Machine Learning Algorithms for Improving Black Box Optimization Solvers

Morteza Kimiaei
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: kimiaeim83@univie.ac.at
WWW: http://www.mat.univie.ac.at/~kimiaei

Vyacheslav Kungurtsev
Department of Computer Science, Czech Technical University
Karlovo Namesti 13, 121 35 Prague 2, Czech Republic
email: vyacheslav.kungurtsev@fel.cvut.cz

Abstract. Black-box optimization (BBO) addresses problems where objectives are accessible only through costly queries without gradients or explicit structure. Classical derivative-free methods—line search, direct search, and model-based solvers such as Bayesian optimization—form the backbone of BBO, yet often struggle in high-dimensional, noisy, or mixed-integer settings.

Recent advances use machine learning (ML) and reinforcement learning (RL) to enhance BBO: ML provides expressive surrogates, adaptive updates, meta-learning portfolios, and generative models, while RL enables dynamic operator configuration, robustness, and meta-optimization across tasks.

This paper surveys these developments, covering representative algorithms such as NNs with the modular model-based optimization framework (`mlrMBO`), zeroth-order adaptive momentum methods (`ZO-AdaMM`), automated BBO (`ABBO`), distributed block-wise optimization (`DiBB`), partition-based Bayesian optimization (`SPBOpt`), the transformer-based optimizer (`B2Opt`), diffusion-model-based BBO, surrogate-assisted RL for differential evolution (`Surr-RLDE`), robust BBO (`RBO`), coordinate-ascent model-based optimization with relative entropy (`CAS-MORE`), log-barrier stochastic gradient descent (`LB-SGD`), policy improvement with black-box (`PIBB`), and offline Q-learning with Mamba backbones (`Q-Mamba`).

We also review benchmark efforts such as the NeurIPS 2020 BBO Challenge and the `MetaBox` framework. Overall, we highlight how ML and RL transform classical inexact solvers into more scalable, robust, and adaptive frameworks for real-world optimization.

Keywords. Black-Box Optimization; Machine Learning; Reinforcement Learning; Surrogate Models; Robust Optimization; Meta-Black-Box Optimization

Contents

1 Introduction

Machine Learning (ML) techniques have recently gained significant attention for their ability to model complex relationships and enhance algorithmic performance across diverse domains. In the realm of **Black Box Optimization (BBO)**—where explicit structural information about the objective function or constraints is unavailable—ML provides a powerful toolkit for improving the efficiency, robustness, and scalability of classical solvers.

BBO problems frequently arise in science and engineering when the optimization objective can only be evaluated through costly simulations, experiments, or function calls, without access to gradients or a problem-specific structure. These problems are inherently challenging: the search space may be high-dimensional, multimodal, and non-convex, while evaluations are often noisy or expensive. As a result, exact solution strategies are impractical, and practitioners rely on heuristic or stochastic algorithms such as evolutionary strategies, Bayesian optimization, and surrogate-assisted methods.

In this context, ML models offer valuable mechanisms for accelerating search. They can leverage previously collected evaluations to construct predictive surrogates, guide exploration–exploitation trade-offs, or adaptively tune algorithmic hyperparameters. Reinforcement Learning (RL) approaches, for example, can be used to learn search policies that dynamically allocate resources across candidate solutions. In contrast, supervised learning methods can extract patterns from historical runs to inform initialization or sampling strategies.

We first define the set of simple bounds

$$\mathbf{X} := \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\} \quad \text{with } \underline{x}, \overline{x} \in \mathbb{R}^n \ (\underline{x} < \overline{x}) \tag{1}$$

on variables $x \in \mathbb{R}^n$ (called the **box**). Then, the continuous BBO problem can be expressed in the same formal way as

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in C_{\text{co}}, \end{array} \tag{2}$$

where $f : C_{\text{co}} \subseteq \mathbf{X} \to \mathbb{R}$ is a real-valued (possibly non-convex) black box objective function defined on the **continuous nonlinear feasible set**

$$C_{\text{co}} := \{x \in \mathbf{X} \mid g(x) = 0, \quad h(x) \leq 0\}. \tag{3}$$

Here $g(x) = (g_1, \ldots, g_m)$ and $h(x) = (h_1, \ldots, h_p)$ represent equality and inequality constraints, respectively, both treated as black box functions accessible only through evaluations. For a positive integer q, we set

$$[q] := \{1, 2, \ldots, q\}.$$

We will use this notation throughout the paper.

