Toward Automated Algorithm Design: A Survey and Practical Guide to Meta-Black-Box-Optimization

Zeyuan Ma, Hongshu Guo, Yue-Jiao Gong, *Senior Member, IEEE*
Jun Zhang, *Fellow, IEEE* and Kay Chen TAN, *Fellow, IEEE*

***Abstract*—In this survey, we introduce Meta-Black-Box-Optimization (MetaBBO) as an emerging avenue within the Evolutionary Computation (EC) community, which incorporates Meta-learning approaches to assist automated algorithm design. Despite the success of MetaBBO, the current literature provides insufficient summaries of its key aspects and lacks practical guidance for implementation. To bridge this gap, we offer a comprehensive review of recent advances in MetaBBO, providing an in-depth examination of its key developments. We begin with a unified definition of the MetaBBO paradigm, followed by a systematic taxonomy of various algorithm design tasks, including algorithm selection, algorithm configuration, solution manipulation, and algorithm generation. Further, we conceptually summarize different learning methodologies behind current MetaBBO works, including reinforcement learning, supervised learning, neuroevolution, and in-context learning with Large Language Models. A comprehensive evaluation of the latest representative MetaBBO methods is then carried out, alongside an experimental analysis of their optimization performance, computational efficiency, and generalization ability. Based on the evaluation results, we meticulously identify a set of core designs that enhance the generalization and learning effectiveness of MetaBBO. Finally, we outline the vision for the field by providing insight into the latest trends and potential future directions. Relevant literature will be continuously collected and updated at https://github.com/GMC-DRL/Awesome-MetaBBO.**

I. Introduction

Optimization techniques have been central to research for decades [1], [2], with methods applied across engineering [3], economics [4], and science [5]. The optimization problems can be classified into White-Box [6] and Black-Box [7] types. White-Box problems, with transparent structures, allow efficient optimization using gradient-based algorithms like SGD [8], Adam [9], and BFGS [10]. In contrast, Black-Box Optimization (BBO) only provides objective values for solutions, making the analysis and search of the problem space even more challenging.

Evolutionary Computation (EC), including Evolutionary Algorithms (EAs) and Swarm Intelligence (SI), is widely recognized as an effective gradient-free approach for solving BBO problems [11]. Over the past decades, EC methods have been extensively applied to various optimization challenges [12]–[16], due to their simplicity and versatility. Though effective for solving BBO problems, traditional EC is constrained by the no-free-lunch theorem [17], which asserts that no optimization algorithm can universally outperform others across all problem types, leading to performance trade-offs depending on the problem's characteristics. In response, various adaptive and self-adaptive EC variants [18]–[25] have been developed. These variants leverage historical optimization data for hyper-parameter control or operator/algorithm selection during optimization, improving general performance. However, they face several limitations. 1) Limited generalization: these methods often focus on a specific set of problems, limiting their generalization due to customized designs. 2) Labor-intensive: designing adaptive mechanisms requires both deep knowledge of EC domain and the target optimization problem, making it a complex task. 3) Additional parameters: many adaptive mechanisms introduce extra hyper-parameters, which can significantly impact performance. 4) Sub-optimal performance: despite increased efforts, design biases and delays in reactive adjustments often lead to sub-optimal outcomes.

Given this, a natural question arises: can we automatically design effective BBO algorithms while minimizing the dependence on expert input? A recently emerging research topic, known as Meta-Black-Box-Optimization (MetaBBO) [26], has shown possibility of leveraging the generalization strength of Meta-learning [27] to enhance the optimization performance of BBO algorithms in the minimal expertise cost. MetaBBO follows a bi-level paradigm: the meta level typically maintains a policy that takes the low-level optimization information as input and then automatically dictates desired algorithm design for the low-level BBO optimizer. The low-level BBO process evaluates the suggested algorithm design and returns a feedback signal to the meta-level policy regarding the performance gain. The meta-objective of MetaBBO is to meta-learn a policy that maximizes the performance of the low-level BBO process, over a problem distribution. Once the training completes, the learned meta-level policy can be directly applied to address unseen optimization problems, hence reducing the need for expert knowledge to adapt BBO algorithms.

Numerous valuable ideas have been proposed and discussed in existing MetaBBO research. From the perspective of algorithm design tasks (meta tasks) that the meta-level policy can address, those MetaBBO works can be cate-

Zeyuan Ma, Hongshu Guo, and Yue-Jiao Gong are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (E-mail: gongyuejiao@gmail.com)

Jun Zhang is with Nankai University, Tianjin, China and Hanyang University, Seoul, South Korea. (E-mail: junzhang@ieee.org).

Kay Chen TAN is with Hong Kong Polytechnic University, Hong Kong, China. (E-mail: kaychen.tan@polyu.edu.hk).

Corresponding author: Yue-Jiao Gong

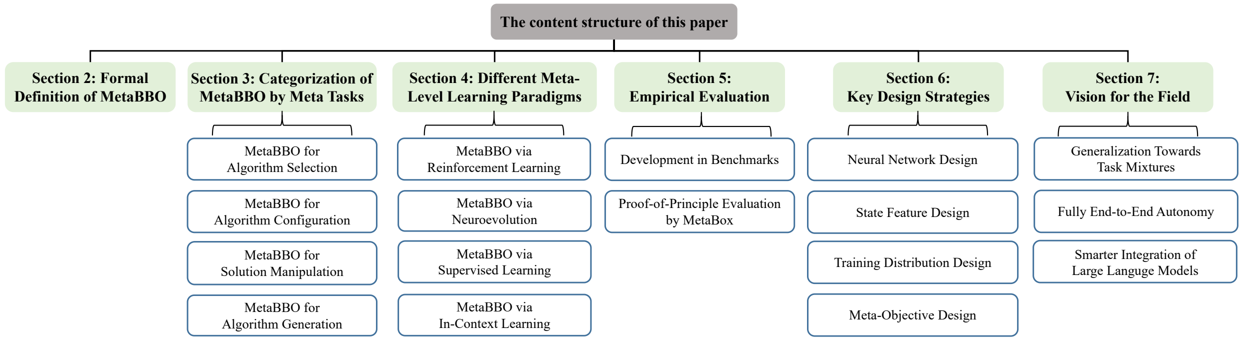

Fig. 1. Roadmap of the content structure, beginning with a concept introduction, followed by a review of existing methods across different taxonomies, a evaluation of selected methods, and a summary of key design strategies and future vision.

gorized into four branches: 1) Algorithm Selection, where for solving the given problem, a proper BBO algorithm is selected by the meta-level policy from a pre-collected optimizer/operator pool. 2) Algorithm Configuration, where the hyper-parameters and/or operators of a BBO algorithm are adjusted by the meta-level policy to adapt for the given problem. 3) Solution Manipulation, where the meta-level policy is trained to act as a BBO algorithm to manipulate and evolve solutions. 4) Algorithm Generation, where each algorithmic component and the overall workflow are generated by the meta-level policy as a novel BBO algorithm. From the perspective of learning paradigms adopted for training the meta-level policy, different learning methods such as reinforcement learning (MetaBBO-RL) [28]–[34], auto-regressive supervised learning (MetaBBO-SL) [35]–[40], neuroevolution (MetaBBO-NE) [41]–[43], and Large Language Models (LLMs)-based in-context learning (MetaBBO-ICL) [37], [44]–[47] have been investigated in existing works. From the perspective of low-level BBO process, MetaBBO has been instantiated to various optimization scenarios such as single-objective optimization [31], [32], [34], multi-objective optimization, multi-modal optimization [48], large scale global optimization [40], [42], [43], and multi-task optimization [49], [50]. Such an intricate combination of algorithm design tasks, learning paradigms, and low-level BBO scenarios makes it challenging for new practitioners to systematically learn, use, and develop MetaBBO methods. Unfortunately, there is still a lack of a comprehensive survey and practical guide to the advancements in MetaBBO.

While some related surveys discussed the integration of learning systems into EC algorithm designs, they have several limitations: 1) Previous surveys [51]–[53] focus on one or two algorithm design tasks, such as algorithm configuration [53] and algorithm generation [51], [52]. These surveys therefore show short in providing comprehensive review and comparison analysis on all four design tasks. 2) Some surveys [54]–[57] focus on a particular learning paradigm - RL [58]. However, in MetaBBO, various learning paradigms can be adopted, each with distinct characteristics. 3) In addition to reviewing relevant papers, existing surveys lack a practical guide that provides a comprehensive experimental evaluation of MetaBBO methods and a summary of key design strategies, falling short in offering in-depth evaluations or actionable insights for implementing MetaBBO methods.

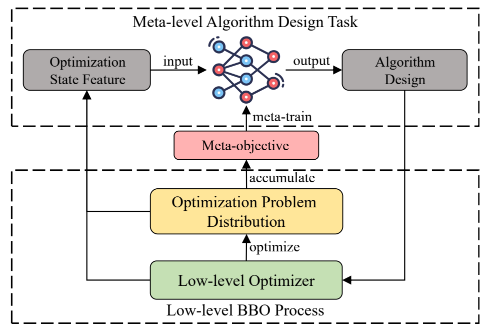

Fig. 2. A conceptual overview of the bi-level learning framework of MetaBBO, illustrating the interactions between its core components to clarify the overall workflow.

To address the gaps in previous surveys, this paper provides a more comprehensive coverage of the MetaBBO field. Fig. 1 offers a roadmap to help readers quickly navigate the overall content structure. We first provide a formal definition of MetaBBO in Section II. Subsequently, we identify four main algorithm design tasks in existing MetaBBO works and their working scenarios in Section III. In Section IV, we further elaborate four learning paradigms, with easy-to-follow technical details. Section V provides a proof-of-principle performance evaluation on nine representative MetaBBO methods. According to the evaluation results, Section VI provides in-depth discussion about the key design strategies in MetaBBO. Finally, we outline the vision for the MetaBBO field in Section VII. The contributions of this survey are generally summarized as follows:

- The first comprehensive survey that sorts out existing literature on MetaBBO. We provide a clear categorization of existing MetaBBO works according to four distinct meta-level tasks, along with a detailed elaboration of four different learning paradigms behind.
- A proof-of-principle evaluation is conducted to provide practical comparison between MetaBBO works, leading to an in-depth discussion over several key design strategies related to the learning effectiveness, training efficiency, and generalization.
- In the end of this paper, we mark several interesting and promising future research directions of MetaBBO, focusing different aspects such as the generalization potential, the end-to-end workflow, and the integration of LLMs.

II. Definition of MetaBBO

MetaBBO [26] is derived from the Meta-learning paradigm [27], applicable to the learning of both models and algorithms [38]. For example, in [59], a novel Recurrent Neural Network (RNN) model architecture is meta-learned and subsequently used as a classifier. In contrast, in [60], a RNN model is meta-learned to serve as a gradient descent algorithm for optimizing other neural networks. The paradigm in [60] quickly becomes popular and the following works explore the possibility of such a paradigm in various white-box optimization scenarios ranging from first-order optimization [60] to combinatorial optimization [61]. To make a distinction with other applications of Learning to Learn, this research line is named by Learning to Optimize (L2O). MetaBBO draws key inspiration from the L2O, while targeting black-box optimization scenarios. In this section, we provide an overview of the abstract workflow shared by existing MetaBBO methods, explaining the motivation of the core components in MetaBBO. MetaBBO operates within a bi-level framework, as depicted in Fig. 2, and is detailed as follows.

We begin with the low-level BBO process. A key component at this level is the low-level optimizer $\mathcal{A}$. $\mathcal{A}$ represents a flexible concept, capable of being any off-the-shelf EC algorithm, its modern variants, an algorithm pool, or a structure for creating new algorithms (rather than a specific existing one). Another crucial element is the optimization problem distribution $\mathcal{P}$, representing a collection of optimization problem instances to be solved. Although the size of $\mathcal{P}$ could theoretically be infinite, facilitating Meta-learning on an infinite problem set is impossible. In practice, we instead sample a collection of N instances $\{f_1, f_2, ..., f_N\}$ from $\mathcal{P}$ as the training set. A meta task $\mathcal{T}$ aims to automatically dictate an algorithm design $\omega \in \Omega$ for the low-level optimizer $\mathcal{A}$ for each problem instance in $\mathcal{P}$, where Ω denotes the algorithm design space of $\mathcal{A}$. For instance, in a basic DE optimizer [62], its algorithm configuration (e.g., values of the two hyper-parameters F and Cr that control the mutation and crossover strength) can be regarded as an algorithm design space Ω. There are various algorithm design spaces, which are discussed in detail in Section III. MetaBBO solves the meta task by learning a meta-level policy π_θ for the algorithm decision.

Formally, for a meta-level algorithm design task $\mathcal{T} := \{\mathcal{P}, \mathcal{A}, \Omega\}$, π_θ is trained to maximize the meta-objective $J(\theta)$:

$$J(\theta) = \mathbb{E}_{f \in \mathcal{P}} \left[\mathrm{R}\left(\mathcal{A}, \pi_\theta, f\right) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathrm{perf}(\mathcal{A}, \omega_i^t, f_i)$$
$$\omega_i^t = \pi_\theta(s_i^t), \quad s_i^t = \mathrm{sf}(\mathcal{A}, f_i, t) \tag{1}$$

where $\mathrm{sf}(\cdot)$ is a state feature extraction function, which captures the optimization state information from the interplay between the optimizer $\mathcal{A}$ and the problem instance f_i. The meta-level policy π_θ is parameterized by learnable parameters θ. It receives s_i^t as input and outputs an algorithm design ω_i^t, which is then adopted by $\mathcal{A}$ to optimize f_i. A performance measurement function $\mathrm{perf}(\cdot)$ is used to evaluate the performance gain obtained by this algorithm design decision. $\mathrm{R}(\cdot)$ is accumulated performance gain during the low-level optimization of a problem instance. We approximate the meta-objective $J(\theta)$ as the average performance gain across a group of N problem instances sampled from $\mathcal{P}$, over a certain number T of optimization steps. To summarize, MetaBBO aims to search for an optimal meta-level policy π_{θ^*} which maximizes the meta-objective $J(\theta)$.

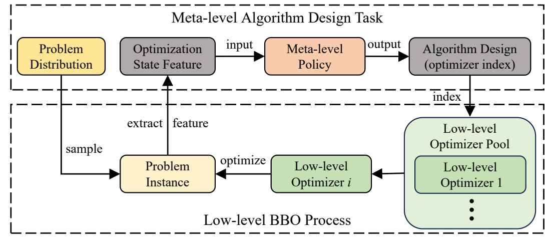

Fig. 3. Workflow of MetaBBO for Algorithm Selection.

III. Categorization of MetaBBO by Meta Tasks

We introduce four common meta-level tasks in MetaBBO: Algorithm Selection in Section III-A, Algorithm Configuration in Section III-B, Solution Manipulation in Section III-C and Algorithm Generation in Section III-D. Generally speaking, they are organized by the size of design space, from smallest to largest. Algorithm selection deals with a small space, choosing from a few BBO optimizers, while algorithm generation explores a vast space, allowing the meta-level policy to create novel BBO optimizers in an open-ended manner. Table I presents a selection of works categorized by the meta tasks, along with their references, publication years, low-level optimizers, targeted problem types[1], and technical summaries.

A. Algorithm Selection

Algorithm Selection (AS) has been discussed for decades [127]. The goal of AS is to select the most suitable algorithm from the algorithm pool according to the target task. The motivation of AS is that optimization behaviors and preferred scenarios vary with the algorithms, resulting in a notable performance difference [128]. Initially, AS is performed by human experts, who suggest algorithms based on their knowledge, which is labor-intensive and requires extensive expertise. To alleviate this dependency, researchers seek to develop more automated approaches.

