respectively.

- **Superior Performance.** Experimental results show that our Q-Mamba effectively achieves at least competitive optimization performance against prior online/offline learning baselines, while consuming at most half training budget of the online baselines. The learned meta-level policy can also be readily applied to enhance the performance of the low-level optimizer on unseen BBO tasks, e.g., Neuroevolution (Such et al., 2017) tasks.

2 RELATED WORKS

2.1 META-BLACK-BOX-OPTIMIZATION

Meta-Black-Box-Optimization (MetaBBO) aims to learning the optimal policy that boosts the optimization performances of the low-level optimizer over a group of optimization problems (Ma et al., 2023). Although several works facilitate supervised learning (Chen et al., 2017; Song et al., 2024; Li et al., 2024b), Neuroevolution (Lange et al., 2023b;a; Ma et al., 2024a) or even LLMs (Ma et al., 2024c; Liu et al., 2024) to meta-learn the control policy, the majority of current MetaBBO methods adopt rather reinforcement learning for the policy optimization to strike a balance between effectiveness and efficiency (Li et al., 2024a). Specifically, the dynamic algorithm configuration during the low-level optimization can be regarded as a Markov Decision Process (MDP), where the state reflects the status of the low-level optimization process, action denotes the configuration space of the low-level optimizer and a reward function is designed to provide feedback to the meta-level control policy. Existing MetaBBO methods differ with each other in the action space. In general, the configuration space of the low-level optimizer involves the operator selection and/or the hyper-parameter tuning. For the operator selection, initial works such as DE-DDQN (Sharma et al., 2019) and DE-DQN (Tan & Li, 2021) facilitate Deep Q-network (DQN) (Mnih, 2013) as the meta-level policy and dynamically suggest one of the prepared mutation operators at each optimization step for the low-level Differential Evolution (DE) (Storn & Price, 1997) optimizer. Following such paradigm, PG-DE (Zhang et al., 2024) and RL-DAS (Guo et al., 2024a) further explore the possibility of using Policy Gradient (PG) (Schulman et al., 2017) methods to train probability model for the operator selection and demonstrate PG methods are more effective than DQN methods. Besides, RLEMMO (Lian et al., 2024) and MRL-MOEA (Wang et al., 2024) extend the target optimization problem domain from single-objective optimization to multi-modal optimization and multi-objective optimization respectively. Unlike the operator selection, the action space in hyper-parameter tuning is not merely discrete since typically the hyper-parameters of an optimizer are continuous with feasible ranges. In such continuous setting, the action space is infinite and can be handled either by discretizing the continuous value range to reduce this space (Liu et al., 2019; Xu & Pi, 2020; Hong et al., 2024; Yu et al., 2024) or directly using PG methods for continuous control (Yin et al., 2021; Sun et al., 2021; Wu & Wang, 2022; Ma et al., 2024b).

While simply doing operator selection or hyper-parameter tuning for part of an optimizer has shown certain performance boost, recent MetaBBO researches such as MADAC (Xue et al., 2022) and ALDes (Zhao et al., 2024) indicate that controlling both sides gains more. However, the massive action space in such setting and the online RL process in these MetaBBO methods make it challenging to balance the training effectiveness and the efficiency. In this paper, we propose Q-Mamba as a novel MetaBBO method to control both the operator selection and hyper-parameter tuning with competitive optimization performance against previous baselines, while reducing training efficiency owing to the proposed sequential Q-function representation and offline learning strategy.

2.2 OFFLINE REINFORCEMENT LEARNING

Offline RL (Levine et al., 2020) aims at learning the optimal control policy from a pre-collected demonstration set, without the direct interaction with the environment. This is appealing for real-world complex control tasks, where on-policy data collection is extremely time-consuming (i.e., the dynamic algorithm configuration for black-box optimization discussed in this paper). A critical challenge in offline RL is the distribution shift (Fujimoto et al., 2019): learning from offline data distribution might mislead the policy optimization for out-of-distribution transitions hence degrades the overall performance. Common practices in offline RL to relieve the distribution shift include a) learning policy model (e.g., Q-value function) by sufficiently exploiting the Bellman backups of the transition data in the demonstration set and constraining the value functions for out-of-distribution ones (Haarnoja et al., 2018; Kumar et al., 2020). b) conditional imitation learning (Chen et al., 2021; Janner et al., 2021; Dai et al., 2024) which turns the MDP into sequence modelling problem and uses sequence models (e.g., recurrent neural network, Transformer or Mamba) to imitate state-action-reward sequences in the demonstration data. Although the conditional imitation learning methods have been used successfully in control domain, they do not provide any mechanism to improve the demonstrated behaviour as those policy model learning methods. A recent offline RL method, termed Q-Transformer (Chebotar et al., 2023), combines the strength of both lines of works by first decomposing the Q-value function for the entire high-dimensional action space into separate one-dimension Q-value functions, and then leveraging transformer architecture for sequential Bellman backups learning. Q-Transformer allows policy improvement during the sequence-to-sequence learning hence achieves superior performance to the prior works. Following Q-Transformer, in this paper, we propose a novel Mamba-based architecture to further enhance the long sequence processing and learning ability under MetaBBO setting.

3 Preliminaries

3.1 Decomposed Q-function Representation

Suppose we have an MDP $\{S, A = (A_1, A_2, ..., A_K), R, \mathcal{T}, \gamma\}$, where the action space is associated by a series of K action dimensions, S, $R(S,A)$, $\mathcal{T}(S'|S,A)$, γ denote the state, reward function, transition dynamic and discount factor, respectively. Value-based RL methods such as Q-learning (Watkins & Dayan, 1992) learn a Q-function $Q(s^t, a^t_{1:K})$ as the prediction of the accumulated return from the time step t by applying $a^t_{1:K}$ at s^t. The Q-function can be iteratively approximated by Bellman backup:

$$Q(a^t_{1:K}|s^t) \leftarrow R(s^t, a^t_{1:K}) + \gamma \max_{a^{t+1}_{1:K}} Q(a^{t+1}_{1:K}|s^{t+1}). \tag{1}$$

However, suppose there are at least M action bins for each of the K action dimensions, the Bellman backup above would be problematic since the associated action space contains M^K feasible actions. Such dimensional curse challenges the learning effectiveness of the value-based RL methods. Recent works such as SDQN (Metz et al., 2017) and Q-Transformer (Chebotar et al., 2023) propose decomposing the associated Q-function into a series of time-dependent Q-function representations for each action dimension to escape the curse of dimensionality. For the i-th action dimension, the decomposed Q-function is rewritten as:

$$Q(a^t_i|s^t) \leftarrow \begin{cases} \max\limits_{a^t_{i+1}} Q(a^t_{i+1}|s^t, a^t_{1:i}), & if \quad i < K \\ R(s^t, a^t_{1:K}) + \gamma \max\limits_{a^{t+1}_1} Q(a^{t+1}_1|s^{t+1}). & if \quad i = K \end{cases} \tag{2}$$

Such a decomposition allows using sequence modelling techniques to learn the optimal policy effectively, while holding the learning consistency with the Bellman backup in Eq. (1). We provide a brief proof in Appendix A.

3.2 State Space Model and Mamba

For an input sequence $x \in \mathbb{R}^{L\times D}$ with time horizon L and D-dimensional signal channels at each time step, State Space Model (SSM) (Gu et al., 2022) processes it by the following first-order differential equation, which maps the input signal $x(t) \in \mathbb{R}^D$ to the time-dependent output $y(t) \in \mathbb{R}^D$ through implicit latent state $h(t)$ as follows:

$$h(t) = \overline{A}h(t-1) + \overline{B}x(t), \quad y(t) = Ch(t). \tag{3}$$

Here, $\overline{A}$, $\overline{B}$ and C are learnable parameters, $\overline{A}$ and $\overline{B}$ are obtained by applying zero-order hold (ZOH) discretization rule. An important property of SSM is linear time invariance. That is, the dynamic parameters (e.g., $\overline{A}$, $\overline{B}$ and C) are fixed for all time steps. Such models hold limitations for sequence modelling problem where the dynamic is time-dependent. To address this bottleneck, Mamba (Gu & Dao, 2023) lets the parameters $\overline{B}$ and C be functions of the input $x(t)$. Therefore, the system now supports time-varying sequence modelling. In the rest of this paper, we use mamba_block() to denote a Mamba computation block described in Eq. (3).

4 Q-Mamba

In this section, we introduce Q-Mamba, an offline learning-based MetaBBO framework, which enables effective control policy search for black-box optimizers with massive configuration space, through efficient offline reinforcement learning. First, we describe the definition of the settings and formulation of MetaBBO tasks. Next, we elaborate how we apply Q-function decomposition and customized Q-Mamba neural network for sequence modelling of a MetaBBO task. Lastly, we derive the training objective of Q-Mamba and introduce how we collect the offline data for the training.