In line with prior surveys and the book of Audet & Hare [2], our focus here is on how ML and RL can enhance **continuous** BBO methods. While extensions of the black-box paradigm to integer and mixed-integer domains are important in practice, we do not discuss them further in this work.

2 Contributions

The main contributions of this paper are as follows:

- In Section 3, we provide a unified taxonomy of classical BBO methods (line search, direct search, model-based, Bayesian optimization) and position recent ML and RL advances as enhancements that extend these foundations.
- In Section 4, we review recent ML and RL approaches designed to complement **classical BBO solvers**. Our emphasis is on their integration into **inexact solution methods**—procedures that do not guarantee global optimality but strive to deliver high-quality solutions under strict time or evaluation budgets. We begin with a brief overview of traditional BBO heuristics and then discuss how data-driven methods are being employed to extend or transform these strategies.
  - In Subsection 4.1, we survey major **ML-enhanced BBO** frameworks, including surrogate-based approaches such as neural networks with the modular model-based optimization framework (`mlrMBO` [5]), search partitioning for Bayesian optimization (`SPBOpt` [67]), and trust-region derivative-free optimization (`DFO-TR` [25]); optimizer-inspired updates such as the zeroth-order adaptive momentum method (`ZO-AdaMM` [9]); meta-learning portfolios such as the automated black-box optimizer (`ABBO` [54]) and distributed block-wise optimization (`DiBB` [14]); and generative optimizers such as the transformer-based optimizer (`B2Opt` [45]) and diffusion-model-based BBO ([46]).
  - In Subsection 4.2, we review **RL-enhanced BBO** methods that address robustness, including robust black-box optimization (`RBO` [10]), log-barrier stochastic gradient descent (`LB-SGD` [73]), and coordinate-ascent model-based optimization with relative entropy (`CAS-MORE` [34]); dynamic operator configuration via surrogate-assisted RL for differential evolution (`Surr-RLDE` [52]); policy search equivalence through policy improvement with black-box updates (`PIBB` [72]); and offline meta-optimization with decomposed Q-learning using a Mamba backbone (`Q-Mamba` [50]).

- In Section 5, we highlight **benchmarking and reproducibility efforts**, including the NeurIPS 2020 BBO Challenge [7], the `MetaBox` framework [51], and recent `MetaBBO-RL` evaluation studies [76], establishing standardized protocols for fair comparison.
- We emphasize the role of **continuous BBO** as a central application domain, where ML and RL provide practical efficiency and robustness beyond classical heuristics.

3 BBO Methods

BBO methods form the algorithmic foundation for tackling problems where the objective function and constraints are available only through evaluations, with no derivative or structural information. In this section, we survey the main families of classical derivative-free optimization (DFO) solvers—polling-based, surrogate-based, and local-approximation-based—as well as stochastic and evolutionary algorithms. These methods provide the groundwork upon which modern ML- and RL-enhanced approaches to BBO are built.

DFO encompasses a class of optimization algorithms designed to minimize or maximize an objective function without requiring gradient information. These methods are essential in scenarios where the objective function is non-differentiable, noisy, or computationally expensive, such as hyperparameter tuning, BBO, or model calibration in ML.

DFO algorithms and their applications were discussed in the books of Audet & Hare [2] and Conn et al. [11] and the survey paper of Larson et al. [44]. The numerical behavior of integer and mixed-integer DFO solvers was investigated in the survey of Ploskas & Sahinidis [61], while the numerical behavior of noiseless continuous DFO algorithms was investigated in the papers by Moré & Wild [55], Rios & Sahinidis [66], Kimiaei et al. [42], and Kimiaei & Neumaier [38], and of noisy continuous DFO algorithms was investigated in the recent papers of Kimiaei [39] and Kimiaei & Neumaier [40].