*1) **Formulation**:* We examine the common AS paradigm. In the low-level BBO process, the component $\mathcal{A} = \{\mathcal{A}_1, ...\mathcal{A}_\mathrm{K}\}$ represents an algorithm pool $\mathcal{A}$ with K candidate BBO algorithms. The algorithm design space $\Omega = \{1, 2..., K\}$ is the selective space involving all indexes of the candidate algorithms, where $\omega \in \Omega$ denotes an index of a candidate from $\mathcal{A}$. For each problem instance f_i in the training set, the goal of AS is to output an algorithm decision ω_i^t for f_i at each optimization step t. As illustrated in Fig. 3, MetaBBO automates this task by maintaining a learnable meta-level policy

[1]We use SOP, MOOP, COP, CMOP, MMOP, MMOOP, LSOP, LS-MOOP, MILP and CO to denote single-objective optimization, multi-objective optimization, constrained optimization, constrained multi-objective optimization, multi-modal optimization, multi-modal multi-objective optimization, large-scale optimization, large-scale multi-objective optimization, mixed integer linear programming and combinatorial optimization, respectively.

TABLE I

REPRESENTATIVE WORKS IN METABBO, CATEGORIZED BY DIFFERENT ALGORITHM DESIGN TASKS. WE HAVE PROVIDED AN ONLINE PAGE, WHERE MORE DETAILS OF EXISTING METABBO WORKS ARE INCLUDED.

	Algorithm	**Year**	**Low-level Optimizer**	**Optimization Type**	**Technical Summary**
Algorithm Selection	Meta-QAP [63]	2008	MMAS	CO	per-instance algorithm selection by MLP classifier for Quadratic Assignment Problem (QAP)
	Meta-TSP [64]	2011	GA	CO	per-instance algorithm selection by MLP classifier for Travelling Salesman Problem (TSP)
	Meta-MOP [65]	2019	MOEA	MOOP	per-instance algorithm selection by SVM classifier from ten multi-objective optimizers
	Meta-VRP [66]	2019	MOEA	CO	per-instance algorithm selection by MLP classifier from four multi-objective optimizers
	AR-BB [67]	2020	EAs, SI	SOP	per-instance algorithm selection by symbolic problem representation and LSTM autoregressive prediction
	ASF-ALLFV [68]	2022	EAs, SI	SOP	per-instance algorithm selection by adaptive local landscape feature and KNN classifier
	AS-LLM [69]	2024	-	SOP	per-instance algorithm selection leverage embedding layer in LLMs
	HHRL-MAR [70]	2024	SI	SOP	dynamically switch SI optimizers along the optimization process with a Q-table RL agent
	R2-RLMOEA [71]	2024	EAs	MOOP	dynamically switch 5 EA optimizers along the optimization process with an MLP RL agent
	RL-DAS [34]	2024	DE	SOP	dynamically switch 3 DE optimizers along the optimization process with an MLP RL agent
	TransOptAS [72]	2024	EAs, SI	SOP	per-instance algorithm selection by Transformer performance predictor from single-objective optimizers
Algorithm Configuration	RLMPSO [73]	2016	PSO	SOP	dynamically select PSO update rules
	QFA [74]	2018	FA	SOP	tuning two control parameters of Firefly Algorithm (FA).
	RL-MOEA/D [75]	2018	MOEA/D	MOOP	dynamically control the neighborhood size and the mutation operators used in MOEA/D
	QL-(S)M-OPSO [76]	2019	PSO	SOP,MOOP	dynamically control the parameters of PSO update rule
	DE-DDQN [28]	2019	DE	SOP	mutation operator selection in DE
	DE-RLFR [77]	2019	DE	MMOOP	mutation operator selection in DE for multi-modal multi-objective problems
	LTO [78]	2020	CMA-ES	SOP	dynamically configure the mutation step-size in CMA-ES
	QLPSO [30]	2020	PSO	SOP	dynamically control the inter-particle communication topology of PSO
	MARLwCMA [79]	2020	DE	SOP	mix the strength of mutation adaptation and CMA-ES
	LRMODE [80]	2020	DE	MOOP	incorporate landscape analysis to operator selection
	RLDE [81]	2021	DE	SOP	dynamically adjust the scaling factor F in DE
	LDE [31]	2021	DE	SOP	use LSTM to adaptively control F and CR in DE
	RLEPSO [82]	2021	PSO	SOP	dynamically adjust factors in EPSO
	qlDE [83]	2021	DE	SOP	dynamically determine parameter combinations of F and Cr in DE
	DE-DQN [33]	2021	DE	SOP	mutation operator selection
	RLEA-SSC [84]	2021	DE	MMOP	dynamically determine where to search
	RL-PSO [85]	2022	PSO	SOP	dynamically adjust random values in PSO update rule
	RLLPSO [86]	2022	PSO	LSOP	adaptively adjust the number of performance levels in the population.
	MADAC [87]	2022	MOEA/D	MOOP	dynamically adjust all parameters in MOEA/D by an multi-agent system
	RL-CORCO [88]	2022	DE	COP	operator selection in constrained problems
	MOEA/D-DQN [89]	2022	MOEA/D	MOOP	leverage DQN to select variation operators in MOEA
	RL-SHADE [90]	2022	SHADE	SOP	perform mutation operator selection in SHADE
	RL-HPSDE [29]	2022	DE	SOP	control parameter sampling method and mutation operator selection
	NRLPSO [91]	2023	PSO	SOP	dynamically adjust learning paradigms and acceleration coefficients
	Q-LSHADE [92]	2023	DE	SOP	dynamically control when to use the scheme to reduce the population.
	LADE [93]	2023	DE	SOP	leverage three LSTM models to generate three sampling distributions of key parameters in DE
	LES [42]	2023	CMA-ES	SOP	use self-attention mechanism to adjust the step size in CMA-ES
	RLAM [94]	2023	PSO	SOP	enhance the PSO convergence by using RL to control the coefficients of the PSO
	MPSORL [95]	2023	PSO	SOP	adaptivly select strategy in multi-strategy PSO
	RLDMDE [96]	2023	DE	SOP	adaptively select mutation strategy of each population in multi-population DE
	RLMMDE [97]	2023	MOEA	MOOP	dynamically determine whether to perform reference point adaptation method
	MARLABC [98]	2023	ABC	SOP	dynamically select optimization strategy
	CEDE-DRL [99]	2023	DE	COP	dynamically select suitable parent population
	AMODE-DRL [100]	2023	MODE	MOOP	two RL agents, one for mutation operator selection, one for parameter tuning
	RLHDE [101]	2023	DE	SOP	use Q-learning to select mutation operators in QLSHADE and control the trigger parameters in HLSHADE
	GLEET [32]	2024	PSO,DE	SOP	dynamic hyper-parameters tuning based on exploration-exploitation tradeoff features
	RLMODE [102]	2024	DE	MOOP	dynamically control the key parameters in DE update rule
	RLNS [103]	2024	SSA,PSO,EO	MMOP	dynamically adjust the subpopulation size
	ada-smoDE [104]	2024	DE	SOP	dynamically control the key parameters in DE update rule
	PG-DE [105]	2024	DE	SOP	dynamic operator selection
	SA-DQN-DE [106]	2024	DE	MMOP	dynamically select proper local search operators
	RLEMMO [48]	2024	DE	MMOP	dynamically select DE mutation operators
	MRL-MOEA [107]	2024	MOEA	MOOP	dynamically select crossover operator in MOEA
	MSoRL [108]	2024	PSO	LSOP	automatically estimate the search potential of each particle
	UES-CMAES-RL [109]	2024	UES CMAES	SOP	determine parameters in restart strategy by RL agent
	HF [110]	2024	DE	SOP,CO	dynamically select DE mutation operators by RL agent or manual mechanism
	MTDE-L2T [49]	2024	DE	MTOP	control parameter F in DE
Solution Manipulation	RNN-OI [38]	2017	-	SOP	use RNN as a BBO algorithm to output solutions iteratively
	RNN-Opt [39]	2019	-	SOP	using RNN as a algorithm to output sample distribution iteratively
	LTO-POMDP [41]	2021	-	SOP	LSTM-based optimizer to output per-dimensional distribution
	MELBA [111]	2022	-	SOP	use Transformer-based model to output sample distribution
	LGA [43]	2023	GA	SOP	use attention mechanism to imitate crossover and mutation in GA
	OPRO [112]	2023	-	SOP	use LLMs as optimizer to output solutions
	LMEA [113]	2023	-	SOP	use LLMs to select parent solutions and perform crossover and mutation to generate offspring solutions
	MOEA/D-LLM [114]	2023	MOEA/D	MOOP	use LLMs as the optimizer in MOEA/D process
	ELM [115]	2023	-	CO	use LLM agent to generate benchmark programs through evolution of existing ones
	ToLLM [116]	2023	-	SOP	prompt LLMs to generate solutions
	GLHF [40]	2024	DE	SOP	use neural network to imitate mutation and crossover in DE
	B2Opt [117]	2024	GA	SOP	use neural network to imitate operators in GA
	RIBBO [37]	2024	-	SOP	use GPT model to output optimization trajectories
	EvoLLM [45]	2024	-	SOP	imitate ES's optimization behaviour by iteratively prompting LLM
	EvoTF [36]	2024	-	SOP	use Transformer-based network to output ES's distribution parameter
	LEO [118]	2024	-	SOP	exploitation via LLM instead of crossover and mutation
	CCMO-LLM [119]	2024	-	CMOP	use LLM as the search operator within a classical CMOEA framework
Algorithm Generation	GSF [120]	2022	-	CO	generate whole BBO algorithm by using RL agent to select operators from fixed algorithmic template
	AEL [121]	2023	-	CO	use LLM to evolve algorithm source code
	EoH [46]	2023	-	CO	use LLM agent to evolve algorithm's thoughts and source code
	SYMBOL [122]	2024	-	SOP	automatically generate symbolic update rules along optimization process through LSTM
	LLaMEA [123]	2024	-	SOP	use LLM to evolve EA algorithm
	LLMOPT [124]	2024	-	MOOP	use LLM to evolve operators for multi-objective optimizer
	LLaMoCo [35]	2024	-	SOP	instruction-tuning for LLM to generate accurate algorithm code
	OptiMUS [47]	2024	-	MILP	develop multi-agent pipelines for LLM to solve MILP problem as a professional team
	LLM-EPS [125]	2024	-	-	use LLM to generate offspring codes in evolutionary program search
	ALDes [126]	2024	-	SOP	sequentially generate each component in an algorithm through auto-regressive inference

π_θ with parameters θ, which takes a state feature s_i^t obtained by $\mathrm{sf}(\cdot)$ describing the optimization state of this optimization step, and then outputs ω_i^t. The selected candidate algorithm $\mathcal{A}[\omega_i^t]$ is used to optimize f_i in the low-level BBO process. Its performance on f_i serves as the performance measurement in Eq. (1). MetaBBO aims to find an optimal meta-level policy that suggests a best-performing algorithm in $\mathcal{A}$ for each f_i at each optimization step t automatically. Suppose the optimization horizon of the low-level BBO process is T, the meta-objective $J(\theta)$ of AS is calculated as:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\mathrm{perf}(\mathcal{A}[\omega_i^t], f_i) \tag{2}$$

After training, π_θ is expected to select well-matched candidate algorithms from $\mathcal{A}$ for unseen problems.

*2) **Related Works**:* First, per-instance AS is widely adopted in the literature, where a single algorithm is selected for the entire optimization progress for each specific problem, meaning that ω_i^t remains time-invariant. A straightforward approach to learning an effective meta-level policy for the AS task is to form a logical association between the attributes of f_i and the algorithm selection decision ω_i that corresponds to them. Since typically the number of candidate algorithms in the pool $\mathcal{A}$ is finite, many early-stage MetaBBO for AS researches transformed the meta-level learning process to a classification task [63]–[69], [129], [130]. In their methodologies, the state feature extraction function $\mathrm{sf}(\cdot)$ in Eq. (1) extracts problem characteristics s_i of f_i, which is significant enough to distinguish f_i with the other problem instances. A benchmarking process is employed to identify the top-performing candidate algorithm for f_i. The identified algorithm is then used as the classification label. The meta-level policy π_θ is regarded as a classifier and hence meta-trained to achieve maximum prediction accuracy. The state feature extraction mechanism $\mathrm{sf}(\cdot)$ in these works can be very different according to the target optimization problem types. Meta-QAP [63], Meta-TSP [64] and Meta-VRP [66] construct an information collection termed as meta data for combinatorial optimization problems, which maintains the nodes information, edge connections in the graph and constraints of a problem instance. For continuous single/multi-objective optimization problem, exploratory landscape analysis techniques are adopted in [65], [68], [129], [130], which profiles the objective space characteristics of a problem instance such as the pareto dominance, convexity, peaks and valleys. These works mainly apply basic classification models such as Support Vector Machine (SVM), K-Nearest Neighbors (KNN) and Multi-Layer Perceptron (MLP) for the label prediction. In contrast, to achieve in-depth data mining of the relationship between the problem structures and the optimizer performance, The study in [67] uses symbolic regression techniques to recover the mathematical equation of the given problem and then leverages a Long Short-Term Memory (LSTM) [131] to auto-regressively predict the desired candidate algorithm. TransOptAS [72] explores the possibility of constructing a performance indicator based solely on the raw objective values to eliminate the computation cost for computing $\mathrm{sf}(\cdot)$. It leverages a Transformer [132]-styled architecture that takes a batch of sampled objective values as input and outputs the performance of the candidate algorithms through supervision under the benchmark results. AS-LLM [69] leverages pre-trained LLM embeddings to extract features from the candidate algorithms and the target optimization problem, then selects the best algorithm by feature similarities.

Several latest MetaBBO works explored the possibility of extending per-instance AS to dynamic AS during the low-level BBO process [34], [70], [71]. Concretely, the meta-level algorithm design task in this paradigm turns to flexibly suggest one candidate algorithm to optimize f_i for each optimization step t. The dynamic AS is regarded as Markov Decision Process in the mentioned MetaBBO works and hence can be maximized by using RL to meta-learn an optimal policy. The optimization state feature in RL-DAS [34] includes not only the problem properties but also the dynamic optimization state information to support such flexible algorithm switch. The optimization performance of its learned AS policy is superior to each state-of-the-art individual DE variant in its algorithm pool, which demonstrates the effectiveness of using MetaBBO for dynamic AS.

*3) **Challenges**:* While past research has made progress in AS, several technical challenges persist:

- The construction of the algorithm pool $\mathcal{A}$ requires deep expertise on the target problem distribution and promising BBO algorithms. A powerful pool should contain diverse BBO algorithms to address problems with different characteristics. Future work could explore measuring algorithm diversity and automating pool construction.
- For per-instance AS, labeling the training set is expensive due to the exhaustive search needed to find the optimal algorithm for each instance. Limited candidates and problem instances lead to generalization issues. In dynamic AS, the increased methodological complexity challenges the learning effectiveness of RL methods.
- The algorithm design space in MetaBBO for AS is coarse-grained, limited by the performance of individual algorithms without tuning their configurations. In the next section, we introduce algorithm configuration tasks, which offer larger and more fine-grained design spaces.

B. Algorithm Configuration

Algorithm configuration (AC) is a key task in optimization, since almost all BBO algorithms possess hyper-parameters [133] and optional operators [134] that affect performance. To automate the AC task, various adaptive and self-adaptive BBO algorithms have been developed in the past decades [135]. Algorithms like JADE [19] and APSO [20] leverage historical optimization data to compute informative decision statistics such as the potential of the hyper-parameter values and the success rates of the optional operators [136]–[138]. However, as discussed in the introduction, these approaches suffer from design bias, limited generalization, and high labor costs.