4.1 Problem Formulation

A MetaBBO task typically involves three key ingredients: a neural network-based meta-level policy π_θ, a black-box optimizer A and a BBO problem distribution P to be solved.

Optimizer A**.** Black-box optimizers such as Evolutionary Algorithms (EAs) have been discussed and developed over decades. Initial EAs such as Differential Evolution (DE) (Storn & Price, 1997) holds few hyper-parameters (only two, F and Cr for balancing the mutation and crossover strength). Modern variants of DE integrate various algorithmic components to enhance the optimization performance. Taking the recent winner DE optimizer in *IEEE CEC Numerical Optimization Competition* (Mohamed et al., 2021), MadDE (Biswas et al., 2021) as an example, it has more than ten hyper-parameters, which take either continuous or discrete values. Hence, the configuration space of MadDE is exponentially larger than original DE. In this paper, we use $A : \{A_1, A_2, ..., A_K\}$ to represent an optimizer with K parameters. We use additional a_i to represent the taken value of A_i.

Problem distribution P**.** By leveraging the generalization advantage of meta-learning, MetaBBO trains π_θ over a problem distribution P. A common choice of P in existing MetaBBO works is the *CoCo BBOB Testsuites* (Hansen et al., 2021), which contains 24 basic synthetic functions, each can be extended to numerous problem instances by randomly rotating and shifting the decision variables. Training on all problem instances in P is impractical. We instead sample a collection of N instances $\{f_1, f_2, ..., f_N\}$ from P as the training set. For the j-th problem f_j, we use f_j^* to represent its optimal objective value, and $f_j(x)$ as the objective value at solution point x.

For an optimizer A and a problem instance f_j, suppose we have a control policy π_θ at hand and we use A to optimize f_j for T time steps (generations). At the t-th generation, we denote the solution population as X^t. An optimization state s^t is first computed to reflect the optimization status information of the current solution population X^t and the corresponding objective values $f_j(X^t)$. Then the control policy dictates a desired configuration for A: $a^t_{1:K} = \pi_\theta(s^t)$. A optimizes X^t by $a^t_{1:K}$ and obtains an offspring population X^{t+1}. A feedback reward $R(s^t, a^t_{1:K})$ can then be computed as a measurement of the performance improvement between $f_j(X^t)$ and $f_j(X^{t+1})$. The meta-objective of MetaBBO is to search the optimal policy π_{θ^*} that maximizes the expectation of accumulated performance improvement over all problem instances in the training set:

$$\theta^* = \arg\max_\theta \frac{1}{N} \sum_{j=1}^{N} \sum_{t=1}^{T} R(s^t, a^t_{1:K} | \pi_\theta), \tag{4}$$

where such a meta-objective can be regarded as MDP. An effective policy search technique for solving MDP is RL, which is widely adopted in existing MetaBBO methods. In this paper, we focus on a particular type of RL: Q-learning, which performs prediction on the Q-function in a dynamic programming way, as described in Eq. (1).

4.2 Mamba-based Q-Learner

Existing MetaBBO works primarily struggle in learning meta-level policy with massive joint-action space, which is the configuration space $A : \{A_1, A_2, ..., A_K\}$ associated by K hyper-parameters of the low-level optimizer A. To relieve this learning difficulty, we introduce Q-function decomposition strategy as described in Section 3.1. For each hyper-parameter A_i in A, we represent its Q-function as a discretized value function $Q_i = \{Q_{i,1}, Q_{i,2}, ..., Q_{i,M}\}$, where M is a pre-defined number of action bins for all A_i in A ($M = 16$ in this paper). For any A_i which takes values from a continuous range, we uniformly discretize the value range into M bins to make universal representation across all A_i. By doing this, we turn the MDP in MetaBBO into a sequence prediction problem: we regard predicting each Q_i as a single decision step, then at time step t of the low-level optimization, the complex associated configuration $a^t_{1:K}$ of A can be sequentially decided. We further design a Mamba-based Q-Learner model to assist sequence modelling of decomposed Q-functions. The overall workflow of the Mamba-based Q-Learner is illustrated in Figure 2. We next elaborate technical elements in the figure with their design motivation.

Optimization state s^t**.** In MetaBBO, optimization state s^t profiles two types of information: the properties of the optimization problem to be solved and the low-level optimization progress. In Q-Mamba, we construct the optimization state s^t similar with latest MetaBBO methods (Ma et al.,

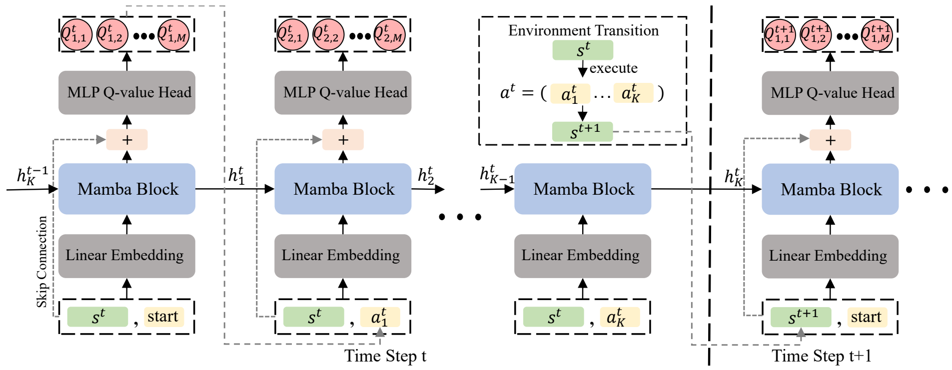


Figure 2: The workflow of the Mamba-based Q-Learner. The forward process of the neural network is similar with the Recurrent Neural Network. At each time step, the Q-function of each decomposed action dimension is output by conditioning the current state and selected action bin of the previous action dimension. The environment transition is executed once all action dimensions are output.

2024b; Chen et al., 2024; Li et al., 2024b). Concretely, at each time step t in the low-level optimization, an optimization state $s^t \in \mathbb{R}^9$ is obtained by calling a function $cal\_state()$. The first 6 dimensions are statistical features about the population distribution, objective value distribution, etc., which provide the problem property information. The last 3 dimensions are temporal features describing the low-level optimization progress. We leave the calculation detail of s^t in Appendix B.

Tokenization of action bins. We represent the $M = 16$ action bins of each hyper-parameters A_i in A by 5-bit binary coding: $00000 \sim 01111$. Besides, since we sequentially predict the Q-function for A_1 to A_K, we additionally use 11111 as a *start* token to activate the sequence prediction. We have to note that for an optimizer A, some of its discrete hyper-parameters might hold less than M action bins. For this case, we only use the first several tokens to represent the action bins in these hyper-parameters. In the rest of this paper, we use $token(a_i^t)$ to denote the binary coding of the action bin selected for A_i at time step t of the low-level optimization.

The Mamba-based Q-learner auto-regressively outputs the Q-function values Q_i^t for each A_i in A.

Linear embedding. To obtain Q_i^t, the first step is to prepare the input as the concatenation of the optimization state s^t and the previously selected action bin token $token(a_{i-1}^t)$. Then we apply a linear embedding layer on the input and obtain the embedding feature as follows:

$$\mathbb{E}_i^t = \text{Linear}([s^t, token(a_{i-1}^t)] | W_{emb}, b_{emb}), \tag{5}$$

where $W_{emb} \in \mathbb{R}^{14\times16}$ and $b_{emb} \in \mathbb{R}^{16}$ are weights and bias, respectively. For $\mathbb{E}_1^t$, *start* token is used to concat s^t, since there is no action bin before a_1^t.

Mamba block. The computation of the mamba block is described in Section 3.2. It receives the hidden state h_{i-1}^t and the embedding feature $\mathbb{E}_i^t$ and outputs the decision information $\mathbb{O}_i^t$ and hidden state h_i^t. $\mathbb{O}_i^t$ is used to parse Q-function Q_i^t and h_i^t is used for next decision step as follows:

$$\mathbb{O}_i^t, h_i^t = \text{mamba\_block}(\mathbb{E}_1^t, h_{i-1}^t | W_{mamba}), \tag{6}$$

where W_{mamba} denotes all learnable parameters in Mamba, which includes the state transition parameters A, B and C, the parameters of discretization step matrix, and time-varying mapping parameters for the state transition parameters. In this paper we use the mamba-block in Mamba repo[1], with default settings. To obtain $\mathbb{O}_1^t$, the last hidden state of time step $t-1$, h_K^{t-1} is used. The motivation of using Mamba is that: a) For a MetaBBO task, the sequence length involves thousands of decision steps since there are hundreds of optimization steps and K hyper-parameters to be decided per optimization step. We hence adopt Mamba rather than Transformer due to the the inefficiency and performance downside of Transformer for very lone sequence (Ota, 2024), which is addressed by Mamba using data-dependent embedding and hardware-aware design. b) Mamba allows selectively extracting essential information and filter out irrelevant noise according to the input sequence (Gu & Dao, 2023), which would enhance the sequence-to-sequence learning effectively.