Classical DFO methods can be broadly categorized into three families [2, 11, 44]: **polling-based methods**, **surrogate-based methods**, and **local-approximation-based methods**. This taxonomy emphasizes the underlying search mechanism: polling-based methods explore through deterministic or random directions, surrogate-based methods build global or semi-global response models, and local-approximation-based methods exploit linear or quadratic models in trust-region or line-search frameworks. Bayesian optimization can be regarded as a surrogate-based method with a probabilistic model and specialized acquisition optimization.

Line-search (`LS`) methods can be divided into two classes depending on whether

approximate gradient information is used. The first LS class corresponds to LS+dd, while the second LS class corresponds to LS-dd.

LS+dd employs an approximated gradient, and the LS condition can take the form of the approximate Armijo, Goldstein, or Wolfe conditions. This class naturally falls under the category of **local-approximation-based methods**, since it relies on constructing first-order models to guide search. In noisy DFO, however, the efficiency of these methods is significantly reduced, as the approximate gradient may be inaccurate. Examples include the Armijo line search, which uses backtracking interpolation to enforce the Armijo condition; the Wolfe line search, which enforces both the Armijo and curvature conditions through interpolation and extrapolation; and the Goldstein line search, which extends Armijo with a two-sided condition. Recently, Neumaier et al. [59] proposed an improved Goldstein method that combines interpolation and extrapolation more efficiently. Representative solvers in this class include SSDFO [42] and FMINUNC [53], both based on approximate Wolfe conditions.

LS-dd replaces directional derivative scaled by a step size with a forcing function (positive and non-decreasing, e.g., $c_1\alpha^2$ with the tuning parameter $0 < c_1 < 1$ and the step size $\alpha > 0$) as the line-search condition. This class fits into the category of **polling-based methods**, since it relies solely on function evaluations along search directions without constructing explicit gradient approximations. Representative solvers in this category include DFLINIT [48] (integer deterministic LS), DFLBOX [47] and DFNDFL [26] (integer and continuous deterministic LS), VRBBO [38] and VRDFON [39] (continuous randomized LS), and SDBOX [49] (continuous deterministic LS). All these solvers can handle small- to large-scale DFO problems, while FMINUNC—based on BFGS Hessian approximation—is only suitable for small- to medium-scale problems.

In summary, classical DFO methods span a wide spectrum, from polling-based algorithms that rely solely on function values, to surrogate-based approaches constructing global models, and local-approximation methods that use first- or second-order information within trust-region or line-search frameworks. This taxonomy provides a unified view of the methodological foundations upon which more advanced, ML-enhanced BBO techniques are built.

3.1 Surrogate-Based Methods

The goal of surrogate-based optimization (SBO) is to construct and refine an inexpensive model of the objective function, which may be stochastic, noisy, or otherwise expensive to evaluate. By iteratively updating the surrogate with new evaluations, SBO methods guide the search toward promising regions of the domain while reducing the number of costly function calls [23, 65].

SBO constructs various surrogate models such as Gaussian processes, radial basis functions, and polynomial regression. The initial model can be constructed

by selecting sampling points, and then the models are updated based on the new accepted points. `pySOT` [18], `Dakota` [1], `SPLINE` [31], and `MISO` [56] are the four `SBO` solvers. `pySOT` is a flexible surrogate optimization framework supporting various surrogates and methods such as Gaussian processes, Kriging, and radial basis functions for global optimization and handling expensive black-box functions, `Dakota` is a multilevel parallel object-oriented framework using a wide range of optimization algorithms (e.g., surrogate-based optimization, trust-region algorithms, and evolutionary algorithms) for high-dimensional and multi-fidelity optimization problems, `SPLINE` is a radial basis function method with Cubic Splines for low- to medium-dimensional global optimization. `MISO` combines multi-fidelity models with surrogate optimization to optimize high-dimensional and expensive DFO problems.

Gaussian Process (`GP`) Surrogates. A `GP` surrogate defines a prior over functions $f : \mathbb{R}^d \to \mathbb{R}$ such that for any finite set $\{x_i\}_{i=1}^N$, the vector $(f(x_1), \ldots, f(x_N))$ follows a multivariate Gaussian distribution. A `GP` is specified by a mean function $m(x)$ and a kernel (covariance function) $k(x, x')$, giving predictive mean and variance

$$\mu(x) = k(x, X)(K + \sigma^2 I)^{-1} y, \qquad \sigma^2(x) = k(x, x) - k(x, X)(K + \sigma^2 I)^{-1} k(X, x),$$

where K is the kernel matrix on the training inputs X, y are observed outputs, and σ^2 is the noise variance. `GP`s provide both interpolation accuracy and calibrated uncertainty estimates, which are crucial for acquisition functions in Bayesian optimization.