*1) **Formulation**:* As shown in Fig. 4, MetaBBO overcomes the limitations of manual AC techniques by using meta-learning to develop a meta-level configuration policy. This policy dynamically adjusts a BBO algorithm throughout the lower-level BBO procedure. More formally: in the low-level BBO process, the optimizer $\mathcal{A}$ represents the BBO algorithm to be configured. The algorithm design space Ω is hence the configuration space of $\mathcal{A}$. The size of Ω can be either infinite (with continuous hyper-parameters) or finite (with discrete hyper-parameters or several optional operators). MetaBBO dictates AC in a dynamic manner: given a problem instance f_i, at each optimization step t of the low-level BBO process, a state feature s_i^t is obtained by $\mathrm{sf}(\cdot)$ to describe the state of this optimization step. The meta-level policy $\pi_\theta\left(s_i^t\right)$ outputs the algorithm design ω_i^t, which sets the configuration of $\mathcal{A}$ as $\mathcal{A}.\mathrm{set}\left(\omega_i^t\right)$. Then the algorithm is used to optimize f_i for the current optimization step. Suppose the optimization horizon of the low-level BBO process is T, the meta-objective $J(\theta)$ of MetaBBO for AC is formulated as

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\mathrm{perf}(\mathcal{A}.set(\omega_i^t), f_i) \tag{3}$$

MetaBBO for AC improves on human-crafted adaptive methods by meta-learning the configuration policy through optimizing the meta-objective in Eq. (3), removing the need for labor-intensive, expert-driven designs. The bi-level meta-learning paradigm also enhances generalization, as the policy can be trained on a large set of problem instances, distilling configuration strategies that can be applied to new problems.

2) ***Related Works:*** In this paper, we further divide existing MetaBBO for AC works into three sub-categories according to the configuration space Ω of the low-level BBO algorithm. The first sub-category is hyper-parameter optimization (HPO), where the hyper-parameter values are controlled by the meta-level policy. The second is adaptive operator selection (AOS), where several optional operators is flexibly selected by the meta-level policy. The last is the combination of HPO and AOS, where Ω is a complex configuration space including both hyper-parameters and operators. Next we introduce related works in these sub-categories.

a) Hyper-parameter Optimization: Several early attempts meta-learn a configuration policy that dictates a single hyper-parameter setting throughout the entire process of solving a problem instance [139], [140]. Now, most MetaBBO for AC approaches follow the dynamic AC paradigm in Eq. (3), offering a flexible exploration-exploitation tradeoff to further improve the optimization performance. Since different BBO algorithms have distinct hyper-parameters, existing MetaBBO for AC works customize their methods to explore the intricate relationships between the hyper-parameters and the resulting exploration-exploitation tradeoff in each specific algorithm.

Since DE is known to be highly sensitive to hyperparameter settings, particularly the scaling factor F and the crossover probability Cr, many efforts have focused on meta-tuning DE. RLDE [81] propose a simple Q-table policy to adjust F when optimizing the power generation efficiency in solar energy system. It uses a Boolean indicator as the optimization state feature: indicating whether the solution quality is improved between two optimization steps. The algorithm design space, represented as $\delta F \in \{-0.1, 0, 0.1\}$, indicates the variation

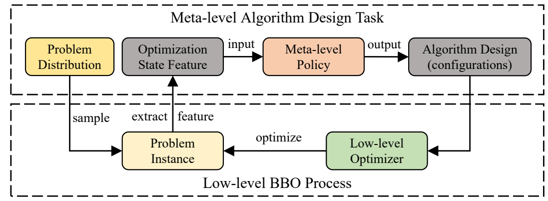

Fig. 4. Workflow of MetaBBO for Algorithm Configuration.

in F for the subsequent optimization step. Following RLDE, QLDE [83] extends the algorithm design space to five combinations of the parameter values. RLMODE [102] further proposes a specific state extraction function for constrained multi-objective optimization, which divided the state feature into eight possible situations, according to the solution feasibility and the dominance relationship. The algorithm design space is three combinations of different F and Cr settings to represent different exploration-exploitation tradeoffs. The Q-table agent is updated by first selecting the most promising combination and then observing the resulting performance improvement. For more fine-grained parameter control, LDE [31] firstly considers using recurrent neural network (i.e., LSTM) as the meta-level policy, which extracts hidden state feature for separate optimization step and outputs the values for F and Cr from a continuous range $[0, 1]$. The same authors subsequently propose LADE [93] as a extension of LDE. Compared to LDE, LADE aims to control more hyper-parameters including not only the mutation strength and crossover rate but also the update weights. All parameters are represented as matrix operations. Instead of using one LSTM for controlling all parameters, LADE's meta-level policy comprises three LSTM networks for controlling these parameters respectively. LADE show more robust learning effectiveness than LDE. A recent study, L2T [49], employs the MetaBBO framework to regulate the setting of DE parameters and the likelihood of knowledge transfer within the multitask optimization working scenarios. The state feature is represented by the rate of successful transfers and the enhancement in sub-population performance. Despite adapting F and Cr, the control of population size is considered in Q-LSHADE [92]. The algorithm design space is the decay rate of the linear population size reduction in LSHADE [141], which can take values from $\{0, 0.2\}$. Q-LSHADE also meta-learns a Q-table policy by the feedback indicating the performance improvement. There are also several MetaBBO works which facilitate hyper-parameter optimization on other algorithms, such as PSO [30], [32], [76], [82], [85], [86], [94], [103], ES [42], [78], and the Firefly algorithm [74]. In addition, a recent work GLEET [32] proposes a general learning paradigm which show generic HPO ability for both DE and PSO. Due to the space limitation, other related works are summarized in Table I.

b) Adaptive Operator Selection: The works in this line aims to dynamically switch the operators of the low-level BBO algorithms during the optimization process. The majority of them still focus on DE algorithms [28], [33], [79], [80], [88]–[90], [96], [101], due to their strong performance and the availability of various operators for selection. These works share similar methodologies: a mutation operator pool is maintained, involving representative mutation operators such as DE/rand/2, DE/best/2, DE/current-to-rand/1, DE/current-to-best/1 and DE/current-to-pbest/1. In order to address different types of problems, the technical differences in these works revolve around the tailored state feature extraction design and the operator pool. The state feature extraction functions in these works can be divided into two main strategies: discrete representation and continuous representation.

For discrete state representation, the study in [79] first computes the diversity variation and the performance improvement between two consecutive optimization steps as an effective profile of the optimization dynamics. These two indicators, being continuous variables, are then divided into five distinct levels each. According to the discretized state feature, a Q-table policy is constructed to select one operator from an operator pool with three candidates. RLHDE [101] uses the relative density in the solution space and the objective space against the initial population and objective values to indicate the convergence trend and the performance improvement. The values of the two density indicators are discretized into five and four levels respectively, constituting 20 different optimization states. The operators pool in RLHDE involves six mutation operators, which improve the diversity of the optimization behaviours, hence strengthening the generalization. RL-CORCO [88] addresses constrained multi-objective optimization by enhancing the CORCO algorithm through multiple Q-table policies. In the algorithm, each sub-population maintains a Q-table, where rows represent nine states indicating different levels of objective improvement and constraint violation, and columns represent two mutation operators. The policy selects the appropriate mutation operator to optimize the solution as effectively as possible.

Compared to discrete features, continuous state feature extraction enables finer state modeling, providing unique representations for optimization states and leading to smarter decisions by the meta-level policy. For instance, DE-DDQN [28] proposes a very comprehensive optimization state extraction function, which computes a total of 99 features: the first 19 features describe the optimization progress and the properties of the target optimization problems, while the rest 80 are statistics describing the optimization potential of the four mutation operators in the operator pool. An MLP neural network-based meta-level policy generates Q-values for the candidate mutation operators and the one with maximal Q-value is chosen for the next optimization step. Following DE-DDQN, DEDQN [33] and MOEA/D-DQN [89] also construct MLP policies. DEDQN indicates that the features in DE-DDQN show certain redundancy and might fall short in capturing the local landscape features. To address this, DEDQN proposes a feature extraction mechanism inspired from classical fitness landscape analysis [142]. By using random walk sampling, DEDQN computes ruggedness and fitness distance correlations in the local landscape. Results show that landscape features are effective for MetaBBO methods to generalize across problem types. For addressing multi-objective optimization problem, MOEA/D-DQN embeds the information of the reference vectors in MOEA/D into the state extraction. In particular, for a solution x of which the corresponding reference vector has weights w, the state vector is combining the solution with the weights: $\{x, w\}$. The meta-level policy receives this state feature and then suggests an operator combination for the next optimization step. There are four operator combinations in the algorithm design space of MOEA/D-DQN, involving two mutation operators and two crossover operators. To tackle multi-modal optimization problem, RLEMMO [48] first clusters solutions to compute the neighbourhood features. The optimization state is then constructed by concatenating the solution's optimization progress, population's distributional features, and neighbourhood features. RLEMMO designs an operator pool with five diverse mutation operators, showing low-to-high degrees of of exploration-exploitation tradeoff. A Transformer-like policy is adopted to enhance the information sharing during the population evolution.

c) Hybrid Control: Some MetaBBO works explore other AC perspectives [105], [110]. In particular, the combination of HPO and AOS has gained significant attention [29], [75], [87], [91], [100], [101], since learning a meta-level policy in $\Omega_{\text{HPO+AOS}}$ would probably result in a better AC policy than learning them separately. Nevertheless, this poses a significant challenge as learning from an expanded algorithm design space necessitates more intricate learning strategies and model frameworks. Thoughtful design is essential to guarantee effective learning.

*3) **Challenges**:* Despite their success, existing MetaBBO works for AC still face some challenges.

- A certain proportion of existing methods use a very limited set of training problems. In particular, some only train their meta-level policies on a specific optimization problem instance, raising doubts about the actual generalization performance of the resulting policies.
- MetaBBO for AC works operate on the basis of predefined low-level BBO algorithms. Hence, the performance of these methods is closely tied to the original BBO algorithm. Furthermore, the inherent algorithm structures, optimization logic, and design biases significantly restrict the algorithm design space. Can we further expand the algorithm design space and step out this boundary? In the next two subsections, we introduce two novel categories of MetaBBO works that offer potential solutions.

C. Solution Manipulation

So far, we have introduced two basic categories of MetaBBOAC. An intuitive observation is that within the MetaBBO framework for AS/AC tasks, the low-level BBO procedure necessitates a BBO algorithm as the foundational optimizer, which comes with a defined algorithm design space (e.g., algorithm pool or configuration space). This leads to two limitations. First, it requires expert knowledge to select an appropriate BBO algorithm, otherwise the meta-level policy's learning effectiveness and overall performance may suffer. Second, managing both the meta-level policy and the low-level BBO optimizer simultaneously incurs certain computational costs. To address these limitations, several MetaBBO works have explored the potential of directly using the meta-level policy for solution manipulation. This approach integrates meta-level training and low-level optimization into a single entity, eliminating the need for a predefined BBO algorithm. In this framework, the meta-level policy itself functions as an optimization algorithm, directly manipulating candidate solutions throughout the optimization process. We illustrate this MetaBBO workflow in Fig. 5, referring to it as MetaBBO for solution manipulation (SM).

*1) **Formulation**:* To formulate the process of solution manipulation in MetaBBO, some clarifications have to be made. First, MetaBBO for SM integrates the functions of meta-level policy and the low-level BBO algorithm into a single parameterized agent π_θ, removing the need for a traditionally perceived BBO algorithm. Therefore, the meta-level policy π_θ, typically a neural network, inherently serves as the BBO algorithm. In this case, the algorithm design space Ω turns to the parameter space of the policy, where each algorithm design ω in this space corresponds to the values of the neural network parameters θ. Given a problem instance f_i, at each optimization step t, the optimization state feature s_i^t is first computed by $\mathrm{sf}(\cdot)$. According to s_i^t, the policy (acts as the BBO algorithm) π_θ optimizes f_i for one optimization step, e.g., reproducing the candidate solutions. The performance improvement is hence measured as $\mathrm{perf}(\pi_\theta(s_i^t), f_i)$. Suppose the optimization horizon of the low-level BBO process is T, the meta-objective of MetaBBO for SM is formulated as

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathrm{perf}(\pi_\theta(s_i^t), f_i) \tag{4}$$

Through maximizing $J(\theta)$ over N problem instances in the training set, a neural network-based BBO algorithm is obtained, functioning similarly to human-crafted BBO algorithms: iteratively optimizes the problem instances. Next, we next introduce representative MetaBBO works for SM.

*2) **Related Works**:* An intuitive way of resembling the iterative optimization behaviour by neural networks is considering temporal network structure such as recurrent neural networks [38], [39], [41], which enable MetaBBO to directly adjust candidate solutions over sequential steps. The corresponding mathematical formulation is quite straightforward:

$$X^t, h^t = \pi_\theta(X^{t-1}, Y^{t-1}, h^{t-1}), \quad Y^t = f_i(X^t) \tag{5}$$

where π_θ is an RNN/LSTM, h^t is the hidden state. This paradigm is first adopted in RNN-OI [38], which meta-learns an LSTM to reproduce candidate solutions. For each f_i in the training problem set, RNN-OI randomly initializes a solution X^0, obtains the corresponding objective values Y^0, and then optimizes f_i by iteratively inferring the next-step solution. To meta-learn a well-performing π_θ, the observed improvement per step is computed as the $\mathrm{perf}(\cdot)$ function. Once trained, the LSTM serves as a BBO algorithm and iteratively optimizes the target optimization problem following Eq. (5). Due to the end-to-end inferring process, RNN-OI is shown to be faster in terms of the running time compared to hand-crafted algorithms. Following RNN-OI, similar works include RNN-Opt [39] improving RNN-OI through input normalization and constraint-dependent loss function, LTO-POMDP [41] using neuroevolution to learn the network parameters, MELBA [111] improving the long sequence modelling of RNN/LSTM by introducing Transformer structure, and RIBBO [37] leveraging efficient and generic behaviour cloning framework to learn an optimizer that resembles the given teacher optimizer.

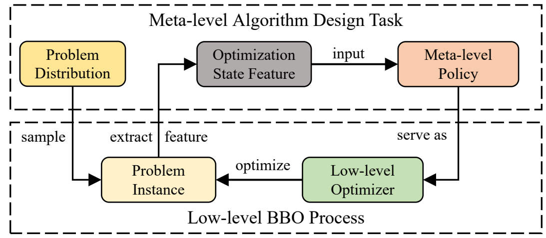

Fig. 5. Overview of MetaBBO for Solution Manipulation.

Nevertheless, the above works still suffer from generalization limitation and interpretability issues. On the one hand, the optimization state features only include the raw population information, which makes the policy easily overfits to the training problems. On the other hand, the learned policies in these works shift toward "black-box" systems, which hinders further analysis on what they have learned. In the last two years, several more interpretable MetaBBO for SM works are proposed to address these issues [36], [40], [43], [117]. These works propose using higher-level features as a substitute for the raw features to achieve generalizable state features across diverse problems. Typically, these features include the distributional characteristics of the solution space and the objective space, the rank of objective values, and the temporal features reflecting the optimization dynamics. They have proposed several novel architecture designs to make the meta-level policy explicitly resembles representative EC algorithms such GA [43], [117], DE [40], and ES [36]. For instance, LGA [43] designs two attention-based neural network modules to act as the selection and mutation rate adaption mechanisms in GA. The parameterized selection module applies cross-attention between the parent population and the child population, and the obtained attention score matrix is used as the selection probability. The parameterized mutation rate adaption module applies self-attention within the child population, and the obtained attention scores is used as the mutation rate variation strength. B2Opt [117] improves LGA by proposing a novel, fully end-to-end network architecture which resembles all algorithmic components in GA, including crossover, mutation, selection. For example, the selection module within B2Opt utilizes a method similar to the residual connection in Transformer, facilitating the use of matrix operations for selecting populations. By meta-training the proposed meta-level policies on the training problem set, these MetaBBO for SM works show competitive optimization performance. In particular, their meta-level policies are trained with low dimensional synthetic problems ($\leq$ 10) yet could be directly generalized for solving high dimensional continuous control problems ($>$ 500), e.g., neuroevolution [143].

With the emergence of LLMs, their ability to understand the reasoning in natural language outlines a novel opportunity for SM. Related works in this line widely leverage the In-Context Learning (ICL) [144] to prompt with general LLMs iteratively as an analog to BBO algorithms to reproduce solutions. A pioneer work is OPRO [112], which first provides LLMs a context of the problem formulation and historical optimization trajectory described in natural language. It then prompts LLMs to suggest better solutions based on the provided context. This idea soon becomes popular and spreads to multiple optimization scenarios such as program search [115] combinatorial optimization [113], multi-objective optimization [114], [119], large scale optimization problem [45], [118] and prompt optimization [44]. The eye-catching advantage of LLM-based SM is that it requires minimal expertise - users only need to describe the optimization problem in nature language, and LLMs handle the rest.