[1]https://github.com/state-spaces/mamba

Q-value head. The Q-value head parses the decision information $\mathbb{O}_i^t$ into the decomposed Q-function Q_i^t through a linear mapping layer. Before the linear mapping, we add the input $[s^t, token(a_{i-1}^t)]$ to $\mathbb{O}_i^t$ as a skip connection as follows:

$$Q_i^t = Norm(\sigma(\mathbb{S}_i^t)), \quad \mathbb{S}_i^t = \text{Linear}(\mathbb{O}_i^t + [s^t, token(a_{i-1}^t)] | W_{head}, b_{head})). \tag{7}$$

Here, σ is Leaky ReLU activation function, $Norm$ is the min-max normalization over M bins of Q_i^t. $W_{head} \in \mathbb{R}^{16\times16}$ and $b_{head} \in \mathbb{R}^{16}$ are weights and bias. When we obtain Q_i^t, we select the action bin with the maximum value for hyper-parameter A_i: $a_i^t = \arg\max_j Q_{i,j}^t$, and use $token(a_i^t)$ for inferring the decomposed Q-function Q_{i+1}^t of next decision step. Once the action bins of all hyper-parameters $A_1 \sim A_K$ have been decided, the optimizer A optimizes the problem for one step and obtains the optimization state s^{t+1} from the updated solution population. To summarize, in Q-Mamba, the meta-level policy π_θ is the Mamba-based Q-Learner, of which the learnable parameters θ includes $\{W_{emb}, b_{emb}, W_{mamba}, W_{head}, b_{head}\}$.

4.3 Training Objective

Online learning is widely adopted in existing works, which is especially inefficient under MetaBBO setting, where the low-level optimization typically involves hundreds of optimization steps hence extremely time-consuming. In this paper we propose learning the decomposed sequential Q-function through offline RL to improve the training efficiency of MetaBBO. Concretely, we consider a trajectory $\tau = \{s^1, (a_1^1, ..., a_K^1), r^1, ..., s^T, (a_1^T, ..., a_K^T), r^T\}$, which is previously sampled by an offline policy $\hat{\pi}$. Here, a_i^t denotes the action bin selected for A_i at time step t. The training objective of Q-Mamba is a synergy of Bellman backup update (Eq. (2)) and conservative regularization as

$$J(\tau|\theta) = \sum_{t=1}^{T}\sum_{i=1}^{K}\sum_{j=1}^{M} J(Q_{i,j}^t|\theta) = \begin{cases} \frac{1}{2}(Q_{i,j}^t - \max_j Q_{i+1,j}^t)^2, & if \quad i < K, j = a_i^t \\ \frac{\beta}{2}\left[Q_{i,j}^t - (r^t + \gamma \max_j Q_{1,j}^{t+1})\right]^2, & if \quad i = K, j = a_i^t \\ \frac{\lambda}{2}(Q_{i,j}^t - 0)^2, & if \quad j \neq a_i^t \end{cases} \tag{8}$$

where $Q_{i,j}^t$ is the Q-value of the j-th bin in Q_i^t, which is output by our Mamba-based Q-Learner π_θ, with $[s^t, token(a_{i-1}^t)]$ as input. The first two branches in Eq. (8) are TD error following the Bellman backup for decomposed Q-function (as described in Eq. (2)). We additionally add a weight β (we set $\beta = 10$ in this paper) on the last action dimension to reinforce the learning on this dimension. As described in Eq. (2), the other action dimension is updated by the inverse maximization operation, so ensuring the accuracy of the Q-value in the last action dimension helps secure the accuracy of the other dimensions. The last branch in Eq. (8) is the conservative regularization introduced in representative offline RL method CQL (Kumar et al., 2020), which is used to relieve the over-estimation due to the distribution shift. Here, the Q-values of action bins which are not selected in the trajectory τ ($j \neq a_i^t$) is regularized to 0. This would accelerate the learning of the TD error. We set the weight of the conservative regularization $\lambda = 1$ in this paper.

4.4 E&E Dataset

The trajectory samples play a key role in offline RL applications (Ball et al., 2023). On the one hand, good quality data helps the training converges. On the other hand, randomly generated data help RL explores and learns more robust model. In Q-Mamba, we collect a trajectory dataset $\mathbb{C}$ of size $D = 10$K which combines the good quality data and randomly generated data. Concretely, for a low-level black-box optimizer A with K hyper-parameters and a problem distribution P, we pre-train a series of up-to-date MetaBBO methods (e.g., RLPSO (Wu & Wang, 2022), LDE (Sun et al., 2021), GLEET Ma et al. (2024b)) which control hyper-parameters of A to optimize the problems in P. Then we rollout the pre-trained MetaBBO methods on problem instances in P to collect $\mu \cdot D$ complete trajectories. We then use the random strategy to randomly control the hyper-parameters of A to optimize the problems in P and collect $(1 - \mu) \cdot D$ trajectories. By combining the exploitation experience in the trajectories of MetaBBO methods and the exploration experience in the random trajectories, our Q-Mamba learns robust and high-performance meta-level policy. In this paper, we set $\mu = 0.5$ to strike a good balance. To meta-train a Q-Mamba agent for controlling A to optimize problems in P, we use AdamW with a learning rate $5e - 3$ to minimize the expectation training objective $\mathbb{E}_{\tau \in \mathbb{C}} J(\tau|\theta)$. The training lasts for 300 epochs with a batch size of 64. After the training, the learned Q-Learner model π_θ can be directly used to control A for unseen problems. These unseen problems can be either those within the same problem distribution P, or totally out-of-distribution ones. We validate both generalization aspects of our Q-Mamba in the following experimental section.

5 Experimental Results

In the experiments, we aim to answer the following questions: a) How Q-Mamba performs compared with the other online/offline baselines? b) Can Q-Mamba be zero-shot to more challenging realistic optimization scenario? c) How important are the key designs in Q-Mamba?

5.1 Experiment Setup

Training dataset. We have prepared 10 different low-level black-box optimizer $Alg0 \sim Alg9$, which cover several types of algorithms such as DE, PSO and GA. Due to the different algorithm structure inside, the number of hyper-parameters (action dimensions) in these optimizers range from $3 \sim 16$, hence showing different difficulty-levels for MetaBBO methods. We introduce how we construct these optimizer and their algorithm structures in Appendix D.1. The problem distribution selected for the training is the ***CoCo BBOB Testsuites*** (Hansen et al., 2021), which contains 24 basic synthetic functions with diverse properties such as uni-modal, multi-modal, (non-)separable, (a)symmetrical, flattened areas, and continuity features. We denote it as P_{bbob}. We further facilitate train-test split on P_{bbob}, dividing it into 16 problem instances for the training, and 8 problem instances for the testing. These problem instances range from $5 \sim 50$-dimensional, we randomly apply shift and rotation on their solution spaces to make the optimization landscapes more challenging. Details of P_{bbob} and its train-test split is provided in Appendix D.2. By using $Alg0 \sim Alg9$ and the 16 training problem instances, we create 10 E&E Datasets by the procedure described in Section 4.4. For online baselines, we train them on each low-level optimizer to optimize the training problem instances. For offline baselines including our Q-Mamba, we train them on each E&E Dataset. We note that the total optimization steps for the low-level optimization is set as $T = 500$.

Baselines. We compare a wide range of baselines to obtain comprehensive and significant experimental observations. Concretely, we compare four **online baselines**: RLPSO (Wu & Wang, 2022) that uses simple MLP architecture for controlling low-level optimizers. LDE (Sun et al., 2021) that facilitates LSTM architecture for sequential controlling low-level optimizers using temporal optimization information. GLEET (Ma et al., 2024b) that uses Transformer architecture for mining the exploration-exploitation tradeoff during the low-level optimization. These three baselines are all trained to output associated configuration without decomposition as our Q-Mamba. We also provide an online baseline of our Q-Mamba, which learns by interacting with the environments. We also compare two **offline baselines**: Decision Transformer (Chen et al., 2021) and Q-Transformer (Chebotar et al., 2023). The former tokenizes the state, action and return-to-go signal and uses Transformer for sequence-to-sequence fitting, which is an offline RL method through conditional imitation learning. The latter applies Q-function decomposition as our Q-Mamba and facilitates offline Q-learning. However it has to split the trajectory sequence into short context windows for Transformer to process and hence is claimed relatively weak in super long sequence modelling such as the decomposed Q-value sequence in this paper. The settings of these baselines primarily follows their original papers, with a little fix up to make it compatible with the tasks in this paper. We elaborate them in Appendix D.3. To ensure the fairness of the comparison, all baselines go through the same order of training data, which is $10K$ trajectories.