Radial Basis Function (`RBF`) Surrogates. An `RBF` surrogate models the objective as a weighted sum of radially symmetric kernels centered at previously evaluated points:

$$\hat{f}(x) \;=\; \sum_{i=1}^{N} w_i \, \phi(\|x - x_i\|),$$

where $\{x_i\}_{i=1}^N \subset \mathbb{R}^d$ are the N sampled design points, $w_i \in \mathbb{R}$ are interpolation weights chosen to fit $f(x_i)$ at these points, $\phi : \mathbb{R}^+ \to \mathbb{R}$ is a radial kernel, commonly $\phi(r) = \exp(-\gamma r^2)$ with shape parameter $\gamma > 0$ (Gaussian `RBF`), or alternatives such as multiquadric or thin-plate splines. Because `RBF` surrogates interpolate known evaluations and are smooth, they are widely used in trust-region DFO to approximate the objective within the local trust-region.

Bayesian optimization (`BO`). The model-based `BO` methods construct a probabilistic model (often a `GP`) to approximate the nonlinear objective function and combine uncertainty to balance exploration and exploitation. A key

component of BO is the **acquisition function**, which guides exploration and exploitation based on the predictive posterior of the surrogate. Given predictive mean $\mu(x)$, standard deviation $\sigma(x)$, and the best observed value f_{best}, common acquisition functions are **upper confidence bound** (UCB), **probability of improvement** (**PI**), and **expected improvement** (EI). They are defined as

$$m(x,\theta) := \begin{cases} \mu(x) + \kappa\sigma(x) & \texttt{UCB}, \\ \Phi\Big((\mu(x) - f_{\text{best}} - \xi)/\sigma(x)\Big) & \texttt{PI}, \\ (\mu(x) - f_{\text{best}} - \xi)\Phi(z) + \sigma(x)\phi(z) & \texttt{EI}, \end{cases}$$

where $z = (\mu(x) - f_{\text{best}} - \xi)/\sigma(x)$, $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and PDF of the standard normal distribution, respectively. The parameters κ and ξ control the trade-off between exploration ($\sigma(x)$) and exploitation ($\mu(x)$). Optimizing $m(x,\theta)$ is itself a nontrivial subproblem, usually approached by multi-start local search or evolutionary heuristics.

GPyOpt [27], BOHB [20], Scikit-Optimize [12], Spearmint [69], and Dragonfly [37] are the six BO solvers. GPyOpt includes a GP-based BO library, Spearmint is a popular BO framework for hyperparameter tuning, Scikit-Optimize includes a lightweight library for BO with scikit-learn, BOHB combines BO with resource-aware HyperBand for scalable optimization, Dragonfly is a scalable BO package for multi-fidelity and high-dimensional problems.

Algorithm 1 provides a generic framework for SBO applied to the CNLP problem (2). (S0_1) initializes the procedure by generating well-distributed sample points via a space-filling design and evaluating their function values. (S1_1) constructs a surrogate model $m(x,\theta)$ over the feasible region, typically chosen as $\mathcal{R} := C_{\text{co}}$. The form of $m(x,\theta)$ depends on the surrogate type:

- For GP, θ encodes kernel hyperparameters (e.g., length scales, variance) and possibly acquisition parameters when used in BO;
- For RBF, θ includes basis function type and regularization weights;
- For BO, $m(x,\theta)$ corresponds to an acquisition function with θ governing the exploration–exploitation balance (e.g., κ in UCB, ξ in EI/PI).

(S2_1) computes the objective at the trial point selected by minimizing $m(x,\theta)$. (S3_1) augments the surrogate with the new sample $(x_{\text{trial}}, f_{\text{trial}})$ and re-solves the model to propose a new candidate. Finally, (S4_1) updates the incumbent best solution if $f_{\text{trial}} < f_{\text{best}}$.

Among these surrogate-based methods, GPs are particularly important since they underpin BO.