*3) **Challenges**:* As a novel direction, MetaBBO for SM is promising due to the end-to-end manner. However, several technical challenges remain:

- Approaches like RNN-Opt directly learn to manipulate candidate solutions without following a specific algorithm structure. While this provides flexibility, these methods often lack transparency and clear understanding of their inner workings. Additionally, due to the complexity of BBO tasks, exploring strong neural networks capable of handling diverse, complex problems remains a challenge.
- In contrast, methods like LGA closely mimic the structure and components of existing EAs, making the process more transparent. However, because these approaches resemble existing algorithms, their performance might be inherently constrained by the limits of the original methods.
- MetaBBO approaches that use LLMs, while reducing the need for manual algorithm design, face significant computational overhead. The iterative interactions with LLMs generate large volumes of tokens, leading to inefficiencies in both time and cost.
- Finally, MetaBBO for SM treats the policy itself as the optimizer, targeting at learning the optimal mapping from current landscape to next candidate positions. However, this remains a highly challenging task for continuous BBO tasks. The possible landscapes are diverse and infinite. As a result, so far, it is very challenging to build and train a model that can effectively handle these complexities in practice. In the next section, we will explore the "algorithm generation" approach, which leverages the meta-level policy as an algorithm discoverer, namely, using learning to create new algorithmic workflows, update rules, and implementations.

D. Algorithm Generation

Besides MetaBBO for SM, an interesting research question comes out: whether learning-based systems such as MetaBBO could automatically create (generate) new BBO algorithms with competitive optimization performance and minimal expertise requirement? To this end, MetaBBO for algorithm generation (AG) presents a different methodology: meta learning a parameterized policy that could discover novel algorithms accordingly without the human-expert prior, of which the workflow is illustrated in Fig. 6. The difference between AG and SM is that the meta-level policy in SM plays both the role of the meta-level policy and the low-level optimizer, while the meta-level policy in AG is trained to output a complete optimizer which is used then in the low-level BBO process.

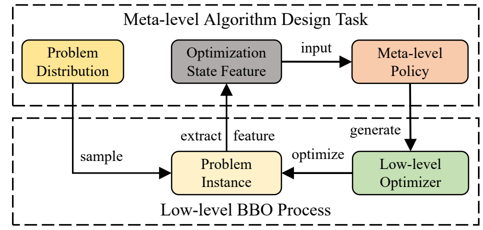

Fig. 6. Overview of MetaBBO for Algorithm Generation.

*1) **Formulation**:* MetaBBO for AG works construct an algorithm representation space Ω as its design space. For example, Ω can be a algorithm workflow space, a mathematical expression space or a programming language space, reflecting the way humans express algorithms - through modular algorithm workflows, symbolic mathematical expressions or programming language syntax. For a problem instance f_i, a concrete algorithm design ω_i^t is output by the meta-level policy π_θ, according to the optimization state feature s_i^t. The sf$(\cdot)$ function, in this case, can incorporate landscape features, symbolic representations, or natural language descriptions of f_i. The generated ω_i^t can be a complete workflow, a mathematical expression or a functional program that represents a novel BBO algorithm $\mathcal{A}$. the meta-objective of MetaBBO for AG is to meta learn a policy π_θ capable of generating well-performing algorithms:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\text{perf}(\omega_i^t, f_i), \quad \omega_i^t = \pi_\theta(s_i^t) \tag{6}$$

where perf(ω_i^t, f_i) is the one-step optimization performance gain of the generated algorithm on f_i. Through training the policy across a problem set, the policy is expected to automatically generate flexible and even novel BBO algorithms to address various optimization problems. Besides, note that MetaBBO for AG could work with varying granularity: a) generating a universal algorithm for all problems [46], b) generating customized algorithms for each problem [126], and c) generating flexible optimization rules that adapt to each step of the optimization process and each specific problem [120], [122]. In Eq. (6), we demonstrate the case c). In contrast, in the case a), a single algorithm ω is generated to serve as ω_i^t in Eq. (6). In case b), a problem-specific ω_i is generated to serve as ω_i^t for each optimization step in solving f_i.

*2) **Related Works**:* Creating a comprehensive algorithm representation space Ω is crucial for the meta-level policy to produce innovative and efficient BBO algorithms. Current MetaBBO methodologies for AG can be categorized into three types based on their formulation of algorithm representation space Ω: algorithm workflow composition, mathematical expressions, or natural/programming languages.

First, we introduce the works that perform algorithm workflow composition. GSF [120] first defines an algorithm template for EAs, then uses RL to fill each part of the template with operators from a predefined operator pool. ALDes [126] overcomes the limitation of using fixed-length template through autoregresive learning. It first tokenizes the common algorithmic components and the corresponding configuration parameters in EAs, as well as the execution workflows such as loop and condition. Then, the algorithm generation task turns into a sequence generation task of the tokens. Concretely, ALDes prepares three types of operators: four "selection for evolution" operators, six evolution operators and five "selection for replacement" operators, each is associated with some hyper-parameters. Given the property of the target optimization problem, a Transformer-style policy is used to auto-regressively select one desired operator and configure its hyper-parameter from the candidate pool of each operator type. The novel workflow generated by ALDes demonstrate superior performance compared to several traditional BBO algorithms.

Second, we introduce the works that leverage mathematical expression to formulate Ω. The motivation behind this line is that the design space of GSF and ALDes is highly dependent on manual engineering, which may limit the exploration of more novel algorithm structures. SYMBOL [122] addresses this issue by breaking down the update equations of BBO algorithms into atomic mathematical operators and operands. SYMBOL constructs a token set of common mathematical symbols used in EAs, such as $\{+, -, \times, x, x^*, x^-, x_i^*, \Delta x, x_r, c\}$. It then designs an LSTM-based policy which is capable of auto-regressively generating a sequence of these mathematical symbols. The generated sequence can be parsed into update equations for optimizing the low-level optimization problem. SYMBOL generates flexible update rules for each optimization step and each problem instance, bringing in certain self-adaptation capabilities. Comparison results show that SYMBOL achieves state-of-the-art performance among powerful optimizers.

Third, we introduce works that leverage natural language and programming language to define Ω. All works in this line leverage LLMs as their meta-level policies [35], [46], [47], [121], [123], [124]. The differences lie in the learning methodologies, the generation workflows and the target problem types. OptiMUS [47] leverages modular-structured LLM agents to formulate and solve (mixed integer) linear programming problems. There are four agents in OptiMUS: formulator, programmer, evaluator, and manager, which constitute an optimization expert team and automate the algorithm generation task through their cooperation. To enable more general-purpose algorithm generation, AEL [121] and EoH [46]AEL [121] and EoH [46] are inspired by the evolution capability of large models [115], prompting LLMs to perform mutation and crossover operations on code implementations of previous algorithms. After evolution, the best-so-far algorithm generated shows superior performance to human-crafted heuristics on combinatorial optimization problems. Subsequent works such as LLaMEA [123] and LLMOpt [124] generalize this paradigm to continuous BBO scenarios, and LLaMEA is shown to be capable of generating a more complex algorithm that is competitive with CMA-ES. Despite the above works, LLaMoCo [35] offers a novel perspective: instruction-tuning the general LLMs to act as an expert-level optimization programmer. LLaMoCo allows users to describe their specific optimization problems in Python/LaTex formulation, then it outputs the complete Python implementation of a desired optimizer for solving the given problems. To achieve this, a large-scale benchmarking is conducted to attain thousands of problem-solver pairs as the expert-level optimization knowledge. This knowledge is then injected into LLMs through instruction tuning. The experimental results in LLaMoCo demonstrates that a small model (e.g., codeGen-350M) could generate superior algorithm program to larger models which are not fine-tuned by LLaMoCo (e.g., GPT-4), underscoring that domain specific knowledge might be the key for LLMs to understand, reasoning and solve optimization problems.

*3) **Challenges**:* MetaBBO for AG works operate in a more expressive algorithm design space. The experimental results in some of these works demonstrate that the generated algorithms are on par with or even superior to human-crafted ones. The generated BBO algorithms can not only be applied to address optimization problems, but also be further analysed by human experts for novel insights in developing optimization techniques. Nevertheless, there are still several bottlenecks in existing works:

- As an early-stage research avenue, related works in this area are still limited. More studies are expected to further unleash the potential of MetaBBO for AG.
- For symbolic system-based generation frameworks such as ALDes and SYMBOL, the token sets are relatively small, which leads to limited representation capability. How to construct a comprehensive and expressive token set tailored for BBO algorithm, and how to ensure the learning effectiveness in the enlarged algorithm design space need further investigation.
- For LLM-assisted MetaBBO for AG, although the workflow promises an efficient development pipeline, the computational resources required to obtain a competitive BBO algorithm are substantial. Besides, these works rely heavily on the prompt engineering, since LLMs are sensitive to the prompts they receive.

IV. Different Learning Paradigms at Meta Level

We introduce four main learning paradigms behind existing MetaBBO works to help practitioners grasp the design motivation and principle when developing MetaBBO methods. The four learning paradigms are MetaBBO-RL, MetaBBO-SL, MetaBBO-NE and MetaBBO-ICL, which are introduced in subsequent subsections respectively.

A. MetaBBO-RL

RL [58] is an effective tool for solving Markov Decision Process (MDP) [145]. Existing MetaBBO-RL works model the meta-level algorithm design task as a MDP. In this case the environment is the low-level BBO process for a given problem instance f. The optimization state feature s is the state, the algorithm design space Ω is the action space, the performance metric $\mathrm{perf}(\cdot)$ is the reward function. In this way, the meta-objective we established in Eq. (1), Section II turns into the expectation of the accumulated reward in a normal MDP. Various RL techniques are adopted in existing works yet limited literature summarizes or analyses the motivation behind. We in this subsection provide a systematic overview of different design motivations.

1) ***Discrete State & Discrete Action:*** Tabular Q-learning [146] and SARSA [58] are two main value-based RL techniques which maintain a Q-table and iteratively update their entries through interaction with the environment. An important advantage of Tabular Q-learning and SARSA is that their simple Q-table structures show efficient convergence and certain effectiveness. However, such techniques are only suitable for MDP where the state space and the action space are both discrete (finite). Many MetaBBO-RL works adopt such techniques for the simplicity. In these works, several available optimization states and algorithm designs are pre-defined to serve as the rows and columns in Q-table respectively. At each optimization step t in the low-level BBO process, the meta-level policy suggest an algorithm design ω^t according to the s^t and Q-table. Then a transition $< s^t, \omega^t, \text{perf}(s^t, \omega^t, f), s^{t+1} >$ is obtained and the Q-table is updated as:

$$Q(s^t, \omega^t) = \text{perf}(s^t, \omega^t, f) + \gamma \max_{\omega \in \Omega} Q(s^{t+1}, \omega) \tag{7}$$

We examplify a representative work QLPSO [30] here. In QLPSO, the meta task is to dynamically adjust the topology of the particle swarm. QLPSO operates within a ring topology swarm, and the available optimization states in this case are $\{L2, L4, L8, L10\}$ which denotes the neighbourhood size of a particle . The corresponding four actions are either staying at current neighbourhood size or transforming to another size. Success topology adjustment leading to performance improvement is rewarded. Other MetaBBO-RL works which hold similar methodology includes RLNS [103], QFA [74], qlDE [83], RLMPSO [73], DE-RLFR [77], QL-(S)M-OPSO [76], MARL-wCMA [79], LRMODE [80], RLEA-SSC [84], RLDE [81], RL-CORCO [88], RL-SHADE [90], etc.

2) ***Continuous State & Discrete Action:*** Although Tabular Q-learning and SARSA are simple and effective, some MetaBBO scenarios pursue continuous optimization state to facilitate fine-grained algorithm design, Therefore, the MDP in these works hold infinite state space which is incompatible with Q-table structure. In this case, neural network-based Q-agent Q_θ is introduced to handle the continuous state features, e.g., DQN [147] and DDQN [148]. The Q-agent is updated by minimizing the estimation error between target and predicted Q-function, which is formulated as the loss below:

$$\begin{aligned} \text{Loss}(\theta) = \frac{1}{2}\Bigg[Q_\theta(s^t, \omega^t) - \Bigg(&\text{perf}(s^t, \omega^t, f) \\ &+\gamma \max_{\omega \in \Omega} Q_\theta(s^{t+1}, \omega)\Bigg)\Bigg]^2 \end{aligned} \tag{8}$$

We examplify a representative work DEDQN [33] here. In DEDQN, the optimization state feature comprises four continuous FLA indicator features obtained by random walking strategy. The meta-level policy is a MLP-based Q-agent with three hidden layers (10,10,12 neurons respectively). During the low-level BBO process, the Q-agent outputs Q-values of three candidate DE mutation operators and then dictates one for current optimization step. The performance improvement after this optimization step serves as a reward. This transition is used to update the Q-agent by Eq. (8). Other MetaBBO-RL works which hold similar methodology include R2-RLMOEA [71], DE-DDQN [28], MADAC [87], MOEA/D-DQN [89], CEDE-DRL [99], SA-DQN-DE [106], UES-CMAES-RL [109], HF [110].

3) ***Continuous State & Continuous Action:*** Following the success of RL techniques in continuous control [149], some MetaBBO-RL works turn to policy gradient-based RL (e.g., REINFORCE [150], A2C [151], PPO [152], etc.) for supporting both continuous states and algorithm designs. Compared to discrete ones, continuous space allows these works control the optimization behaviour in the low-level BBO process flexibly hence enhances the overall performance. In this case, a policy neural network π_θ is introduced which outputs a probability distribution over the algorithm design space by conditioning the optimization state. The gradient $\nabla_\theta J(\theta)$ used to update π_θ is computed as:

$$\nabla_\theta J(\theta) = -\nabla_\theta \log \pi_\theta(\omega^t \mid s^t) \left(\sum_{t'=t}^{T} \gamma^{t'-t} \text{perf}(s^{t'}, \omega^{t'}, f)\right) \tag{9}$$

We examplify a representative work GLEET [32] here. In GLEET, the optimization state feature is a structured feature collection including several low-level optimization state information (e.g., the density in solution/objective space, the performance improvement indicator etc.). A Transformer-style policy network (3 layers) is used to output the posterior Gaussian distribution of the coefficients $c1$ and $c2$ in PSO update rule, for each particle in the swarm. Then the concrete coefficient values are sampled from the output distribution for the current optimization step and the corresponding reward is given. Once an optimization episode is completed (T steps), π_θ is updated by the sum of the gradients of each step shown in Eq (9). Other MetaBBO-RL works which hold similar methodology include LTO [78], RLEPSO [82], LDE [31], RL-PSO [85], MELBA [111], MOEADRL [153], LADE [93], RLAM [94], AMODE-DRL [100], PG-DE [105], GLEET [32], RLEMMO [48], RL-DAS [34], SYMBOL [122].

B. MetaBBO-NE

Neuroevolution [154] is a subfield within machine learning where neural networks are evolved by EC methods instead of updated by gradient descent. In [7], Evolution Strategy (ES) is demonstrated as a scalable alternative of RL for MDP where actions have long-lasting effects. This inspires some researchers to develop MetaBBO methods with ES as the learning paradigm, we name these works as MetaBBO-NE. Concretely, the meta level maintains a policy population $\{\pi_{\theta_1}, \dots, \pi_{\theta_K}\}$ in parallel. Each π_{θ_k} is used as the meta-level policy to aid the algorithm design task on the training problem set. The fitness of each π_{θ_k} is computed as the average performance gain across problem instances in the training set. ES is used to iteratively update the parameter distribution of the meta-level policy. After the evolution, an optimal meta-level policy network π_{θ^*} is obtained. Representative MetaBBO-NE works include LTO-POMDP [41] and LGA [43]. For instance, in LGA, a group of attention-based neural networks are maintained at the meta-level, each of which serves as a neural GA. OpenAI-ES [7] is used to evolve $K = 32$ such networks over ten 10D synthetic functions in COCO [155]. The finally obtained π_{θ^*} achieves desirable generalization performance on high dimensional BBO problems.