Performance metric. We adopt the accumulated performance improvement $Perf(A, f|\pi_\theta)$ for measuring the optimization performance of the compared baselines and our Q-Mamba. Given a MetaBBO baseline π_θ, the corresponding low-level optimizer A and an optimization problem instance f, the accumulated performance improvement is calculated as the sum of reward feedback at each optimization step t: $Perf(A, f|\pi_\theta) = \sum_{t=1}^{T} r^t$. The reward feedback is calculated as the relative performance improvement between two consecutive optimization steps: $r^t = \frac{f^{*,t-1} - f^{*,t}}{f^{*,0} - f^*}$, where $f^{*,t}$ is the objective value of the best found solution until time step t, f^* is the optimum of f. The maximal accumulated performance improvement is 1 when the optimum of f is found.

Table 1: Performance comparison between Q-Mamba and other online/offline baselines. All baselines are tested on unseen problem instances within the training distribution P_{bbob}. We additionally present the averaged training/inferring time of all baselines in the last row.

	Online				Offline		
	RLPSO (MLP)	LDE (LSTM)	GLEET (Transformer)	Online Q-Mamba	Decision-Transformer	Q-Transformer	Q-Mamba
$Alg0$	9.855E-01 ±9.038E-03	9.563E-01 ±1.830E-02	9.616E-01 ±3.110E-03	9.873E-01 ±2.096E-01	9.325E-01 ±2.680E-02	9.646E-01 ±3.975E-02	**9.889E-01 ±7.779E-03**
$Alg1$	9.833E-01 ±6.924E-03	9.597E-01 ±1.882E-02	9.793E-01 ±6.555E-03	9.719E-01 ±2.841E-02	5.699E-01 ±1.054E-01	**9.847E-01 ±6.167E-03**	9.779E-01 ±3.602E-02
$Alg2$	9.542E-01 ±4.945E-02	**9.747E-01 ±1.748E-02**	8.913E-01 ±2.192E-02	9.347E-01 ±1.050E-01	9.297E-01 ±2.899E-02	8.290E-01 ±7.413E-02	9.325E-01 ±9.763E-02
$Alg3$	9.894E-01 ±7.337E-03	9.866E-01 ±2.054E-02	9.887E-01 ±3.853E-03	9.910E-01 ±6.400E-03	7.852E-01 ±5.396E-02	9.895E-01 ±9.949E-03	**9.915E-01 ±1.962E-02**
$Alg4$	9.953E-01 ±3.322E-03	9.877E-01 ±1.118E-02	9.938E-01 ±2.834E-03	9.951E-01 ±4.103E-03	6.764E-01 ±1.193E-01	9.951E-01 ±3.487E-03	**9.963E-01 ±7.592E-03**
$Alg5$	9.740E-01 ±2.250E-02	9.857E-01 ±8.725E-03	9.795E-01 ±1.501E-02	9.841E-01 ±9.374E-02	7.265E-01 ±1.011E-01	9.474E-01 ±2.329E-02	**9.865E-01 ±2.508E-02**
$Alg6$	9.725E-01 ±1.581E-02	9.769E-01 ±1.596E-03	9.525E-01 ±2.431E-02	9.704E-01 ±3.878E-02	9.233E-01 ±3.921E-02	8.837E-01 ±5.120E-02	**9.842E-01 ±3.285E-02**
$Alg7$	9.450E-01 ±2.050E-02	**9.735E-01 ±1.117E-02**	9.678E-01 ±1.225E-02	9.611E-01 ±2.182E-02	8.426E-01 ±4.855E-02	9.598E-01 ±3.276E-02	9.665E-01 ±6.986E-02
$Alg8$	9.924E-01 ±4.745E-03	9.867E-01 ±9.023E-03	9.898E-01 ±5.875E-03	9.9294E-01 ±1.421E-02	9.734E-01 ±1.463E-02	9.509E-01 ±1.903E-02	**9.933E-01 ±2.633E-02**
$Alg9$	9.914E-01 ±4.497E-03	9.904E-01 ±6.306E-03	9.910E-01 ±5.846E-03	9.920E-01 ±9.485E-03	8.706E-01 ±3.951E-02	9.895E-01 ±6.754E-03	**9.950E-01 ±9.981E-03**
Avg Time	28h / 11s	28h / 12s	25h / 13s	63h / 10s	13h / 10s	50h / 11s	13h / 10s

5.2 In-distribution Generalization

After the training, we compare the generalization performance of our Q-Mamba and other baselines on the 8 problem instances in P_{bbob} which are not used for the training of all baselines. Concretely, for each baseline and each low-level optimizer, we report in Table 1 the average value and error bar of the accumulated performance improvement across the 8 tested problems and 19 independent runs. We additionally present the average training time and inferring time (time consumed to complete a trajectory) for each baseline in the last row. The results in Table 1 show that: a) **Q-Mamba v.s. Online baselines.** Q-Mamba significantly outperforms the online baselines RLPSO, LDE and GLEET, which control the low-level optimizer in the massive associated configuration spaces. This evidences the effectiveness of using the decomposed Q-function representation, which could significantly reduce the configuration hence eases the learning difficulty. Meanwhile, due to the offline learning paradigm, Q-Mamba consumes only half of the training time the online baselines require. This is especially appealing for BBO scenarios where the simulation is expensive and time-consuming. b) **Q-Mamba v.s. Decision Transformer.** We observe that Decision-Transformer holds similar training efficiency with our Q-Mamba. The difference between it and Q-Mamba is that DT generally imitates the trajectory by predicting the tokens in the transitions. Results in the table show the performance of DT is quite unstable. In opposite, our Q-Mamba allows policy improvement during the sequence learning, which shows better learning convergence and effectiveness than the conditional imitation-learning based offline RL such as DT. c) **Q-Mamba v.s. Q-Transformer.** While our Q-Mamba shares the Q-function decomposition as a core design, a major novelty we introduced is the Mamba architecture and the corresponding weighted Q-function representation learning. The superior performance of Q-Mamba to the Q-Transformer possibly roots from the inability of Transformer architecture for extremely long Q-function sequence in MetaBBO setting. In Q-transformer, the entire sequence is divided into numerous context windows and learned respectively. Such forced truncation not only influences the long-term temporal dependency but also increases the training time. d) **Q-Mamba v.s. Online Q-Mamba.** We observe a performance degradation when training Q-Mamba under the online learning setting. It might reveal that the offline data provided by the other policies could enrich the experience of the meta-level policy, while online data sorely comes from the meta-level policy itself. The generalization performance is hence degraded.

5.3 Out-of-distribution Generalization

We further validate the generalization performance of Q-Mamba and other baselines on more challenging scenario, e.g., neuroevolution (Such et al., 2017) tasks. In a neuroevolution task, a black-box optimizer is used to evolve a population of neural networks according to their performance on a specific machine learning task, i.e., classification, robotic control (Galván & Mooney, 2021). Specifically, we consider four continuous control tasks in Mujoco (Todorov et al., 2012). We optimize a

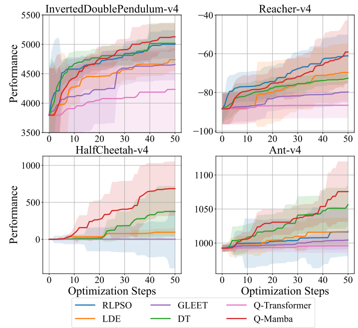


Figure 3: Zero shot performance of Q-Mamba and the other baselines on neuroevolution tasks.

Table 2: Performance analysis on the importance of loss ratio λ and β.

	$\lambda = 0$	$\lambda = 1$	$\lambda = 10$
$\beta = 1$	9.756E-01 ±1.570E-02	9.828E-01 ±1.203E-02	9.855E-01 ±1.192E-02
$\beta = 10$	9.833E-01 ±1.424E-02	**9.889E-01 ±7.780E-03**	9.857E-01 ±1.134E-02

Table 3: Performance of Q-Mamba under different proportion of good quality data.

μ	0	0.25	0.5	0.75	1
Perf.	9.832E-01 ±1.264E-02	9.874E-01 ±6.489E-03	**9.889E-01 ±7.780E-03**	9.793E-01 ±1.614E-02	9.834E-01 ±9.692E-03

2-layer MLP policy for each task by Q-Mamba and other baselines trained for controlling $Alg0$ on P_{bbob}. To align with the challenging condition in realistic BBO tasks, we only allow the low-level optimization involves a small network population (10 solutions) and $T = 50$ optimization steps. We present the average optimization curves across 10 independent runs in Figure 3. The results underscore the superior generalization performance of Q-Mamba to all other baselines: while only trained on synthetic problems with at most 50 dimensions, our Q-Mamba is capable of optimizing the MLP polices which hold thousands of parameters in these neuroevolution tasks.