In (S0_1) of SBO, sample points (in ML an initial dataset) are often generated by **space-filling methods**, which aim to cover the search space Ω uniformly without clustering. Two common approaches are:

Algorithm 1 A Generic SBO Framework for the CNLP Problem (2)

Initialization: ($S0_1$) Use a space-filling method to generate n sample points, evaluate their function values, choose a sample point with the smallest function value as the best point x_{best}, let $f_{\text{best}} = f(x_{\text{best}})$, and take the tuning parameters.

repeat

Forming and solving $m(x, \theta)$: ($S1_1$) Form surrogate model $m(x, \theta)$ and find its solution $x_{\text{trial}} := \underset{x \in \mathcal{R}}{\operatorname{argmin}}\, m(x, \theta)$.

Computing f at x_{trial}: ($S2_1$) Compute $f_{\text{trial}} = f(x_{\text{trial}})$.

Updating surrogate model: ($S3_1$) Augment $(x_{\text{trial}}, f_{\text{trial}})$ to the list of sample points for forming the surrogate model in ($S1_1$) for the next iteration.

Updating the best point

($S4_1$) If $f_{\text{trial}} < f_{\text{best}}$, set $x_{\text{best}} = x_{\text{trial}}$ and $f_{\text{best}} = f_{\text{trial}}$.

until the stopping criterion is met

- **Latin Hypercube Sampling:** divides each input dimension into n intervals and samples one point per interval, ensuring uniform coverage of marginal distributions.
- **Sobol Sequences:** low-discrepancy quasi-random sequences that minimize discrepancy with respect to the uniform distribution, providing deterministic, evenly spread samples across $[0, 1]^d$.

These two methods are used in ML methods that will be discussed in Section 4.1.

A recent advancement ([41, Section 2 – Algorithm 1]) in space-filling techniques refines uniform distribution to ensure that a finite set of random sample points is both well-distributed and adequately spaced apart, thereby enhancing the likelihood of discovering a global minimizer.

3.2 Polling-Based Methods

In this section, we discuss two classical polling-based algorithms: **Direct Search (DS)** and **Line Search (LS-dd)** without approximate directional derivatives. Both methods rely purely on function evaluations along chosen directions and use a forcing function as a sufficient decrease condition, rather than constructing approximate gradients or directional derivatives. In this way, they explore the search space by polling trial points and accepting steps only if the

forcing condition is satisfied. This makes them particularly robust in settings where gradients are unavailable or unreliable, such as noisy or discontinuous BBO problems.

The difference between these two algorithm is that `LS-dd` performs an extrapolation step along a fixed direction as long as reductions in the objective function values are found while step sizes are increased. The goal of extrapolation is to speed up the algorithm to find an approximate stationary point.

These two algorithms, although in theory guaranteed only to converge to **approximate stationary points** (points at which the gradient norm falls below a given threshold) and often requiring second-order techniques to refine these into **approximate local minima** (points whose function values are close to those of local minima), are nevertheless able in practice to identify **approximate global minimizers** (points whose function values lie within a small tolerance of the global minimum) in BBO problems; e.g., see [38].

Direct Search (`DS`) methods. The goal of `DS` methods is to escape from regions close to a saddle point or maximizer by rejecting all trial points that violate a direct search condition. The search directions can be coordinate directions or random directions, or even any approximate descent directions if the gradient is approximated by fitting or a finite difference method. Three main versions of direct search methods are pattern search, the Nelder-Mead method, and mesh adaptive direct search. The state-of-the-art direct search solvers are `NOMAD` [16], `BFO` [62], `NMSMAX` [32], `PSM` [15], and `DSPFD` [28]. `NOMAD` is a mesh-adaptive direct search, which can handle small- to large-scale constrained mixed-integer DFO problems, but for large-scale problems is quite slow. `BFO` uses a direct search algorithm, which handles small- to large-scale constrained mixed-integer DFO problems. `NMSMAX` uses a nonlinear multistart maximum algorithm, which can handle small- and medium-scale DFO problems. `PSM` uses a pattern search algorithm, which is effective for small-scale DFO problems. `DSPFD` uses a randomized direct search algorithm, which can handle small- to large-scale linearly constrained DFO problems. All of them have excellent numerical performance to solve both noiseless and noisy DFO problems as well.