C. MetaBBO-SL

MetaBBO works using supervised learning paradigm to meta-train the meta-level policy are closely related to one particular meta task: solution manipulation (SM). As we described in Eq (4), SM aims to learn an parameterized meta-level policy π_θ as the low-level optimizer. The optimization process proceeds by iteratively calling π_θ to optimize current solution population, as described in Eq. (5). A major difference between MetaBBO-SL and MetaBBO-RL is that, MetaBBO-SL works meta-train their policies through direct gradient descent on an explicit supervising objective, which resembles a regret minimization [156] of the objective function of the target optimization problem. We examplify this explicit supervising objective by introducing a latest work GLHF [40], which proposes a fully end-to-end MetaBBO method which acts like a DE algorithm. GLHF first unifies the computation of DE mutation and crossover operator as matrix operation, and then designs π_θ as two customized network modules LMM and LCM to analog the matrix mutation/crossover. The authors have come up with a unique design which uses Gumbel-Softmax function to make the discrete crossover module differentiable, refer to the original paper for more details. Given a solution population X^t at the optimization step t when optimizing a problem f, the π_θ in GLHF optimizes X^t to get offspring population: $X^{t+1} = \pi_\theta(X^t)$. The explicit supervising objective in this case is the objective value $f(X^{t+1})$ which serves as a regret function and should be minimized. Then the gradient used to update the policy at step t is computed as:

$$\nabla_\theta J(\theta) \propto \frac{\partial f(X^{t+1})}{\partial \pi_\theta} \cdot \frac{\partial \pi_\theta}{\partial \theta} \tag{10}$$

By minimizing this regret-based supervising objective, the meta-level policy is trained to perform effective optimization on the target problems. Note that in above gradient descent process, the problem instance f is required to be differentiable. However, this is somewhat impossible when the target optimization problems are BBO problems. In GLHF, the problems used for training are synthetic ones which are "white box" hence differentiable. Its experimental results reveal that training on such synthetic problems is sufficient to gain certain generalization toward realistic tasks. Despite GLHF, other works in this line include RNN-OI [38], RNN-Opt [39], B2Opt [117], EvoTF [36], LEO [118], RIBBO [37], NAP [157]. In particular, RIBBO [37] and EvoTF [36] show different methodologies. They apply supervised imitation learning to meta-train their policies resemble a teacher BBO algorithm. For instance, RIBBO uses a GPT architecture to imitate optimization trajectory from diverse existing BBO algorithms. It tokenizes each of the collected optimization trajectories into a target token sequence $\{R^1, X^1, Y^1, ..., R^T, X^t, Y^T\}$, where R^t, X^t and Y^t are the regret-based explicit supervising objective, population positions and objective values respectively. RIBBO trains the GPT to imitate these trajectories through behaviour cloning, which leads to generalized optimization behaviour.

D. MetaBBO-ICL

The concept of In-Context Learning (ICL) [144] is a popular prompting paradigm in LLMs researches, which prompts LLMs by a well-structured text collection: a *task description*, several *in-context examples* and a concrete *task instruction*. Such prompt form ensures the robust reasoning ability of LLMs based on well-described context information. Note that there is no gradient descent or parameter update for the neural network language model. Instead, prompting LLMs is to let LLMs "learn the context" and then dictate desired answer. There are three ways of practices in existing MetaBBO works. Note that MetaBBO-ICL is closely related to two meta tasks: solution manipulation (SM) and Algorithm Generation (AG). The main difference in-between these works is how they construct effective in-context prompts.

For SM task, OPRO [112] firstly proposes optimization by prompting LLMs iteratively. For each iteration, the *task description* is customized for any specific problem types which include the objective functions and constraints described in language. The *in-context examples* include the optimization trajectories before this iteration. The *task instruction* indicates what exactly LLMs should do, i.e., to find a solution better than the best ever found. However such paradigm challenges the expertise of the general LLMs for optimization, which is less developed during their pre-training [35]. To address this, latest studies innovatively propose prompting LLMs to behave like specific EAs [113], that is, instructing LLMs to perform mutation, crossover operations and elitism strategy on the given *in-context examples*. For AG task, a group of programs of optimizers are maintained and the LLMs are also used as evolution operators to evolve these programs. A representative work is EoH [46]. In EoH, the *task description* include two parts: the optimization problem formulation, the concrete algorithm design task that asks LLMs to first describe the newly suggested heuristic and then implement it using Python. The *in-context examples* are the programs suggested previously. A special design of EoH is the *task instruction*, which provides five evolution instructions with different extents of code refinement. Other related works include AEL [121], LLaMEA [123], LLMOPT [124].

E. Summary

In this section, we have introduced four main learning paradigms widely adopted in existing MetaBBO works to meta-train their meta-level policy. Note that although these learning paradigms can all be used to develop MetaBBO methods and their applications, there are significant technical differences among them. By closely examining the experimental results presented in the original papers and comparing their specific implementations, we outline several key factors to highlight the characteristics of each learning paradigm. We present a comparison using six criteria in Table II, include: 1) Development difficulty, measured by the complexity of code implementation. 2) Expertise dependency, indicating how

TABLE II
COMPARISON OF FOUR LEARNING PARADIGMS, INCLUDING THE DEVELOPMENT PROPERTIES, TECHNICAL PERFORMANCES AND RESEARCH INTERESTS.

	Development difficulty ↓	Expertise dependency ↓	Data utilization ↑	Training efficiency ↑	Inference efficiency ↑	Optimization Performance ↑	Publications
Reinforcement Learning	★★	★★	★★★	★★★	★★★	★★★	[28]–[34], [48], [49], [70], [71] [73]–[111], [120], [122], [126]
Neuroevolution	★★★	★★★	★	★	★★★	★★	[41]–[43]
Supervised Learning	★★	★★	★★	★★	★★★	★★★	[36]–[40], [63]–[66], [68], [72], [117]
In-Context Learning	★	★	★	★	★	★	[35], [45]–[47], [69], [112]–[116] [118], [119], [121], [123]–[125]

much degree of expert-level knowledge is required. 3) Data utilization efficiency, measured by the theoretical performance gain that can be obtained by the same batch of data. 4) Training efficiency, measured by the training wall time. 5) Inference efficiency, measured by the wall time required for solving an optimization problem. 6) Optimization performance, which shows the empirical optimization results.

To summarize, each learning paradigm has its own advantages and drawbacks. MetaBBO-ICL, powered by LLMs, requires the least efforts in development and design. MetaBBO-RL, despite necessitating specific skills in constructing RL systems and crafting the MDP for the meta-level task, excels in data utilization due to its ability to improve the policy in an unsupervised setting. Besides, many MetaBBO-RL methods apply simple yet powerful tabular Q-learning, which enjoy high training/inferring efficiency with low computational overhead. As for the final performance, we empirically observe that MetaBBO-RL and MetaBBO-SL generally achieve superior optimization results. Concerning the research focus on the four learning paradigms, we observed that MetaBBO-RL consistently attracts interest from researchers, whereas MetaBBO-ICL is rapidly gaining traction with the advent of LLMs.

V. EMPIRICAL EVALUATION

A. Development in Benchmarks

For benchmarking BBO optimizers, many well-known testsuites have been extensively studied and developed [158], [159]. With the ongoing development of BBO, the corresponding benchmarks aim to 1) propose more diverse benchmark problems in synthetic [67], [160]–[165] and realistic [143], [166], [167] scenarios; and 2) automate the benchmarking process through a software platform [155], [168]. These traditional BBO benchmarks can serve as evaluation tools for MetaBBO methods. However, compared with traditional EC algorithms, the system structure of MetaBBO is more intricate. Its bi-level learning paradigm involves a meta-level policy, a low-level optimizer, the training/testing logic of the entire system, and the interfaces between the meta and lower levels. This complexity creates a gap between the conventional BBO benchmarks and MetaBBO methods. To address this compatibility issue, a recent work termed MetaBox [26] proposes the first benchmark platform specifically for developing and evaluating MetaBBO methods. It provides three different problem collections (Synthetic-10D, Noisy-Synthetic-10D, Protein-Docking-12D) , along with two different train-test split modes (easy and difficult), which benefits MetaBBO's training under different problem distributions and difficulties. In the next subsection, we provide a proof-of-principle evaluation of several representative MetaBBO methods using MetaBox.

B. Proof-of-Principle Evaluation by MetaBox

In this section, we use MetaBox [26] to evaluate the performance of 3 traditional EC algorithms and 9 representative MetaBBO methods. For traditional EC algorithms, we include JADE [19], GLPSO [21] and CMA-ES [22]. Further, we include the MetaBBO methods covering all four meta-tasks: RL-DAS [34] for algorithm selection; DE-DDQN [28], LDE [31], RLEPSO [82], LES [42], and GLEET [32] for algorithm configuration; SYMBOL [122] for algorithm generation; and RNN-OI [38] and GLHF [40] for solution manipulation. All baselines are trained for 1.5×10^6 learning steps and tested over 51 independent runs to ensure fairness. Other settings follow their original papers.

1) ***Comparison on the Optimization Performance****:* The AEI score in MetaBox [26] evaluates the overall optimization performance of a MetaBBO method by aggregating three key metrics: final optimization results, FEs consumed, and runtime complexity, using an exponential average, lager is better.

The left side of Fig. 7 presents the final optimization accuracy of all baselines on Synthetic BBOB (top) and Realistic Protein Docking (bottom) testsuites, while the right side presents their respective AEI scores. The results show that: 1) When considering only the final accuracy, MetaBBO methods such as RL-DAS, LDE, and GLEET achieve comparable or even superior performance to traditional BBO optimizers, while some other MetaBBO methods still perform inferiorly compared to traditional BBO methods. This indicates that while MetaBBO methods show potential, as an emerging topic, there is still significant room for improvement. 2) Different evaluation metrics yield different conclusions regarding performance. When considering both optimization performance and computational overhead, traditional BBO optimizers such as CMA-ES achieve a significantly better trade-off, as shown on the right side of Fig. 7. This highlights a potential limitation of MetaBBO methods: they typically involve additional computation during the meta-level decision-making process. 3) Comparing the performance on the Synthetic BBOB and Protein Docking test suites, we observe that the performance gap between MetaBBO methods and traditional BBO optimizers narrows on both evaluation metrics as the problem

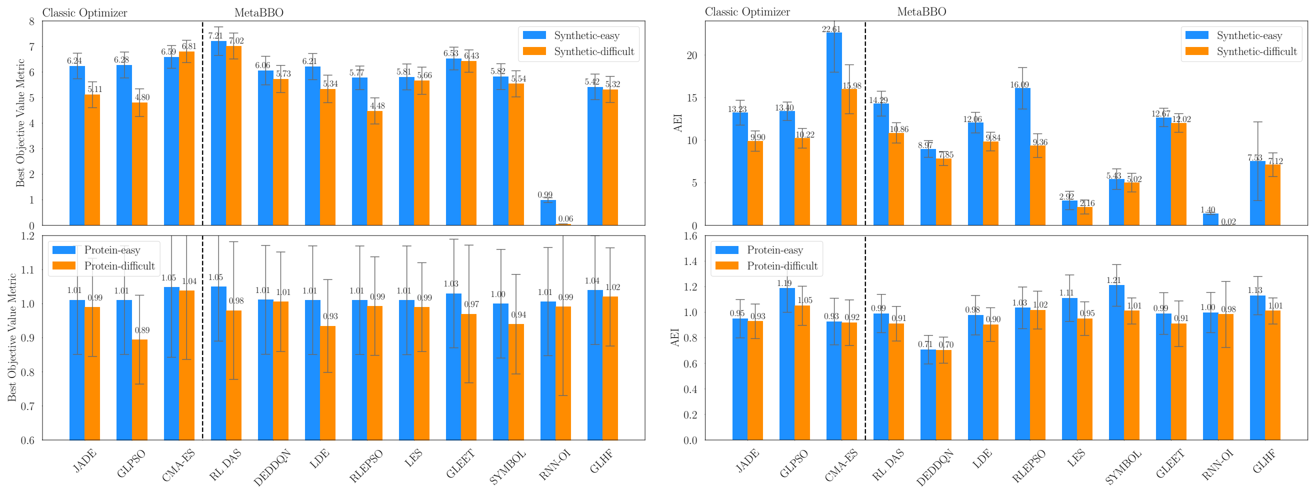

Fig. 7. Performance comparisons. **Top Left:** Best objective values on synthetic testsuites. **Top Right:** AEI scores on synthetic testsuites. **Bottom Left:** Best objective values on protein docking testsuites. **Bottom Right:** AEI scores on protein docking testsuites.

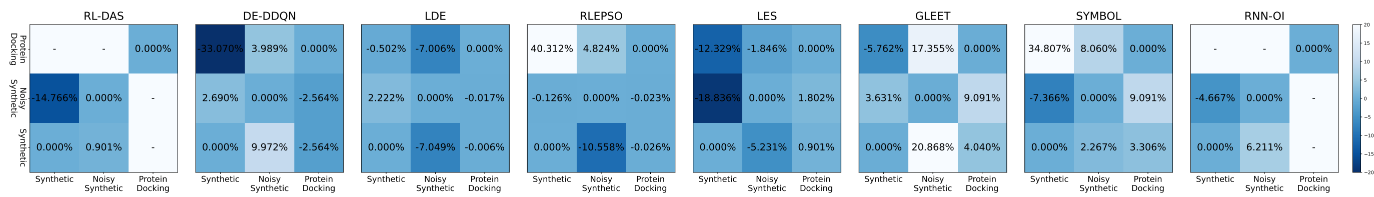

Fig. 8. MGD scores of baselines. The value at i-th row and j-th column is the MGD(i, j), with smaller value indicating better performance.

type shifts from the relatively simpler synthetic set to the more challenging realistic protein docking set. This suggests that MetaBBO is promising for solving complex optimization problems. 4) MetaBBO-RL methods (including RL-DAS, DE-DDQN, LDE, RLEPSO, SYMBOL) outperform MetaBBO-NE methods (LES) and MetaBBO-SL methods (RNN-OI). This observation highlights an important future direction for the MetaBBO domain: analyzing the theoretical performance bounds of different MetaBBO methods.

2) ***Comparison on the Learning Capabilities:*** As a learning based paradigm, MetaBBO should also be evaluated using metrics that reflect its learning effectiveness. Next, we evaluate the MetaBBO methods using the Meta Generalization Decay (MGD) and Meta Transfer Efficiency (MTE) from MetaBox.

a) Meta Generalization Decay: MGD measures the generalization performance of a MetaBBO method for unseen tasks. Concretely, MetaBox trains two models for the MetaBBO method on two source suites (A and B) and test them on the target suit B. We record the AEI scores of these two models on the target suit as AEI_A and AEI_B respectively. The MGD(A, B) is computed as

$$\text{MGD}(A, B) = 100 \times (1 - \frac{AEI_A}{AEI_B})\%, \tag{11}$$

A smaller MGD score indicates that the method generalizes well from A to B.

Fig. 8 shows the MGD plot of MetaBBO baselines, with GLHF omitted since it does not provide training codes. The '-' indicates that the model fails to generalize to target testsuit. We can observe that: 1) RL-DAS and RNN-OI cannot be generalized from Synthetic-10D to Protein Docking-12D due to their optimization state features are dimension-dependent, highlighting the importance of optimization state design. 2) MetaBBO-NE methods (LES) achieves more robust generalization than MetaBBO-RL (RL-DAS, DE-DDQN, LDE, RLEPSO, GLEET, SYMBOL) and MetaBBO-SL (RNN-OI) baselines, possible revealing the learning effectiveness advantage of neuroevolution paradigm due to its global learning ability. However, neuroevolution is exponentially resource-consuming as the neuron counts scale, implying a tradeoff between effectiveness and efficiency. 3) Larger models (e.g., GLEET with 3 Transformer layers) underperform smaller ones (e.g., DE-DDQN with a single MLP) in the generalization evaluation, even though they outperform within the training distribution. Since the generalization performance is closely tied to the model capacity and the data scale, further investi-

TABLE III
MTE SCORES OF THE METABBO METHODS IN THE TRANSFER FROM SOURCE TESTSUITES TO TARGET TESTSUITES.