5.4 Ablation Study

We perform two ablation experiments on our Q-Mamba to validate the effectiveness of the key designs. First, we demonstrate the effectiveness over the proposed training objective in Eq. (8). As shown in Table 2, when $\lambda = 0$, the training objective in Eq. (8) turns into the Bellman backup without conservative regularization. The performance degradation under this setting reveals the importance of the conservative term for relieving the distribution shift caused by offline leaning. When $\beta = 1$, the training objective would not focus on the Q-value prediction of the last action dimension, which in turn interferes the prediction of other action dimensions through the inverse maximization operation in Eq. (2). A setting with $\lambda = 1$ and $\beta = 10$ ensures the overall learning effectiveness. Next, we analyse the data mixing ratio μ in the E&E dataset (Section 4.4). When $\mu = 0$, all trajectories come from a random configuration strategy. When $\mu = 1$, all trajectories come from the well-performing MetaBBO baselines. The results in Table 3 reveal that mixing these two types of data equally ($\mu = 0.5$) might enhance Q-Mamba's learning effectiveness by leveraging the rich historical experiences from both exploration and exploitation.

6 Conclusion

In this paper, we propose Q-Mamba as a novel offline learning-based MetaBBO framework which improves both the effectiveness and the training efficiency of existing online leaning-based MetaBBO methods. To achieve this, Q-Mamba decomposes the associated Q-function for the massive configuration space into sequential Q-functions for each configuration. We further propose a Mamba-based Q-Learner for effective sequence learning tailored for such Q-function decomposition mechanism. By incorporating with a large scale offline dataset which includes both the exploration and exploitation trajectories, Q-Mamba consumes less than half training time of existing online baselines, while achieving strong control power across various black-box optimizers and diverse BBO problems. Our framework does have certain limitations. First the number of the action bins M cannot be too large under the Q-learning paradigm, this might become cumbersome if fine-grained control is required for some optimizers. Second, Q-Mamba is trained for a given optimizer and requires re-training for other optimizers. An effective optimizer feature extraction mechanism may enhance Q-Mamba's co-training on various optimizers. We mark this as an important future work.

REFERENCES

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, 2023.

Subhodip Biswas, Debanjan Saha, Shuvodeep De, Adam D Cobb, Swagatam Das, and Brian A Jalaian. Improving differential evolution through bayesian hyperparameter optimization. In *Proceedings of the IEEE Congress of Evolutionary Computation*, 2021.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, 2023.

Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. In *Advances in Neural Information Processing Systems*, 2023.

Jiacheng Chen, Zeyuan Ma, Hongshu Guo, Yining Ma, Jie Zhang, and Yue-Jiao Gong. SYMBOL: Generating flexible black-box optimizers through symbolic equation learning. In *International Conference on Learning Representations*, 2024.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 2021.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, 2017.

Yang Dai, Oubo Ma, Longfei Zhang, Xingxing Liang, Shengchao Hu, Mengzhu Wang, Shouling Ji, Jincai Huang, and Li Shen. Is mamba compatible with trajectory optimization in offline reinforcement learning? *arXiv preprint arXiv:2405.12094*, 2024.

Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

Andrej Dobnikar, Nigel C Steele, David W Pearson, Rudolf F Albrecht, Kalyanmoy Deb, and Samir Agrawal. A niched-penalty approach for constraint handling in genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Portorož, Slovenia, 1999*, pp. 235–243. Springer, 1999.

Agoston E Eiben and Selmar K Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 2011.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2019.

Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2021.

David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pp. 69–93. Elsevier, 1991.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

Hongshu Guo, Yining Ma, Zeyuan Ma, Jiacheng Chen, Xinglin Zhang, Zhiguang Cao, Jun Zhang, and Yue-Jiao Gong. Deep reinforcement learning for dynamic algorithm selection: A proof-of-principle study on differential evolution. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2024a.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024b.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 2021.

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

Jiale Hong, Bo Shen, and Anqi Pan. A reinforcement learning-based neighborhood search operator for multi-modal optimization and its applications. *Expert Systems with Applications*, 2024.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pp. 1942–1948. IEEE, 1995.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Robert Lange, Tom Schaul, Yutian Chen, Chris Lu, Tom Zahavy, Valentin Dalibard, and Sebastian Flennerhag. Discovering attention-based genetic algorithms via meta-black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023a.

Robert Tjarko Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. In *International Conference on Learning Representations*, 2023b.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Pengyi Li, Jianye Hao, Hongyao Tang, Xian Fu, Yan Zhen, and Ke Tang. Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms. *IEEE Transactions on Evolutionary Computation*, 2024a.

Xiaobin Li, Kai Wu, Yujian Betterest Li, Xiaoyu Zhang, Handing Wang, and Jing Liu. Glhf: General learned evolutionary algorithm via hyper functions. *arXiv preprint arXiv:2405.03728*, 2024b.

Hongqiao Lian, Zeyuan Ma, Hongshu Guo, Ting Huang, and Yue-Jiao Gong. Rlemmo: Evolutionary multimodal optimization assisted by deep reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024.

Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*, 2024.

Yaxian Liu, Hui Lu, Shi Cheng, and Yuhui Shi. An adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning. In *IEEE Congress on Evolutionary Computation*, 2019.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Zhenrui Li, Guojun Peng, Yue-Jiao Gong, Yining Ma, and Zhiguang Cao. Metabox: A benchmark platform for meta-black-box optimization with reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Zeyuan Ma, Jiacheng Chen, Hongshu Guo, and Yue-Jiao Gong. Neural exploratory landscape analysis. *arXiv preprint arXiv:2408.10672*, 2024a.

Zeyuan Ma, Jiacheng Chen, Hongshu Guo, Yining Ma, and Yue-Jiao Gong. Auto-configuring exploration-exploitation tradeoff in evolutionary computation via deep reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024b.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao Gong. Llamoco: Instruction tuning of large language models for optimization code generation. *arXiv preprint arXiv:2403.01131*, 2024c.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ali Wagdy Mohamed, Anas A Hadi, Ali Khater Mohamed, Prachi Agrawal, Abhishek Kumar, and P. N. Suganthan. Problem definitions and evaluation criteria for the cec 2021 on single objective bound constrained numerical optimization. In *Proceedings of the IEEE Congress of Evolutionary Computation*, 2021.

Toshihiro Ota. Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. *arXiv preprint arXiv:2403.19925*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Mudita Sharma, Alexandros Komninos, Manuel López-Ibáñez, and Dimitar Kazakov. Deep reinforcement learning based parameter control in differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 2020.

Lei Song, Chenxiao Gao, Ke Xue, Chenyang Wu, Dong Li, Jianye Hao, Zongzhang Zhang, and Chao Qian. Reinforced in-context black-box optimization. *arXiv preprint arXiv:2402.17423*, 2024.

Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341, 1997.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Jianyong Sun, Xin Liu, Thomas Bäck, and Zongben Xu. Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. *IEEE Transactions on Evolutionary Computation*, 2021.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Zhiping Tan and Kangshun Li. Differential evolution with mixed mutation strategy based on deep reinforcement learning. *Applied Soft Computing*, 2021.

Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of shade using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1658–1665. IEEE, 2014.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 2012.

Jing Wang, Yuxin Zheng, Ziyun Zhang, Hu Peng, and Hui Wang. A novel multi-state reinforcement learning-based multi-objective evolutionary algorithm. *Information Sciences*, 2024.

Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham Kakade. Instabilities of offline rl with pre-trained neural representation. In *International Conference on Machine Learning*, 2021.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

Di Wu and G Gary Wang. Employing reinforcement learning to enhance particle swarm optimization methods. *Engineering Optimization*, 2022.

Yue Xu and Dechang Pi. A reinforcement learning-based communication topology in particle swarm optimization. *Neural Computing and Applications*, 2020.

Ke Xue, Jiacheng Xu, Lei Yuan, Miqing Li, Chao Qian, Zongzhang Zhang, and Yang Yu. Multi-agent dynamic algorithm configuration. In *Advances in Neural Information Processing Systems*, 2022.

Shiyuan Yin, Yi Liu, GuoLiang Gong, Huaxiang Lu, and Wenchang Li. Rlepso: Reinforcement learning based ensemble particle swarm optimizer. In *International Conference on Algorithms, Computing and Artificial Intelligence*, 2021.

Xiaobing Yu, Pingping Xu, Feng Wang, and Xuming Wang. Reinforcement learning-based differential evolution algorithm for constrained multi-objective optimization problems. *Engineering Applications of Artificial Intelligence*, 2024.

Zhi-Hui Zhan, Lin Shi, Kay Chen Tan, and Jun Zhang. A survey on evolutionary computation for complex continuous optimization. *Artificial Intelligence Review*, 2022.