Algorithm 2 is a generic version of the `DS` method. This algorithm consists of four steps, namely ($S0_2$)-($S3_2$), which are designed to find an approximate stationary point (although in practice it can find in most cases an approximate global minimizer). ($S0_2$) is an initialization step, like choosing an initial point and step size, and tuning parameters. `DS` has alternately calls to ($S1_2$)-($S3_2$) until a global minimizer is found. In ($S1_2$), the search direction can be chosen from a set of coordinate directions or a set of random directions. In ($S2_2$), the trial point and its function value are computed. In ($S3_2$), the direct search direction is satisfied at the trial point, the trial point is accepted as the best point, and the step size is expanded; otherwise, the trial point is rejected and the corresponding step size is reduced.

Algorithm 2 A Generic DS Framework for the CNLP Problem (2)

Initialization: (S0$_2$) Given the initial point $x_0 \in \mathbb{R}^n$, the initial step size $\alpha \in \mathbb{R}_+$, the m number of directions in per iteration, the parameter $0 < c_1 < 1$ for the DS condition, and the parameter $\sigma > 1$ for updating the step size, set $x_{\text{best}} = x_0$ and $f_{\text{best}} = f(x_0)$.

repeat

for $i \in [m]$ **do**

Computing d: (S1$_2$) Compute a search direction d.

Computing x_{trial} and f_{trial}: (S2$_2$) Compute the trial point $x_{\text{trial}} = x_{\text{best}} + \alpha d$ and its function value $f_{\text{trial}} = f(x_{\text{trial}})$.

Update information: (S3$_2$) If $f_{\text{trial}} - f_{\text{best}} < -c_1\alpha^2$, set $x_{\text{best}} = x_{\text{trial}}$, $f_{\text{best}} = f_{\text{trial}}$, and $\alpha = \sigma\alpha$; otherwise, set $\alpha = \alpha/\sigma$.

end for

until the stopping criterion is met

Line Search (LS-dd) methods. LS-dd uses no approximate directional derivative. Algorithm 3 is a generic LS-dd framework without approximate directional derivative or approximate gradient vector for solving the CNLP problem (2). ($S0_3$) is the initialization step of LS, which takes the initial point and m step sizes, computes its function value, and saves them as the initial best point and its function value, respectively. The tuning parameters depend on which LS condition is recommended to be used. In ($S1_3$) the search direction is computed, which can be random or coordinate directions. ($S2_3$) is an extrapolation step. For the second class, the term $\alpha\nabla f(x_{\text{best}})^T d$ is replaced by $-\alpha^2$ in the Armijo condition, resulting in the LS condition $\tilde{\mu}(\alpha) \leq c_1$ without the approximate directional derivative $\nabla f(x_{\text{best}})^T d$, where $\tilde{\mu}(\alpha) := (f(x_{\text{best}} + \alpha d) - f(x_{\text{best}}))/(-\alpha^2)$. This LS method uses an extrapolation step, which differs from the extrapolation step used in the line searches of the first class. In an extrapolation step in the second class, the extrapolation step sizes are expanded by a factor (larger than one), and the corresponding trial points and their function values are computed until the LS-dd condition is violated. Then, in ($S3_3$), a trial point with the smallest function value is chosen as the best point by such an extrapolation step.

Algorithm 3 A Generic LS-dd Framework without approximate directional derivative for the CNLP Problem (2)

Initialization: ($S0_3$) the initial point $x_0 \in \mathbb{R}^n$, the number m of search directions in per iteration, the initial step size vector $\alpha \in \mathbb{R}^m$, the parameter $\sigma > 1$ for updating step sizes, and the parameter $0 < c_1 < 1$ for the LS condition, set $x_{\text{best}} = x_0$ and compute $f_{\text{best}} = f(x_0)$.

repeat

for $i \in [m]$ **do**

Computing the search direction: ($S1_3$) Compute the deterministic or randomized direction $d_i \in \mathbb{R}^n$. Then, set $k = 1$ and $\tilde{\alpha}_k = \alpha_i$.