Source Testsuites	Synthetic		Noisy-Synthetic		Protein-Docking	
Target Testsuites	Noisy Synthetic	Protein Docking	Synthetic	Protein Docking	Synthetic	Noisy Synthetic
RL-DAS	fail	-	1	-	-	-
DE-DDQN	-9.5	1	-3.2	fail	-1.41	-9.5
LDE	-0.05	-0.17	0.19	-0.17	1	-0.05
RLEPSO	fail	-4.25	-0.39	-4.25	1	fail
LES	-1.65	-0.05	fail	-0.05	fail	1
GLEET	1	-0.05	fail	-0.05	fail	-0.39
SYMBOL	-0.16	fail	0.97	-2.17	-0.05	fail
RNN-OI	1	-	0.01	-	-	-

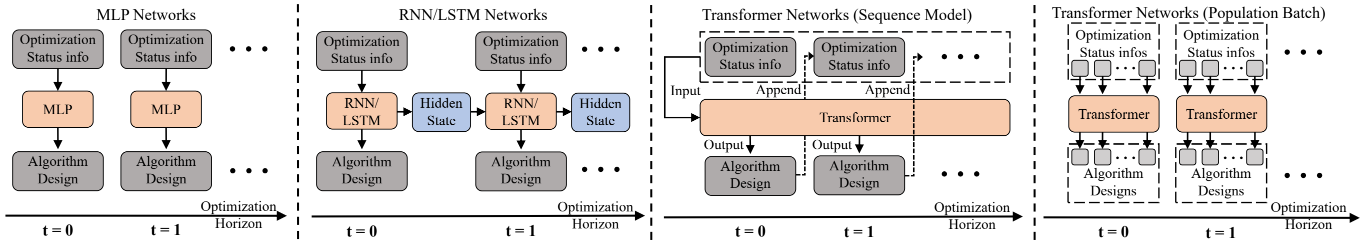

Fig. 9. The workflow of different neural networks used in existing MetaBBO works: MLP, RNN/LSTM and Transformer architectures.

gation into the scaling laws in MetaBBO is highly anticipated.

b) Meta Transfer Efficiency: MTE score measures the transfer learning capability. For a MetaBBO method, its MTE from a problem set A to B is computed as:

$$\mathrm{MTE}(A, B) = 100 \times (1 - \frac{T_{\text{finetune}}}{T_{\text{scratch}}})\%, \tag{12}$$

where T_{scratch} is the learning steps used to attain best performance when training on B. T_{finetune} is the learning steps used to fine-tune a model trained on A to attain the same performance level. A larger MTE score indicates that the knowledge learned in A can be easily transferred to solve B. Table III presents the MTE scores of all baselines under each pair of source-target problem collections, where 'fail' indicates the baseline can not be fine-tuned to achieve similar performance level on the target problems. Results show that: 1) While many baselines highlight their transfer learning ability under some cases (e.g., GLEET: Synthetic to Noisy-Synthetic), they show transfer limitations in other cases, suggesting room for improvement. 2) The overall transferring performances across all baselines and problem collections are relatively noisy, making it difficult to determine whether some transfer failures stem from the algorithm designs or the diversity of the problem collections. This opens up a research opportunity to explore the relationship between problem diversity and generalization, as well as how to construct "good" training set for MetaBBO.

VI. Key Design Strategies

According to the proof-of-principle evaluation we conduct in Section V, we summarize and analyse key design strategies of existing MetaBBO works related to the effectiveness and generalization: neural network design (Section VI-A), optimization state design (Section VI-B), meta-objective design (Section VI-D) and training distribution (Section VI-C).

A. Neural Network Design

We discuss the design motivation of the neural networks π_θ in existing MetaBBO works. Four common neural network designs are frequently adopted: 1) MLP structure, 2) RNN and LSTM structures, 3) temporal dependency Transformer structure, and 4) spatial dependency Transformer structure, which are illustrated in Fig. 9 from left to right. The basic MLP structure (leftmost of Fig. 9) is widely adopted in existing works due to: its relatively superior representation ability to non-parametric model; and its simplicity and corresponding efficient training/inferring. However, MLP structure falls short in analysing the temporal properties and data batch properties within the low-level BBO process of a MetaBBO paradigm. We next introduce novel designs that address this bottleneck.

*1) **Temporal Dependency Architectures**:* The low-level BBO process involves the iterative optimization for T steps (generations). If the meta-level policy is a basic MLP structure, it probably shows inability of leveraging the historical information along the optimization trajectory. An intuitive way is to introduce novel neural network structures that support temporal sequence modelling. To this end, works such as RNN-OI [38], RNN-Opt [39] and LTO-POMDP [41] introduce RNN and LSTM [131] which integrate the historical optimization information in hidden representations and combine them with current optimization states (the second part of Fig. 9). While these works improve the learning effectiveness with historical information, training on long horizons (often involves hundreds of generations) by RNN/LSTM can be challenging due to inherent gradient vanishing or explosion, leading to poor generalization. Subsequent works such as MELBA [111], RIBBO [37] and EvoTF [36] leverage Transformer structures for the long sequence modelling ability. The shared workflow in these works is illustrated in the third part of Fig. 9, where a trajectory of historical optimization states is processed by the Transformer to dictate the next step algorithm design.

*2) **Spatial Dependency Architectures**:* Despite the temporal properties, a key characteristic of EC is the population-based searching. Recent MetaBBO works dictate customized algorithm designs for each individual solution in the population, to achieve maximum flexibility for the low-level optimization. To this end, as illustrated in rightmost of Fig. 9, works such as LGA [43], LES [42], B2Opt [117], GLEET [32], RLEMMO [48], GLHF [40] construct optimization state feature as a collection of individual optimization states and then leverage Transformer's attention computation mechanism to enhance the information sharing across the solution population. For instance, GLEET [32] proposes a unique Transformer-style network, which includes a fully informed encoder and a exploration-exploitation decoder. The fully informed encoder promotes the information sharing by applying self-attention on the optimization state features of all individual solutions. It additionally construct informative features about the exploration and exploitation status of current optimization progress. At last, the exploration-exploitation decoder decodes the hyper-parameter value for updating each individual in the next optimization step through using the obtained exploration-exploitation feature to query the fully informed population feature. Another work, RLEMMO [48], encodes the optimization state for each individual with its historical optimization information and local perception from its neighbours. Then the Transformer-style policy is used to decide the mutation strategies for each individual. Such a flexible algorithm design enhance the exploration-exploitation tradeoff hence results in robust performance when solving challenging multi-modal problems.

B. State Feature Design

A key component to ensure the generalization ability of MetaBBO across diverse optimization problems is the optimization state feature extraction function sf(·). We summarize three key parts an informative optimization state feature should cover: a) *problem identification feature* indicating the landscape properties of target problem. b) *population profiling features* indicating the local distributional properties of the solution population in the low-level BBO process. c) *optimization progress features* indicating that whether the solution population improves at each optimization step. We next introduce common practice for preparing these features.

*1) **Problem Identification Features**:* To identify the target optimization problem, an effective analysis framework is Exploratory Landscape Analysis (ELA) [169]. ELA comprises six groups of metrics such as local search, skewness of objective space and approximated curvature in (second) first-order as a comprehensive abstract of basic landscape properties for a target problem. To compute ELA features, a relatively large amount of points are sampled from the decision space first, then the each feature groups is computed. For instance, when computing *meta-model* features, a group of linear/quadratic models are used to fit the sampled points and their objective values, the best-fitting parameters of these models have been demonstrated as serve as useful identification feature for distinguishing different problems.

*2) **Population Profiling Features**:* Within an optimization problem, there are different landscapes where the current solution population might falls in. MetaBBO aims to dynamically dictate customized algorithm designs to help the low-level optimizer adapt for these diverse population positions. To identify the distributional properties of the solution population, Fitness Landscape Analysis (FLA [142]) is widely adopted which provide different indicators: Fitness Distance Correlation [170], Ruggedness of Information Entropy [171], Auto-Correlation Function [172], Dispersion [173], Negative Slope Coefficient [174] and Average Neutral Ratio [175], some of which measure the local landscape properties according to where the population locates at the problem's fitness space. Concretely, to compute FLA features, a random walking strategy is used to sample moderate points in the neighbourhood of a given individual, which is then used to compute the mentioned statistical indicators subsequently. Population profiling features serve as a complement of the problem identification features to provide the meta-level policy a more accurate optimization state.

*3) **Optimization Progress Features**:* Optimization progress features often serve as a complement to ELA and FLA features to provide the meta-level policy more useful decision information. Optimization progress features focus more on "low-level" properties such as the consumed FEs so far, the distribution/average of the current population and the corresponding objective values, etc. These features closely trace the improvement and convergence of the population. Interestingly, recent MetaBBO works such as DE-DDQN [28], RLEPSO [82] and GLEET [32] found that sorely using optimization progress features might be sufficient for MetaBBO method to learn generalizable meta-level policy with better optimization performance. A key reason behind is that computing ELA/FLA features on the one hand consumes the FEs reserved for the low-level BBO process, and reduces the learning steps of the meta-level policy on the other hand, degrading both the learning effectiveness and final optimization performance.

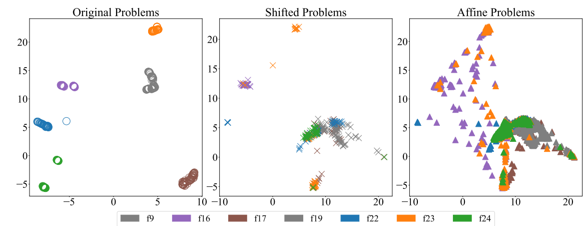

Fig. 10. The projected 2D ELA distributions of **Left:** the original BBOB problems; **Middle:** the BBOB problems with shifted optimum; **Right:** the BBOB problems generated by MA-BBOB.

C. Training Distribution Design

The training problem set plays a key role for meta learning a generalizable meta-level policy. Its diversity, in particular, should be promised. Initial works such as RNN-OI [38] is trained on a few instances of CoCo-BBOB testsuite. Its performance in Fig. 7 demonstrates that narrow training set leads to unsatisfactory generalization. To increase the diversity of the training problem set, two methodologies are primarily adopted in existing MetaBBO approaches.

*1) **Augmenting Existing Benchmarks**:* The standard BBO benchmarks include CoCo-BBOB testsuites [159], [176], CEC BBOB-Competition testsuites [158], [177], etc. There are approximately $20 \sim 30$ synthetic functions in these testsuites, with different problem properties such as multimodality, separability, non-convexity, etc. Almost all of existing MetaBBO works apply mathematical transformation on these synthetic functions to augment the training problem set. Concretely, given a D-dimensional function instance $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}$, it can be transformed to a new function instance $f'(x) = f(M^T(x-o))$, where $M \in \mathbb{R}^{D\times D}$ is a rotation matrix rotating the decision space and $o \in \mathbb{R}^D$ is an offset to the optimal. In recent works such as GLEET [32] and RL-DAS [34], they apply random combinations of the shift and rotation transformation on CEC2021 testsuite [158] to obtain thousands of synthetic problem instances, significantly improving the generalization performance of the learned meta-level policy.

*2) **Constructing New Benchmarks**:* Note that although augmenting the existing standard synthetic functions by shift and rotation matrix helps the generic training, the diversity of the problem set still has improvement room. To validate this, we illustrate 2D projection of the ELA distribution of some CoCo-BBOB problem instances and the corresponding transformed instances in the left and middle of Fig. 10. The results show that transformed instance indeed introduce certain diversity (e.g., shifted function instances of $f23$ scatter to different area). However, the diversity improvement is quite limited. An alternative way to increase the diversity is to generating novel benchmarks. A recent work termed MA-BBOB [165] indicates that affine combination of existing standard synthetic functions could create diverse instance. We examine this by investigating the same 2D projection of the ELA distribution of function instances created by MA-BBOB in the right of Fig. 10. The resulted instances scatter toward wider area of the feature space.

D. Meta-Objective Design

The meta-objective $J(\theta)$ in MetaBBO indicates the expected accumulated performance gain perf$(\cdot)$ can be obtained over the problems in the training problem set. In existing MetaBBO works, perf$(\cdot)$ is closely related to the objective values of the solution population to guide the meta-level policy to pursue better optimization performance. An intuitive way is using indicator function: if the performance improvement is observed between two optimization steps of the low-level BBO process, a positive reward is given. Otherwise, a negative or zero punishment is given. This idea is widely adopted in initial MetaBBO works such as DE-DDQN [28], QLPSO [30], and MARLwCMA [79].

1) ***Scale Normalization:*** However, the above simple mechanism might becomes problematic when one wants to measure the performance improvement accurately hence enhance the flexibility of the learned policy. To this end, an alternative way is to compute perf$(\cdot)$ as the objective value descent $\Delta f^t = f^{*,t-1} - f^{*,t}$ directly, where $f^{*,t}$ is the objective value of the best solution found so far. Naively using this absolute objective value descent might causes unstable learning due to the different objective value scales of different optimization problems. To address this, recent MetaBBO works apply min-max normalization operations on the objective descent:

$$\text{perf}(\cdot, t) = \frac{f^{*,t-1} - f^{*,t}}{f^{*,1} - f^{*}} \tag{13}$$

where $f^{*,1}$ denote the objective value of the best solution in the initialized population, f^* denotes the optimum of the f. In practice, f^* is unknown since f is black-box, which can be approximated by running an efficient BBO algorithm to optimize f for multiple runs in advance.

2) ***Sparse Reward Handling:*** Besides, the difficulty of the low-level BBO process increases with time: at the beginning, the objective value might achieves rapid descent, while the objective value descent approaches 0 in the later stage due to the convergence. This problem is often identified as sparse reward issue in learning system. This setting might misleads the learning of the meta-level policy: the meta-level policy leans to dictate sub-optimal algorithm designs that only focus on the early stage of the optimization process. To further address this, recent works propose adaptive performance metric that adds a scale factor $\lambda(t)$ to Eq. (13) to magnify the performance improvement in the later optimization stage:

$$\text{perf}(\cdot, t) = \lambda(t) \times \frac{f^{*,t-1} - f^{*,t}}{f^{*,1} - f^{*}} \tag{14}$$

where the scale factor $\lambda(t)$ is a incremental function of the optimization step t. For instance, in MADAC [87], $\lambda(t)$ is $2f^* - f^{*,t-1} - f^{*,t}$. Generally speaking, better learning effectiveness is observed in MetaBBO works which apply accurate performance improvement as perf$(\cdot)$, e.g., as shown in ablation studies conducted by RL-DAS [34], GLEET [32], RIBBO [37], and GLHF [40].

VII. Vision for the Field

A. Generalization Toward Task Mixtures

A promising direction is the generalization toward a mixture of tasks. While MetaBBO works have explored various aspects of model generalization, the evaluation and analysis we provide in previous sections outline potential improvement through advanced learning techniques, e.g., transfer learning [178] and multitask learning [179].

First, existing works often focus on algorithm design for specific optimizers. For instance, methods like LDE [31] and GLHF [40] are designed for algorithm configuration or imitation tasks, but primarily with basic DE. This narrow focus might lead to uncertain performance when applying these methods to other optimizers. A more effective approach would be to create a higher-level framework that defines MetaBBO tasks across multiple optimizers, establishing a multitask design space. Developing a universal modularization paradigm for various optimizers could allow training a meta-level policy that generalizes well across tasks.

Additionally, existing works focus exclusively on a single specific problem type. Separate policies are trained for each task type, leading to increased complexity. This outlines an opportunity to develop a unified agent capable of engaging in automatic algorithm design that adapts to various problem types. This method not only streamlines the optimization process but also aligns more closely with real-world scenarios, where practitioners frequently encounter a diverse array of problems. To overcome the limitations of existing methods, a universal problem representation system is essential to bolster the generalization across diverse problem domains.