Haotian Zhang, Jianyong Sun, Thomas Bäck, and Zongben Xu. Learning to select the recombination operator for derivative-free optimization. *Science China Mathematics*, 2024.

Qi Zhao, Tengfei Liu, Bai Yan, Qiqi Duan, Jian Yang, and Yuhui Shi. Automated metaheuristic algorithm design with autoregressive learning. *arXiv preprint arXiv:2405.03419*, 2024.

A Proof of Q-function Decomposition

To show that transforming MDP into a per-action-dimension form still ensures optimization of the original MDP, we show that optimizing the Q-function for each action dimension is equivalent to optimizing the Q-function for the full action. We omit the time step superscript t for the ease of presentation.

If we consider apply full action $a_{1:K}$ at the current state s to transit to the next step state s'. The Bellman update of the optimal Q-function could be written as:

$$\begin{aligned}\max_{a_{1:K}} Q(a_{1:k}|s) &= \max_{a_{1:K}} \left[R(s, a_{1:K}) + \gamma \max_{a_{1:K}} Q(a_{1:K}|s') \right] \\ &= R(s, a^*_{1:K}) + \gamma \max_{a_{1:K}} Q(a_{1:K}|s') \end{aligned} \tag{9}$$

where $R(\cdot)$ is the reward we get after executing the full action $a_{1:K}$. Under the Q-function decompostion, the Bellman update of the optimal Q-function for each action dimension a_i is:

$$\begin{aligned}\max_{a_i} Q(a_i|s, a^*_{1:i-1}) &= \max_{a_i} \left[\max_{a_{i+1}} Q(a_{i+1}|s, a^*_{1:i}) \right] \\ &= \max_{a_i} \left[\max_{a_{i+1}} \left(\max_{a_{i+2}} Q(a_{i+2}|s, a^*_{1:i+1}) \right) \right] \\ &= \cdots \\ &= R(s, a^*_{1:K}) + \gamma \max_{a_1} Q(a_1|s') \\ &= R(s, a^*_{1:K}) + \gamma \max_{a_1} \left[\max_{a_2} Q(a_2|s', a_1) \right] \\ &= \cdots \\ &= R(s, a^*_{1:K}) + \gamma \max_{a_{1:K}} Q(a_{1:K}|s') \end{aligned} \tag{10}$$

Here the first two lines are the inverse maximization operation as described in Section 3.1, the fourth line is the Bellman update for the last action dimension. The last three lines also follow the inverse maximization operation. By comparing Eq. (9) and Eq. (10) we prove that optimizing the decomposed Q-function consistently optimizes the original full MDP.

B Optimization State Design

The formulation of the optimization state features is described in Table 4. States $s_{\{1\sim 6\}}$ are optimization problem property features which collectively represent the distributional features and the statistics of the objective values of the current candidate population. Specifically, state s_1 represents the average distance between each pair of candidate solutions, indicating the overall dispersion level. State s_2 represents the average distance between the best candidate solution in the current population and the remaining solutions, providing insights into the convergence situation. State s_3 represents the average distance between the best solution found so far and the remaining solutions, indicating the exploration-exploitation stage. State s_4 represents the average difference between the best objective value found in the current population and the remaining solutions, and s_5 represents the average difference when compared with the best objective value found so far. State s_6 represents the standard deviation of the objective values of the current candidates. Then, states $s_{\{7,8,9\}}$ collectively represent the time-stamp features of the current optimization progress. Among them, state s_7 denotes the current process, which can inform the framework about when to adopt appropriate strategies. States s_8 and s_9 are measures for the stagnation situation.

C Action Discretization and Reconstruction

Given the $M = 16$ bins of Q values Q_i^t for the i-th action, if the i-th hyper-parameter A_i of the low-level optimizer is in continuous space, we first uniformly discretize the space into M bins: $\hat{A}_i = \{A_{i,1}, A_{i,2}, \cdots, A_{i,M}\}$ where $A_{i,1}$ and $A_{i,M}$ are the lower and upper bounds of the space.

Table 4: Formulations of state features.

		States	Notes
Problem Property	s_1^t	$\underset{x_i,x_j \in X^t}{mean} \|\|x_i - x_j\|\|_2$	Average distance between any pair of individuals in current population.
	s_2^t	$\underset{x_i \in X^t}{mean} \|\|x_i - x^{*,t}\|\|_2$	Average distance between each individual and the best individual in t-th generation.
	s_3^t	$\underset{x_i \in X^t}{mean} \|\|x_i - x^*\|\|_2$	Average distance between each individual and the best-so-far solution.
	s_4^t	$\underset{x_i \in X^t}{mean}(f(x_i) - f(x^*))$	Average objective value gap between each individual and the best-so-far solution.
	s_5^t	$\underset{x_i \in X^t}{mean}(f(x_i) - f(x^{*,t}))$	Average objective value gap between each individual and the best individual in t-th generation.
	s_6^t	$\underset{x_i \in X^t}{std}(f(x_i))$	Standard deviation of the objective values of population in t-th generation, a value equals 0 denotes converged.
Optimization Progress	s_7^t	$(T - t)/T$	The potion of remaining generations, T denotes maximum generations for one run.
	s_8^t	st/T	st denotes how many generations the optimizer stagnates improving.
	s_9^t	$\begin{cases} 1 & \text{if } f(x^{*,t}) < f(x^*) \\ 0 & \text{otherwise} \end{cases}$	Whether the optimizer finds better individual than the best-so-far solution.

Then we use the action a_i^t obtained by $a_i^t = \arg\max_j Q_{i,j}^t$ as an index and assign the value of the i-th hyper-parameter A_i with $A_i = \hat{A}_i[a_i^t]$. If the hyper-parameter is in discrete space $\hat{A}$ with $m_i \leq M$ candidate choices, the action a_i^t is obtained by $a_i^t = \arg\max_{j \in [1, m_i]} Q_{i,j}^t$ and the value of the i-th hyper-parameter is $\hat{A}[a_i^t]$. After the value of all hyper-parameters are decided, the optimizer A takes a step of optimization with the hyper-parameters and return the next state from the updated population.

D Experiment Setup

D.1 Backend Algorithm Generalization

In this paper, we randomly construct 10 optimizers with action space dimensions $\{3, 5, 7, 8, 10, 12, 13, 14, 15, 16\}$. To do so, we first collect a optimization operator space containing operators with controllable parameters such as the mutation and crossover operators from DE (Storn & Price, 1997), PSO update rules (Kennedy & Eberhart, 1995), crossover and mutation operators from GA (Holland, 1992). Operators without controllable parameters such as selection and population reduction operators are also included. Then, to get an optimizer with n hyper-parameters, we randomly sample a batch of operators to construct an optimizer, if the total number of controllable parameters in all operators of the optimizer is not match n, we eliminate it and resample until the wanted optimizer is constructed. The hyper-parameters of the optimizer such as the initial population sizes are randomly determined. Below we present the structure of $Alg0$ (3 actions) and $Alg9$ (16 actions) as examples.

$Alg0$ (as shown in Algorithm 1) is DE/current-to-rand/1/exponential (Storn & Price, 1997) with Linear Population Size Reduction (LPSR) (Tanabe & Fukunaga, 2014). The mutation operator DE/current-to-rand/1 is formulated as:

$$x_i' = x_i + F1(x_{r1} - x_i) + F2(x_{r2} - x_{r3}) \tag{11}$$

Algorithm 1 Pseudo code of $Alg0$

1: **Input**: Optimization problem f, optimization horizon T, Meta-level agent π.
2: **Output**: Optimal solution $x^* = \arg\min_{x \in X} f(x)$.
3: Uniformly initialize a population X_1 with shape $NP_1 = 100$ and evaluate it with problem f;
4: **for** $t = 1$ **to** T **do**
5: Receive the 3 action values $a_t = \{F1, F2, Cr\}$ from the agent π;
6: Generate X'_t by using DE/current-to-rand/1 (Eq. (11)) on X_t;
7: Apply Exponential crossover (Eq. (12)) on X_t and X'_t to get X''_t;
8: Clip the values beyond the search range in X''_t;
9: Calculate $f(X''_t)$;
10: Compare $f(X_t)$ and $f(X''_t)$, select the better solutions to generate X_{t+1};
11: **end for**

where $x_{r\cdot}$ are randomly chosen solutions and $F1, F2 \in [0, 1]$ are two controllable parameters. The Exponential crossover operator is formulated as:

$$x''_i = \begin{cases} x'_{i,j}, \text{ if } rand_{k:j} < Cr \text{ and } k \leq j \leq L + k \\ x_{i,j}, \text{ otherwise} \end{cases}, j = 1, \cdots, Dim \tag{12}$$

where Dim is the solution dimension, $L \in \{1, \cdots, Dim\}$ is a random length, $rand \in [0, 1]^{Dim}$ is a random vector, x'_i is the trail solution generated by mutation operator and $Cr \in [0, 1]$ is a controllable parameter. At the beginning, a population X with size 100 is uniformly sampled and evaluated. In each optimization generation, given the parameters $F1, F2, Cr$ from the meta-level agent, algorithm applies DE/current-to-rand/1 mutation and Exponential crossover operator on the population to generate the trail solution population X''_t. An comparison is conducted between population X_t and X''_t where the better solutions are selected for the next generation population X_{t+1}. Finally the worst solutions are removed from X_{t+1} in the LPSR process.