repeat

Computing trial point: ($S2_3$) Compute the trial point $x^k_{\text{trial}} = x_{\text{best}} + \tilde{\alpha}_k d_i$ and $f_{\text{trial}} = f(x^k_{\text{trial}})$. Then, evaluate the Boolean variable dec $:= f_{\text{trial}} - f_{\text{best}} < -c_1\tilde{\alpha}_k^2$. If dec is true, update $\tilde{\alpha}_{k+1} = \sigma\tilde{\alpha}_k$.

until dec

Updating the best point and the step size: ($S3_3$) If at least one trial point satisfies the LS condition in ($S2_3$), set $b = \text{argmin}_k f(x^k_{\text{trial}})$, $\alpha_i = \tilde{\alpha}_b$, $x_{\text{best}} = x^b_{\text{trial}}$, and $f_{\text{best}} = f(x^b_{\text{trial}})$. Otherwise, reduce the step size to $\alpha_i = \alpha_i/\sigma$.

end for

until the stopping criterion is met

3.3 Local-Approximation-Based Methods

In this section, we discuss two classes of local-approximation-based methods that exploit approximate first-order information: **Line Search with directional derivatives (`LS+dd`)** and **Trust-Region (`TR`)** algorithms. Unlike polling-based approaches, which rely solely on function values and forcing functions, these methods construct local models by approximating gradients or directional derivatives, either through finite differences or interpolation. The resulting local approximations are then used to determine descent directions and step sizes, providing stronger theoretical convergence guarantees to stationary points. At the same time, when combined with curvature information or regularization strategies, they can be extended to approximate local minimizers in nonlinear BBO problems.

Algorithm 4 is a generic `LS+dd` framework with an approximate directional derivative or approximate gradient for solving the CNLP problem (2). ($S0_4$) is the initialization step of `LS+dd`, which takes the initial point and computes its function value, and saves them as the initial best point and its function value, respectively. The tuning parameters depend on which `LS+dd` condition is recommended to be used. In ($S1_4$), the gradient can be approximated by fitting or a finite difference method. ($S2_4$) is the computation of search direction, which can be any approximate descent directions if the gradient is approximated like approximate steepest descent, approximate conjugate gradient, or approximate quasi-Newton directions, and random or coordinate directions if the directional derivative is approximated by a finite difference method otherwise. ($S3_4$) includes either a backtracking or an interpolation step (or extrapolation step). In the first class, we first define the **Goldstein quotient** by

$$\mu(\alpha) := \frac{f(x_{\text{best}} + \alpha d) - f(x_{\text{best}})}{\alpha \nabla f(x_{\text{best}})^T d} \quad \text{for} \quad \alpha > 0.$$

Then, we discuss the four various `LS+dd` methods. To satisfy the Armijo condition $\mu(\alpha) \geq c_1$ or the Goldstein condition $c_1 \leq \mu(\alpha) \leq c_2$ with $0 < c_2 < c_1 < 1$, backtracking is used, where the step size α is reduced by a factor less than one until the `LS+dd` condition is satisfied. The Wolfe line search includes the Armijo condition and the curvature condition defined by

$$\nabla f(x_{\text{best}} + \alpha d)^T d \geq c_3 \nabla f(x_{\text{best}})^T d \quad \text{with } 0 < c_1 < c_3 < 1.$$

Until the Wolfe conditions are satisfied, an interpolation step or extrapolation step is performed. The improved Goldstein `LS+dd` proposed in [59] satisfies the sufficient descent condition

$$\mu(\alpha)|\mu(\alpha) - 1| \geq \beta \quad \text{for some fixed } \beta \in]0, 1/4[.$$

This can be done in two stages: an interpolation step or an extrapolation step is performed to create an interval, and then the geometric mean of the lower and upper bounds of the interval is taken as the new step size.

Algorithm 4 A Generic `LS+dd` Framework with Approximate Gradient for the CNLP Problem (2)

Initialization: ($S0_4$) Given the initial point $x_0 \in \mathbb{R}^n$ and the tuning parameters for the `LS` condition, set $x_{\text{best}} = x_0$ and $f_{\text{best}} = f(x_0)$.

repeat

Approximating gradient: ($S1_4$) Approximate the gradient by a finite difference method.

Computing the search direction: ($S2_4$) Compute an approximate descent direction d.