B. Fully End-to-End Autonomy

The main motivation behind MetaBBO is to reduce the labor-intensive need for expert consultation by offering a general optimization framework. However, existing MetaBBO approaches still introduce design elements that rely on expert knowledge to enhance performance. This reliance typically involves: 1) the low-level optimization state s, often handcrafted as a feature vector to represent problem properties or optimization progress; and 2) the meta-objective, which is mostly developer-defined, introducing subjectivity. While initial efforts have been made to automate feature extraction using neural networks [180], [181] and employ model-based RL to learn the meta-objective objectively [98], further systematic studies are needed. Besides, MetaBBO focuses on designing algorithms in isolation, assuming the optimization problem is predefined and ready for evaluation. In reality, the initial step often involves formulating the problem, either through manual model construction or data-driven methods. This disconnect reveals a major gap in the optimization process. A more integrated approach would involve learning of objective formulation, automatic feature extraction, and then customized algorithm design. Developing a cohesive pipeline for these steps offers a promising direction for advancing optimization and improving problem-solving in practical applications.

C. Smarter Integration of LLMs

As discussed in previous sections, the emergence of LLMs has significantly influenced various fields, including optimization. Over the past two years, there has been rapid growth in research exploring LLM-assisted EC. For those interested in these works, please refer to our introduction in Section III and Section IV, or the original papers and a recent survey [52]. The diverse paradigms of LLM-assisted EC align well with two algorithm design tasks in MetaBBO: using LLMs for algorithm generation [46] and solution manipulation [112]. However, the implicit in-context learning used in existing methods does not involve learning a specific model for the algorithm design task. Instead, they rely on prompting a pre-trained LLM with carefully engineered contextual content iteratively. The rationality behind is that general LLMs present powerful conditional reasoning capabilities, allowing them to provide insights, ideas and even code implementations. Yet, recent works, such as LLaMoCo [35], have shown that the optimization-specific knowledge embedded within LLMs is often insufficient to provide robust performance in complex optimization scenarios.

This suggests two promising directions: First is the automated MetaBBO workflow search, leveraging LLMs for designing MetaBBO workflow through code generation and function search. Designing a learning system like MetaBBO is inherently challenging, as it requires considerable expertise. By providing LLMs with foundational principles of MetaBBO, the chain of thought within the models may uncover novel paradigms. Second, enhancing the semantic understanding of LLMs regrading optimization processes, terminologies, programming logics, problem descriptions would significantly elevate their expertise. To achieve this, an interesting direction is to develop symbolic language tailored to optimization domain, establishing a comprehensive grammar system and accumulating sufficient use cases to train a foundation model specifically for optimization.

VIII. Conclusion

In this survey, we provide a comprehensive review of recent advancements in MetaBBO. As a novel research avenue within the BBO and EC communities, MetaBBO offers a promising paradigm for automated algorithm design. Through a bi-level data-driven learning framework, MetaBBO is capable of meta-learning effective neural network-based meta-level policies. These policies assist with algorithm selection and algorithm configuration for a given low-level optimizer, as well as to imitate or generate optimizers with certain flexibility.

Our review begins with the mathematical definition of MetaBBO, clarifying its bi-level control workflow. Next, we systematically explore four main algorithm design tasks where MetaBBO excels: AS, AC, SM, and AG. Following the discussion of these tasks, we examine four methodologies of training MetaBBO: SL, RL, NE, and ICL. We hope these two parts will provide readers with a clear roadmap to quickly locate their interested MetaBBO methods.

Further, to provide a practical guide in this field, we benchmark representative MetaBBO methods on a customized platform. The results reveal that, while MetaBBO techniques outperform traditional BBO algorithms in certain optimization scenarios, there remains a significant potential for improvement in terms of computational complexity, generalization capability, and transfer learning efficiency in the current research. Furthermore, we provide in-depth analysis on some core designs of MetaBBO: the neural network architecture, optimization state feature extraction mechanism, training problem distribution, and meta-objective design. These insights offer practical guidelines for researchers and practitioners aiming to develop more effective and efficient MetaBBO methods. At last, we propose several interesting and open-ended future directions for MetaBBO research, encouraging further exploration and innovation in this promising field.

References

[1] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *TEC*, vol. 9, no. 3, pp. 303–317, 2005.

[2] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective," *TC*, vol. 50, no. 8, pp. 3668–3681, 2019.

[3] M. H. Yar, V. Rahmati, and H. R. D. Oskouei, "A survey on evolutionary computation: Methods and their applications in engineering," *Mod. Appl. Sci*, vol. 10, no. 11, p. 131139, 2016.

[4] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications," *TEC*, vol. 17, no. 3, pp. 321–344, 2012.

[5] A. M. Gopakumar, P. V. Balachandran, D. Xue, J. E. Gubernatis, and T. Lookman, "Multi-objective optimization for materials discovery via adaptive design," *Sci. Rep.*, vol. 8, no. 1, p. 3738, 2018.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[7] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint*, 2017.

[8] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint*, 2016.

[9] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint*, 2014.

[10] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

[11] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *ICGTSPICC*, 2016, pp. 261–265.

[12] J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization," Tech. Rep., 2013.

[13] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, S. Tiwari *et al.*, "Multiobjective optimization test instances for the CEC 2009 special session and competition," Tech. Rep., 2008.

[14] S. Das, S. Maity, B.-Y. Qu, and P. N. Suganthan, "Real-parameter evolutionary multimodal optimization—a survey of the state-of-the-art," *Swarm Evol. Comput.*, vol. 1, no. 2, pp. 71–88, 2011.

[15] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization," Tech. Rep., 2007.

[16] Q. Xu, N. Wang, L. Wang, W. Li, and Q. Sun, "Multi-task optimization and multi-task evolutionary computation in the past five years: A brief review," *Mathematics*, vol. 9, no. 8, p. 864, 2021.

[17] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *TEC*, vol. 1, no. 1, pp. 67–82, 1997.

[18] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *TSMC*, vol. 24, no. 4, pp. 656–667, 1994.

[19] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *TEC*, vol. 13, no. 5, pp. 945–958, 2009.

[20] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *TSMC*, vol. 39, no. 6, pp. 1362–1381, 2009.

[21] Y.-J. Gong, J.-J. Li, Y. Zhou, Y. Li, H. S.-H. Chung, Y.-H. Shi, and J. Zhang, "Genetic learning particle swarm optimization," *TC*, vol. 46, no. 10, pp. 2277–2290, 2015.

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

[23] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *CEC*, 2013, pp. 71–78.

[24] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *CEC*, 2014, pp. 1658–1665.

[25] V. Stanovov, S. Akhmedova, and E. Semenkin, "Nl-shade-lbc algorithm with linear parameter adaptation bias change for CEC 2022 numerical optimization," in *CEC*, 2022, pp. 01–08.

[26] Z. Ma, H. Guo, J. Chen, Z. Li, G. Peng, Y.-J. Gong, Y. Ma, and Z. Cao, "Metabox: a benchmark platform for meta-black-box optimization with reinforcement learning," *NeurIPS*, 2024.

[27] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[28] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, "Deep reinforcement learning based parameter control in differential evolution," in *GECCO*, 2019, pp. 709–717.

[29] Z. Tan, Y. Tang, K. Li, H. Huang, and S. Luo, "Differential evolution with hybrid parameters and mutation strategies based on reinforcement learning," *Swarm Evol. Comput.*, vol. 75, p. 101194, 2022.

[30] Y. Xu and D. Pi, "A reinforcement learning-based communication topology in particle swarm optimization," *Neural Comput. Appl.*, vol. 32, pp. 10 007–10 032, 2020.

[31] J. Sun, X. Liu, T. Bäck, and Z. Xu, "Learning adaptive differential evolution algorithm from optimization experiences by policy gradient," *TEC*, vol. 25, no. 4, pp. 666–680, 2021.

[32] Z. Ma, J. Chen, H. Guo, Y. Ma, and Y.-J. Gong, "Auto-configuring exploration-exploitation tradeoff in evolutionary computation via deep reinforcement learning," in *GECCO*, 2024, pp. 1497–1505.

[33] Z. Tan and K. Li, "Differential evolution with mixed mutation strategy based on deep reinforcement learning," *Appl. Soft Comput.*, vol. 111, p. 107678, 2021.

[34] H. Guo, Y. Ma, Z. Ma, J. Chen, X. Zhang, Z. Cao, J. Zhang, and Y.-J. Gong, "Deep reinforcement learning for dynamic algorithm selection: A proof-of-principle study on differential evolution," *TSMC*, vol. 54, no. 7, pp. 4247–4259, 2024.

[35] Z. Ma, H. Guo, J. Chen, G. Peng, Z. Cao, Y. Ma, and Y.-J. Gong, "LLaMoCo: Instruction tuning of large language models for optimization code generation," *arXiv preprint*, 2024.

[36] R. Lange, Y. Tian, and Y. Tang, "Evolution transformer: In-context evolutionary optimization," in *GECCO*, 2024, pp. 575–578.

[37] L. Song, C. Gao, K. Xue, C. Wu, D. Li, J. Hao, Z. Zhang, and C. Qian, "Reinforced in-context black-box optimization," *arXiv preprint*, 2024.

[38] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas, "Learning to learn without gradient descent by gradient descent," in *ICML*, 2017.

[39] V. TV, P. Malhotra, J. Narwariya, L. Vig, and G. Shroff, "Meta-learning for black-box optimization," in *ECML PKDD*, 2019.

[40] X. Li, K. Wu, Y. B. Li, X. Zhang, H. Wang, and J. Liu, "GLHF: General learned evolutionary algorithm via hyper functions," *arXiv preprint*, 2024.

[41] H. S. Gomes, B. Léger, and C. Gagné, "Meta learning black-box population-based optimizers," *arXiv preprint*, 2021.

[42] R. Lange, T. Schaul, Y. Chen, T. Zahavy, V. Dalibard, C. Lu, S. Singh, and S. Flennerhag, "Discovering evolution strategies via meta-black-box optimization," in *GECCO*, 2023, pp. 29–30.

[43] R. Lange, T. Schaul, Y. Chen, C. Lu, T. Zahavy, V. Dalibard, and S. Flennerhag, "Discovering attention-based genetic algorithms via meta-black-box optimization," in *GECCO*, 2023, pp. 929–937.

[44] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers," in *ICLR*, 2024.

[45] R. Lange, Y. Tian, and Y. Tang, "Large language models as evolution strategies," in *GECCO*, 2024, pp. 579–582.

[46] F. Liu, T. Xialiang, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *ICML*, 2024.

[47] A. AhmadiTeshnizi, W. Gao, and M. Udell, "OptiMUS: Scalable optimization modeling with (MI) LP solvers and large language models," in *ICML*, 2024.

[48] H. Lian, Z. Ma, H. Guo, T. Huang, and Y.-J. Gong, "Rlemmo: Evolutionary multimodal optimization assisted by deep reinforcement learning," in *GECCO*, 2024, pp. 683–693.

[49] S.-H. Wu, Y. Huang, X. Wu, L. Feng, Z.-H. Zhan, and K. C. Tan, "Learning to transfer for evolutionary multitasking," *arXiv preprint*, 2024.

[50] Y. Huang, X. Lv, S. Wu, J. Wu, L. Feng, and K. C. Tan, "Advancing automated knowledge transfer in evolutionary multitasking via large language models," *arXiv preprint*, 2024.

[51] Q. Zhao, Q. Duan, B. Yan, S. Cheng, and Y. Shi, "Automated design of metaheuristic algorithms: A survey," *TMLR*, 2024.

[52] X. Wu, S.-h. Wu, J. Wu, L. Feng, and K. C. Tan, "Evolutionary computation in the era of large language model: Survey and roadmap," *arXiv preprint*, 2024.

[53] T. Stützle and M. López-Ibáñez, "Automated design of metaheuristic algorithms," *Handbook of metaheuristics*, pp. 541–579, 2019.

[54] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm Evol. Comput.*, vol. 44, pp. 228–246, 2019.

[55] M. Chernigovskaya, A. Kharitonov, and K. Turowski, "A recent publications survey on reinforcement learning for selecting parameters of meta-heuristic and machine learning algorithms." in *CLOSER*, 2023, pp. 236–243.

[56] Y. Song, Y. Wu, Y. Guo, R. Yan, P. N. Suganthan, Y. Zhang, W. Pedrycz, S. Das, R. Mallipeddi, O. S. Ajani *et al.*, "Reinforcement learning-assisted evolutionary algorithm: A survey and research opportunities," *Swarm Evol. Comput.*, vol. 86, p. 101517, 2024.

[57] P. Li, J. Hao, H. Tang, X. Fu, Y. Zhen, and K. Tang, "Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms," *TEC*, 2024.

[58] R. S. Sutton, "Reinforcement learning: An introduction," *A Bradford Book*, 2018.

[59] B. Zoph, "Neural architecture search with reinforcement learning," *arXiv preprint*, 2016.

[60] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *NeurIPS*, 2016.

[61] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *ICLR*, 2019.

[62] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, pp. 341–359, 1997.

[63] K. A. Smith-Miles, "Towards insightful algorithm selection for optimisation using meta-learning concepts," in *IJCNN*, 2008, pp. 4118–4124.

[64] J. Y. Kanda, A. C. de Carvalho, E. R. Hruschka, and C. Soares, "Using meta-learning to recommend meta-heuristics for the traveling salesman problem," in *ICML*, vol. 1, 2011, pp. 346–351.

[65] Y. Tian, S. Peng, T. Rodemann, X. Zhang, and Y. Jin, "Automated selection of evolutionary multi-objective optimization algorithms," in *SSCI*, 2019, pp. 3225–3232.

[66] A. E. Gutierrez-Rodríguez, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, "Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning," *Expert Syst. Appl.*, vol. 118, pp. 470–481, 2019.

[67] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K. C. Tan, and Y. Jin, "A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks," *TAI*, vol. 1, no. 1, pp. 5–18, 2020.

[68] Y. Li, J. Liang, K. Yu, K. Chen, Y. Guo, C. Yue, and L. Zhang, "Adaptive local landscape feature vector for problem classification and algorithm selection," *Appl. Soft Comput.*, vol. 131, p. 109751, 2022.

[69] X. Wu, Y. Zhong, J. Wu, B. Jiang, and K. C. Tan, "Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation," in *IJCAI*, 2024.

[70] N. Zhu, F. Zhao, and J. Cao, "A hyperheuristic and reinforcement learning guided meta-heuristic algorithm recommendation," in *CSCWD*, 2024, pp. 1061–1066.

[71] F. Tahernezhad-Javazm, D. Rankin, N. D. Bois, A. E. Smith, and D. Coyle, "R2 indicator and deep reinforcement learning enhanced adaptive multi-objective evolutionary algorithm," *arXiv preprint*, 2024.

[72] G. Cenikj, G. Petelin, and T. Eftimov, "Transoptas: Transformer-based algorithm selection for single-objective optimization," in *GECCO*, 2024, pp. 403–406.

[73] H. Samma, C. P. Lim, and J. M. Saleh, "A new reinforcement learning-based memetic particle swarm optimizer," *Appl. Soft Comput.*, vol. 43, pp. 276–297, 2016.

[74] A. K. Sadhu, A. Konar, T. Bhattacharjee, and S. Das, "Synergism of firefly algorithm and q-learning for robot arm path planning," *Swarm Evol. Comput.*, vol. 43, pp. 50–68, 2018.

[75] W. Ning, B. Guo, X. Guo, C. Li, and Y. Yan, "Reinforcement learning aided parameter control in multi-objective evolutionary algorithm based on decomposition," *Prog. Artif. Intell.*, vol. 7, pp. 385–398, 2018.

[76] Y. Liu, H. Lu, S. Cheng, and Y. Shi, "An adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning," in *CEC*, 2019, pp. 815–822.

[77] Z. Li, L. Shi, C. Yue, Z. Shang, and B. Qu, "Differential evolution based on reinforcement learning with fitness ranking for solving multimodal multiobjective problems," *Swarm Evol. Comput.*, vol. 49, pp. 234–244, 2019.

[78] G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter, "Learning step-size adaptation in CMA-ES," in *PPSN*, 2020, pp. 691–706.

[79] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Evolutionary framework with reinforcement learning-based mutation adaptation," *IEEE Access*, vol. 8, pp. 194 045–194 071, 2020.