For $Alg9$ (as shown in Algorithm 2), the population sampled in Halton sampling (Halton, 1960) is divided into four sub-populations. The first sub-population uses GA operators MPX (Holland, 1992) crossover and Polynomial mutation (Dobnikar et al., 1999) accompanying with the Roulette selection (Holland, 1992). MPX crossover is formulated as:

$$x'_i = \begin{cases} x'_{r1,j}, \text{ if } rand_j < Cr_1 \\ x'_{i,j}, \text{ otherwise} \end{cases}, j = 1, \cdots, Dim \tag{13}$$

where $rand_j \in [0, 1]$ are random numbers, Cr_1 is a controllable parameter and x_{r1} is a random solution. The sample method of x_{r1} is also a controllable action Xr_{mpx} which can be uniform sampling or sampling with fitness based ranking. The Polynomial mutation is as follow:

$$x''_i = \begin{cases} x'_i + ((2u)^{\frac{1}{1+\eta_m}} - 1)(x'_i - lb), \text{if } u \leq 0.5; \\ x'_i + (1 - (2 - 2u)^{\frac{1}{1+\eta_m}})(ub - x'_i), \text{if } u > 0.5. \end{cases} \tag{14}$$

where $\eta_m \in \{1, 2, 3\}$ is a controllable parameter, $u \in [0, 1]$ is a random number, ub and lb are the upper and lower bound of the search range.

The second sub-population uses SBX crossover (Deb et al., 1995), Gaussian mutation (Holland, 1992) and Tournament selection Goldberg & Deb (1991):

$$x'_i = 0.5 \cdot [(1 \mp \beta)x_i + (1 \pm \beta)x_{r1}], \text{ where } \beta = \begin{cases} (2u)^{\frac{1}{1+\eta_c}} - 1, \text{if } u \leq 0.5; \\ (\frac{1}{2-2u})^{\frac{1}{1+\eta_c}}, \text{if } u > 0.5. \end{cases} \tag{15}$$

where $\eta_c \in \{1, 2, 3\}$ is controllable parameter and $u \in [0, 1]$ is random number. Similar to MPX, SBX also uses an action Xr_{sbx} to select parent solutions x_{r1}. The Gaussian mutation operator applies Gaussian noise with controllable parameter $\sigma \in [0, 1]$ on the solution:

$$x''_i = \mathcal{N}(x'_i, \sigma \cdot (ub - lb)) \tag{16}$$

The third sub-population is DE/rand/2/exponential (Storn & Price, 1997) where the DE/rand/2 mutation operator is:

$$x'_i = x_{r1} + F1_3(x_{r2} - x_{r3}) + F2_3(x_{r4} - x_{r5}) \tag{17}$$

Algorithm 2 Pseudo code of $Alg9$

```
Input: Optimization problem f, optimization horizon T, Meta-level agent π.
Output: Optimal solution x* = arg min_{x∈X} f(x).
Initialize 4 sub-populations {X_{i,1}}_{i=1,2,3,4} using Halton sampling with sizes {200, 100, 100, 100}.
Evaluate the sub-populations with problem f;
for t = 1 to T do
    Receive the 16 action values a_t from the agent π;
    Generate X_{1,t+1} using MPX (Eq. (13)), Polynomial mutation (Eq. (14)) and Roulette selection on X_{1,t};
    Generate X_{2,t+1} using SBX (Eq. (15)), Gaussian mutation (Eq. (16)) and Tournament selection on X_{2,t};
    Generate X_{3,t+1} using DE/rand/2 mutation (Eq. (17)), Exponential crossover (Eq. (12)) on X_{3,t};
    Generate X_{4,t+1} using DE/current-to-best/1 mutation (Eq. (18)), Binomial crossover (Eq. (19)) on X_{4,t};
    for i = 1 to 4 do
        Replace the worst solution in X_{i,t+1} by the best solution in X_{cm_i,t+1}
    end for
end for
```

where $x_{r\cdot}$ are randomly selected solutions and $F1_3, F2_3 \in [0, 1]$ are controllable parameters for the third sub-population. The Exponential crossover formulated as Eq. (12) is used in this sub-population with parameter $Cr_3 \in [0, 1]$.

The last sub-population is DE/current-to-best/1/binomial (Storn & Price, 1997). The mutation operator with parameter $F1_4, F2_4 \in [0, 1]$ is formulated as:

$$x_i' = x_i + F1_4(x^* - x_i) + F2_4(x_{r1} - x_{r2}) \tag{18}$$

where x^* is the best performing solution in the sub-population. The Binomial crossover uses a similar process as MPX but introduces a randomly selected index $jrand \in \{1, \cdots, Dim\}$ to ensure the difference between the generated solution and the parent solution:

$$x_i'' = \begin{cases} x_{i,j}', & \text{if } rand_j < Cr_4 \text{ or } j = jrand \\ x_{i,j}', & \text{otherwise} \end{cases} \quad , j = 1, \cdots, Dim \tag{19}$$

where $rand_j$ are random numbers and $Cr_4 \in [0, 1]$ is the controllable parameter.

Besides, $Alg9$ conducts the controllable information sharing among the sub-populations where the worst solution in current sub-population $X_{i,g}$ is replaced by the best solution from the target sub-population $X_{cm_i,g}$, $cm_{\{1,2,3,4\}} \in \{1, 2, 3, 4\}$ are four actions indicating the target sub-population.

Given the 16 actions $\{Cr_1, Xr_{mpx}, \eta_m, \eta_c, Xr_{sbx}, \sigma, F1_3, F2_3, Cr_3, F1_4, F2_4, Cr_4, cm_1, cm_2, cm_3, cm_4\}$, $Alg9$ uses these parameters to configure the mutation and crossover operators and applies them on the 4 sub-populations. Then the information sharing is activated for better exploration. Finally, the next generation population is obtained through the population reduction processes.

D.2 Train-test split of BBOB Problems

As shown in Table 5, the BBOB testsuite (Hansen et al., 2021) contains 24 different optimization problems with diverse characteristics such as unimodal or multi-modal, separable or non-separable, high conditioning or low conditioning. To maximize the problem diversity of the training problem set and hence empower the agent better generalization ability, we choose the most diverse 16 problem instance for training, whose fitness landscapes in 2D scenario are shown in Figure 4. The rest 8 instances are used as testing set whose 2D landscapes are shown in Figure 5. The dimensions of each problem instances in both training and testing set are randomly chosen from $\{5, 10, 20, 50\}$.

D.3 Baseline Implementation

RLPSO (Wu & Wang, 2022) uses two MLP policy networks to configure the algorithm parameters. For each solution in each optimization generation, given the solution and the best so far solution, RLPSO generates the a pair of μ and σ of the target parameter using the two networks respectively.

Table 5: Overview of the BBOB testsuites.

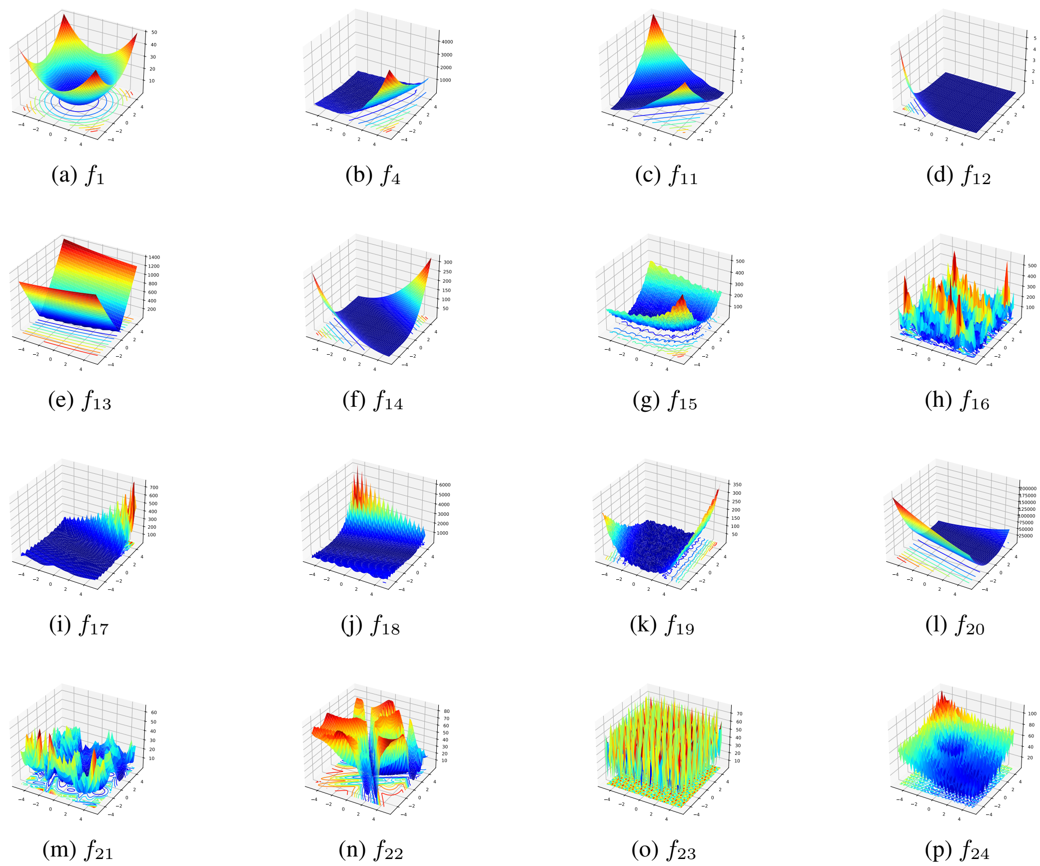
	Problem	Functions	Dimensions
Separable functions	f_1	Sphere Function	50
	f_2	Ellipsoidal Function	5
	f_3	Rastrigin Function	5
	f_4	Buche-Rastrigin Function	10
	f_5	Linear Slope	50
Functions with low or moderate conditioning	f_6	Attractive Sector Function	5
	f_7	Step Ellipsoidal Function	20
	f_8	Rosenbrock Function, original	10
	f_9	Rosenbrock Function, rotated	10
Functions with high conditioning and unimodal	f_{10}	Ellipsoidal Function	10
	f_{11}	Discus Function	5
	f_{12}	Bent Cigar Function	50
	f_{13}	Sharp Ridge Function	10
	f_{14}	Different Powers Function	20
Multi-modal functions with adequate global structure	f_{15}	Rastrigin Function (non-separable counterpart of F3)	5
	f_{16}	Weierstrass Function	20
	f_{17}	Schaffers F7 Function	50
	f_{18}	Schaffers F7 Function, moderately ill-conditioned	50
	f_{19}	Composite Griewank-Rosenbrock Function F8F2	10
Multi-modal functions with weak global structure	f_{20}	Schwefel Function	20
	f_{21}	Gallagher's Gaussian 101-me Peaks Function	20
	f_{22}	Gallagher's Gaussian 21-hi Peaks Function	10
	f_{23}	Katsuura Function	20
	f_{24}	Lunacek bi-Rastrigin Function	20
Default search range: [-5, 5]^{Dim}			

Then the parameter value is sampled from $\mathcal{N}(\mu, \sigma)$ and the two policy networks are updated by policy gradient. The original design of using the solution and best solution as network input hinders the generalization ability of the RLPSO policy across problems with different dimensions, therefore in the experiment we replace the network input by the same 9-dimensional state representation as Q-Mamba. To control the algorithms with up to 16 actions in our experiment, we set the output dimension of the two networks to 16 and use the first few values if the number of actions of the algorithm is lower than 16. In summary, for RLPSO baseline we use the MLP with structure ($9 \times 64 \times 32 \times 16$) for both networks and retain their original Policy Gradient training process.

LDE (Sun et al., 2021) adopts a Long Short-Term Memory (LSTM) network to integrate the optimization information from previous optimization generations and the fitness of solutions in current population. Then two MLP networks predict the μ and σ for the target parameters of each solution based on the integrated optimization status. REINFORCE is used to update the policy at the end of an optimization trajectory. LDE configure the individual-level parameters for each solution therefore its state representation and action design are related to the population size. To adapt the network to our generated algorithms where population sizes may reduce, we conduct the modification similar to that on RLPSO: we use our 9-dimensional state instead of its original population size related state. The output dimensions of the networks are also set to 16 to perform the population-level parameter configuration. In this paper, we use an one-layer LSTM with input dimension 9 and hidden dimension 32. The MLP network for μ and σ are both (32×16).

GLEET (Ma et al., 2024b) designs a feature embedding module for feature extraction, a Transformer-based fully informed encoder for information processing amongst individuals and an exploration-exploitation decoder for individual-level parameter configuration in which the encoded individual features are decoded in a Transformer block and generate the individual-wise μs and σs for action sampling. The problem dimension-free state representation and Transformer-based network structure make GLEET compatible to our generated algorithms and problems. Therefore we retain its network designs except the output dimension: a meanpooling is conducted on the decoded features in the exploration-exploitation decoder to transform the individual-level features into a population feature, then 16-dimensional μ and σ are predicted by two MLPs.

The Decision Transformer adopts a trajectory-based learning approach, utilizing a Transformer architecture to model the decision-making process from sequential data. It consists primarily of three components: a trajectory embedding module for embedding state-action-return sequences, a Transformer-based decision module for processing sequential information, and a policy decoder for



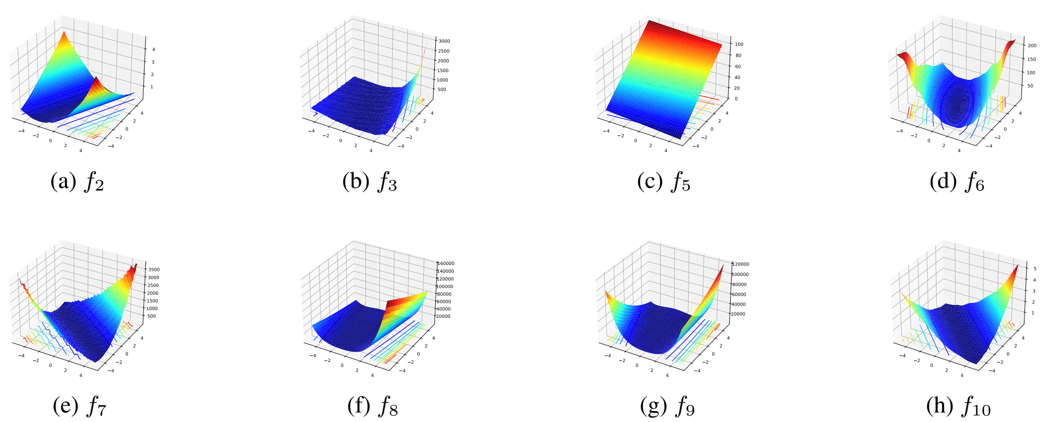
(a) f_1 (b) f_4 (c) f_{11} (d) f_{12}

(e) f_{13} (f) f_{14} (g) f_{15} (h) f_{16}

(i) f_{17} (j) f_{18} (k) f_{19} (l) f_{20}

(m) f_{21} (n) f_{22} (o) f_{23} (p) f_{24}

Figure 4: Fitness landscapes of functions in BBOB **train** set when dimension is set to 2.



(a) f_2 (b) f_3 (c) f_5 (d) f_6

(e) f_7 (f) f_8 (g) f_9 (h) f_{10}

Figure 5: Fitness landscapes of functions in BBOB **test** set when dimension is set to 2.

action generation. The trajectory embedding module encodes states, actions, and returns into token sequences. These tokens are processed through standard Transformer encoder blocks, leveraging self-attention mechanisms to capture long-range dependencies within the trajectory. The encoded sequences are then passed to the policy decoder, which generates predictions for the next action based on the observed past states, actions, and expected returns. The Transformer-based structure enables the Decision Transformer to handle sequences of varying lengths and complex state-action dynamics. In our task, each action space dimension of the original Decision Transformer is treated as an independent token, changing the input sequence format from the original DT's $(R_1, s_1, a_1, R_2, s_2, \dots)$ to $(R_1, s_1, a_{1_1}, a_{1_2}, \dots, a_{1_n}, R_2, s_2, \dots)$ to avoid the exponential growth of the action space.

Q-Transformer is a scalable offline reinforcement learning approach that employs a Transformer-based architecture to model Q-functions for multi-task policies. This method discretizes each dimension of the action space, treating each as a separate token, which facilitates auto-regressive Q-learning through effective sequence modelling techniques. By adopting this strategy, Q-Transformer effectively mitigates the exponential growth of the action space, making it well-suited for large-scale offline reinforcement learning tasks. A notable feature of Q-Transformer is its implementation of conservative Q-function regularization, which addresses distributional shifts in conjunction with n-step returns to improve learning efficiency. In our implementation, we utilize a linear action encoder, a single-layer Transformer encoder combined with an MLP as a Q-value head to maintain a compact model size with Q-Mamba.