[80] Y. Huang, W. Li, F. Tian, and X. Meng, "A fitness landscape ruggedness multiobjective differential evolution algorithm with a reinforcement learning strategy," *Appl. Soft Comput.*, vol. 96, p. 106693, 2020.

[81] Z. Hu, W. Gong, and S. Li, "Reinforcement learning-based differential evolution for parameters extraction of photovoltaic models," *Energy Rep.*, vol. 7, pp. 916–928, 2021.

[82] S. Yin, Y. Liu, G. Gong, H. Lu, and W. Li, "RLEPSO: Reinforcement learning based ensemble particle swarm optimizer," in *ACAI*, 2021, pp. 1–6.

[83] T. N. Huynh, D. T. Do, and J. Lee, "Q-learning-based parameter control in differential evolution for structural optimization," *Appl. Soft Comput.*, vol. 107, p. 107464, 2021.

[84] H. Xia, C. Li, S. Zeng, Q. Tan, J. Wang, and S. Yang, "A reinforcement-learning-based evolutionary algorithm using solution space clustering for multimodal optimization problems," in *CEC*, 2021, pp. 1938–1945.

[85] D. Wu and G. G. Wang, "Employing reinforcement learning to enhance particle swarm optimization methods," *Eng. Optim.*, vol. 54, no. 2, pp. 329–348, 2022.

[86] F. Wang, X. Wang, and S. Sun, "A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization," *Inf. Sci.*, vol. 602, pp. 298–312, 2022.

[87] K. Xue, J. Xu, L. Yuan, M. Li, C. Qian, Z. Zhang, and Y. Yu, "Multi-agent dynamic algorithm configuration," *NeurIPS*, vol. 35, pp. 20 147–20 161, 2022.

[88] Z. Hu and W. Gong, "Constrained evolutionary optimization based on reinforcement learning using the objective function and constraints," *KBS*, vol. 237, p. 107731, 2022.

[89] Y. Tian, X. Li, H. Ma, X. Zhang, K. C. Tan, and Y. Jin, "Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization," *TETCI*, vol. 7, no. 4, pp. 1051–1064, 2022.

[90] I. Fister, D. Fister, and I. Fister Jr, "Reinforcement learning-based differential evolution for global optimization," in *Differential Evolution: From Theory to Practice*, 2022, pp. 43–75.

[91] W. Li, P. Liang, B. Sun, Y. Sun, and Y. Huang, "Reinforcement learning-based particle swarm optimization with neighborhood differential mutation strategy," *Swarm Evol. Comput.*, vol. 78, p. 101274, 2023.

[92] H. Zhang, J. Sun, T. Bäck, Q. Zhang, and Z. Xu, "Controlling sequential hybrid evolutionary algorithm by q-learning [research frontier] [research frontier]," *CIM*, vol. 18, no. 1, pp. 84–103, 2023.

[93] X. Liu, J. Sun, Q. Zhang, Z. Wang, and Z. Xu, "Learning to learn evolutionary algorithm: A learnable differential evolution," *TETCI*, vol. 7, no. 6, pp. 1605–1620, 2023.

[94] S. Yin, M. Jin, H. Lu, G. Gong, W. Mao, G. Chen, and W. Li, "Reinforcement-learning-based parameter adaptation method for particle swarm optimization," *Complex Intell. Syst.*, vol. 9, no. 5, pp. 5585–5609, 2023.

[95] X. Meng, H. Li, and A. Chen, "Multi-strategy self-learning particle swarm optimization algorithm based on reinforcement learning," *MBE*, vol. 20, no. 5, pp. 8498–8530, 2023.

[96] Q. Yang, S.-C. Chu, J.-S. Pan, J.-H. Chou, and J. Watada, "Dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning for optimization problems," *Complex Intell. Syst.*, vol. 10, no. 2, pp. 1845–1877, 2024.

[97] Y. Han, H. Peng, C. Mei, L. Cao, C. Deng, H. Wang, and Z. Wu, "Multi-strategy multi-objective differential evolutionary algorithm with reinforcement learning," *KBS*, vol. 277, p. 110801, 2023.

[98] F. Zhao, Z. Wang, L. Wang, T. Xu, N. Zhu *et al.*, "A multi-agent reinforcement learning driven artificial bee colony algorithm with the central controller," *Expert Syst. Appl.*, vol. 219, p. 119672, 2023.

[99] Z. Hu, W. Gong, W. Pedrycz, and Y. Li, "Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization," *Swarm Evol. Comput.*, vol. 83, p. 101387, 2023.

[100] T. Li, Y. Meng, and L. Tang, "Scheduling of continuous annealing with a multi-objective differential evolution algorithm based on deep reinforcement learning," *TASE*, vol. 21, no. 2, pp. 1767–1780, 2023.

[101] L. Peng, Z. Yuan, G. Dai, M. Wang, and Z. Tang, "Reinforcement learning-based hybrid differential evolution for global optimization of interplanetary trajectory design," *Swarm Evol. Comput.*, vol. 81, p. 101351, 2023.

[102] X. Yu, P. Xu, F. Wang, and X. Wang, "Reinforcement learning-based differential evolution algorithm for constrained multi-objective optimization problems," *EAAI*, vol. 131, p. 107817, 2024.

[103] J. Hong, B. Shen, and A. Pan, "A reinforcement learning-based neighborhood search operator for multi-modal optimization and its applications," *Expert Syst. Appl.*, vol. 246, p. 123150, 2024.

[104] H. Zhang, J. Shi, J. Sun, A. W. Mohamed, and Z. Xu, "A gradient-based method for differential evolution parameter control by smoothing," in *GECCO*, 2024, pp. 423–426.

[105] H. Zhang, J. Sun, T. Bäck, and Z. Xu, "Learning to select the recombination operator for derivative-free optimization," *Sci. China Math.*, pp. 1–24, 2024.

[106] Z. Liao, Q. Pang, and Q. Gu, "Differential evolution based on strategy adaptation and deep reinforcement learning for multimodal optimization problems," *Swarm Evol. Comput.*, vol. 87, p. 101568, 2024.

[107] J. Wang, Y. Zheng, Z. Zhang, H. Peng, and H. Wang, "A novel multi-state reinforcement learning-based multi-objective evolutionary algorithm," *Inf. Sci.*, vol. 688, p. 121397, 2024.

[108] X. Wang, F. Wang, Q. He, and Y. Guo, "A multi-swarm optimizer with a reinforcement learning mechanism for large-scale optimization," *Swarm Evol. Comput.*, vol. 86, p. 101486, 2024.

[109] A. Bolufé-Röhler and B. Xu, "Deep reinforcement learning for smart restarts in exploration-only exploitation-only hybrid metaheuristics," in *MIC*, 2024.

[110] J. Pei, J. Liu, and Y. Mei, "Learning from offline and online experiences: A hybrid adaptive operator selection framework," in *GECCO*, 2024, pp. 1017–1025.

[111] S. Chaybouti, L. Dos Santos, C. Malherbe, and A. Virmaux, "Meta-learning of black-box solvers using deep reinforcement learning," in *NeurIPS*, 2022.

[112] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large language models as optimizers," 2024.

[113] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," in *CEC*, 2024, pp. 1–8.

[114] F. Liu, X. Lin, Z. Wang, S. Yao, X. Tong, M. Yuan, and Q. Zhang, "Large language model for multi-objective evolutionary optimization," *arXiv preprint*, 2023.

[115] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*, 2023, pp. 331–366.

[116] P.-F. Guo, Y.-H. Chen, Y.-D. Tsai, and S.-D. Lin, "Towards optimizing with large language models," *arXiv preprint*, 2023.

[117] X. Li, K. Wu, X. Zhang, H. Wang, and J. Liu, "B2Opt: Learning to optimize black-box optimization with little budget," *arXiv preprint*, 2023.

[118] S. Brahmachary, S. M. Joshi, A. Panda, K. Koneripalli, A. K. Sagotra, H. Patel, A. Sharma, A. D. Jagtap, and K. Kalyanaraman, "Large language model-based evolutionary optimizer: Reasoning with elitism," *arXiv preprint*, 2024.

[119] Z. Wang, S. Liu, J. Chen, and K. C. Tan, "Large language model-aided evolutionary search for constrained multiobjective optimization," in *ICIC*, 2024, pp. 218–230.
[120] W. Yi, R. Qu, L. Jiao, and B. Niu, "Automated design of metaheuristics using reinforcement learning within a novel general search framework," *TEC*, vol. 27, no. 4, pp. 1072–1084, 2022.
[121] F. Liu, X. Tong, M. Yuan, and Q. Zhang, "Algorithm evolution using large language model," *arXiv preprint*, 2023.
[122] J. Chen, Z. Ma, H. Guo, Y. Ma, J. Zhang, and Y.-J. Gong, "SYMBOL: Generating flexible black-box optimizers through symbolic equation learning," in *ICLR*, 2024.
[123] N. van Stein and T. Bäck, "LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics," *arXiv preprint*, 2024.
[124] Y. Huang, S. Wu, W. Zhang, J. Wu, L. Feng, and K. C. Tan, "Autonomous multi-objective optimization using large language model," *arXiv preprint*, 2024.
[125] R. Zhang, F. Liu, X. Lin, Z. Wang, Z. Lu, and Q. Zhang, "Understanding the importance of evolutionary search in automated heuristic design with large language models," in *PPSN*, 2024, pp. 185–202.
[126] Q. Zhao, T. Liu, B. Yan, Q. Duan, J. Yang, and Y. Shi, "Automated metaheuristic algorithm design with autoregressive learning," *arXiv preprint*, 2024.
[127] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *ECJ*, vol. 27, no. 1, pp. 3–45, 2019.
[128] J. R. Rice, "The algorithm selection problem," in *Advances in Computers*, 1976, vol. 15, pp. 65–118.
[129] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß, "Algorithm selection based on exploratory landscape analysis and cost-sensitive learning," in *GECCO*, 2012, pp. 313–320.
[130] P. Kerschke and H. Trautmann, "Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning," *ECJ*, vol. 27, no. 1, pp. 99–127, 2019.
[131] S. Hochreiter, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[132] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017.
[133] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter, "HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO," in *NeurIPS*, 2021.
[134] Á. Fialho, "Adaptive operator selection for optimization," Ph.D. dissertation, Université Paris Sud-Paris XI, 2010.
[135] S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter, "Automated dynamic algorithm configuration," *JAIR*, vol. 75, pp. 1633–1699, 2022.
[136] S. Biswas, D. Saha, S. De, A. D. Cobb, S. Das, and B. A. Jalaian, "Improving differential evolution through bayesian hyperparameter optimization," in *CEC*, 2021, pp. 832–840.
[137] J. Brest, M. S. Maučec, and B. Bošković, "Self-adaptive differential evolution algorithm with population size reduction for single objective bound-constrained optimization: Algorithm j21," in *CEC*, 2021, pp. 817–824.
[138] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Improved multi-operator differential evolution algorithm for solving unconstrained problems," in *CEC*, 2020, pp. 1–8.
[139] M. V. Seiler, J. Rook, J. Heins, O. L. Preuß, J. Bossek, and H. Trautmann, "Using reinforcement learning for per-instance algorithm configuration on the TSP," in *SSCI*, 2023, pp. 361–368.
[140] D. Karapetyan and G. Gutin, "Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem," *EJOR*, vol. 208, no. 3, pp. 221–232, 2011.
[141] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *CEC*, 2014, pp. 1658–1665.
[142] S. Wright *et al.*, "The roles of mutation, inbreeding, crossbreeding, and selection in evolution," 1932.
[143] R. Lange, Y. Tang, and Y. Tian, "Neuroevobench: Benchmarking evolutionary optimizers for deep learning applications," *NeurIPS*, 2023.
[144] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" *arXiv preprint*, 2022.
[145] R. Bellman, "A markovian decision process," *JMM*, pp. 679–684, 1957.
[146] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.
[147] V. Mnih, "Playing atari with deep reinforcement learning," *arXiv preprint*, 2013.
[148] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, 2016.
[149] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *ICRA*, 2019.
[150] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.
[151] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *NeurIPS*, 1999.
[152] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, 2017.
[153] M. Gao, X. Feng, H. Yu, and X. Li, "An efficient evolutionary algorithm based on deep reinforcement learning for large-scale sparse multiobjective optimization," *Appl. Intell.*, vol. 53, no. 18, pp. 21 116–21 139, 2023.
[154] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, no. 3, pp. 347–361, 1990.
[155] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optim. Methods Softw.*, vol. 36, no. 1, pp. 114–144, 2021.
[156] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *NeurIPS*, 2007.
[157] A. Maraval, M. Zimmer, A. Grosnit, and H. Bou Ammar, "End-to-end meta-bayesian optimisation with transformer neural processes," *NeurIPS*, 2024.
[158] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound constrained numerical optimization," Tech. Rep., 2021.
[159] U. Škvorc, T. Eftimov, and P. Korošec, "GECCO black-box optimization competitions: progress from 2009 to 2018," in *GECCO*, 2019, pp. 275–276.
[160] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *TEC*, vol. 10, no. 5, pp. 477–506, 2006.
[161] X. Li, A. Engelbrecht, and M. G. Epitropakis, "Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization," Tech. Rep., 2013.
[162] C. Li, S. Yang, T.-T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. Beyer, and P. N. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," Tech. Rep., 2008.
[163] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China, "Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization," Tech. Rep., 2013.
[164] M. A. Muñoz and K. Smith-Miles, "Generating new space-filling test instances for continuous black-box optimization," *ECJ*, vol. 28, no. 3, pp. 379–404, 2020.
[165] D. Vermetten, F. Ye, T. Bäck, and C. Doerr, "MA-BBOB: A problem generator for black-box optimization using affine combinations and shifts," *TELO*, 2024.
[166] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "AClib: A benchmark library for algorithm configuration," in *LION*, 2014, pp. 36–40.
[167] A. Kumar, G. Wu, M. Z. Ali, R. Mallipeddi, P. N. Suganthan, and S. Das, "A test-suite of non-convex constrained optimization problems from the real-world and some baseline results," *Swarm Evol. Comput.*, vol. 56, p. 100693, 2020.
[168] C. Doerr, H. Wang, F. Ye, S. Van Rijn, and T. Bäck, "IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv preprint*, 2018.
[169] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory landscape analysis," in *GECCO*, 2011, pp. 829–836.
[170] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue, "A study of fitness distance correlation as a difficulty measure in genetic programming," *TEC*, vol. 13, no. 2, pp. 213–239, 2005.
[171] K. M. Malan and A. P. Engelbrecht, "Quantifying ruggedness of continuous landscapes using entropy," in *CEC*, 2009, pp. 1440–1447.
[172] G. Merkuryeva and V. Bolshakovs, "Benchmark fitness landscape analysis," *IJSSST*, vol. 12, no. 2, pp. 38–45, 2011.
[173] M. Lunacek and D. Whitley, "The dispersion metric and the CMA evolution strategy," in *GECCO*, 2006, pp. 477–484.

[174] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Vérel, "Fitness clouds and problem hardness in genetic programming," in *GECCO*, 2004, pp. 690–701.

[175] L. Vanneschi, P. Collard, S. Verel, M. Tomassini, Y. Pirola, and G. Mauri, "A comprehensive view of fitness landscapes with neutrality and fitness clouds," in *EuroGP*, 2007, pp. 241–250.

[176] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking 2010: Experimental setup," Ph.D. dissertation, INRIA, 2010.

[177] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Tech. Rep., 2005.

[178] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, pp. 1–40, 2016.

[179] Y. Zhang and Q. Yang, "A survey on multi-task learning," *TKDE*, vol. 34, no. 12, pp. 5586–5609, 2021.

[180] M. V. Seiler, P. Kerschke, and H. Trautmann, "Deep-ELA: Deep exploratory landscape analysis with self-supervised pretrained transformers for single-and multi-objective continuous optimization problems," *arXiv preprint*, 2024.

[181] Z. Ma, J. Chen, H. Guo, and Y.-J. Gong, "Neural exploratory landscape analysis," *arXiv preprint*, 2024.