Updating the best point: ($S3_4$) Perform an interpolation step or an extrapolation step to update the best point $x_{\text{best}} = x_{\text{best}} + \alpha d$ and its function value $f_{\text{best}} = f(x_{\text{best}})$.

until the stopping criterion is met

`TR` methods approximate nonlinear objective function by a linear, quadratic, or cubic model, which is restricted to a trust region to avoid large steps and increase the accuracy of the model. If the agreement between the objective function and the model function is good, the trial point is accepted as the new point and the trust region remains unchanged or expanded; otherwise, the trial point is discarded and the trust region is reduced. `BOBYQA` [63], `BCDFO` [29], `UOBYQA` [64], `MATRS` [41], and `SNOBFIT` [35] are the four `TR` solvers, which are suitable for continuous small-scale DFO problems. Unlike these mentioned solver, `MATRS` employs randomized sampling to construct quadratic models, where the gradient is obtained through a fitting procedure and the symmetric Hessian is computed as the product of an approximate covariance matrix with its transpose.

A generic `TR` framework can be written in the form of Algorithm 1 for solving the CNLP problem (2). ($S0_1$) initializes `TR` by generating well-distributed sample points using a space-filling method and evaluating their function values. ($S1_1$) constructs a local surrogate model $m(x, \theta)$, where the trust region is defined as

$$\mathcal{R} := \{x \in C_{\text{co}} \mid \|x - x_{\text{best}}\| \leq \Delta\},$$

with C_{co} from (3) and the trust-region radius Δ (which initially is a positive tuning parameter). ($S2_1$) evaluates the function at the trial point. ($S3_1$) solves the surrogate within $\mathcal{R}$ to propose x_{trial}. Here, θ corresponds to the trust-region radius Δ. ($S4_1$) updates the incumbent if $f_{\text{trial}} < f_{\text{best}}$. The **trust-region ratio**

$$\rho := \frac{f_{\text{best}} - f_{\text{trial}}}{m(x_{\text{best}}, \Delta) - m(x_{\text{trial}}, \Delta)}$$

measures the agreement between the true objective and the surrogate model. If $\rho \geq \eta \in (0, 1]$, the step is considered successful, and Δ is unchanged or expanded

as $\Delta \leftarrow \lambda\Delta$ with $\lambda > 1$. Otherwise, the step is unsuccessful, and Δ is reduced as $\Delta \leftarrow \Delta/\lambda$.

3.4 Evolutionary and Population-Based Methods

In this section, we review classical and modern **evolutionary and population-based** approaches to BBO. These methods maintain and adapt a population of candidate solutions, using stochastic variation operators and selection mechanisms to explore the search space. Unlike model-based or gradient-driven techniques, they rely only on function evaluations, making them broadly applicable to non-differentiable and noisy objectives. We highlight four prominent families: Evolution Strategies (`ES`) and their principled variants such as Natural Evolution Strategies (`NES`) and Covariance Matrix Adaptation (`CMA-ES`), as well as Particle Swarm Optimization (`PSO`) and Differential Evolution (`DE`). Each embodies a different design philosophy for balancing exploration and exploitation while adapting the search distribution over time.

Evolution Strategies (`ES`). `ES` maintains a population of λ candidate solutions sampled from a search distribution $\pi(x;\theta)$, typically Gaussian:

$$x_i \sim \mathcal{N}(\mu_t, \Sigma_t), \qquad i \in [\lambda],$$

where $\mu_t \in \mathbb{R}^d$ is the mean vector, $\Sigma_t \in \mathbb{R}^{d\times d}$ is the covariance matrix, and $\theta = (\mu_t, \Sigma_t)$ are the distribution parameters at iteration t. Each solution x_i is assigned a weight w_i based on its fitness rank, with $\sum_{i=1}^{\lambda} w_i = 1$. The distribution parameters are then updated by weighted recombination:

$$\mu_{t+1} = \mu_t + \eta \sum_{i=1}^{\lambda} w_i (x_i - \mu_t),$$

where $\eta > 0$ is the learning rate (also called step size). Some `ES` variants also adapt Σ_t using rank-one or rank-μ updates. Here, recombination refers to updating the distribution parameters (e.g., the mean) as a weighted average of sampled candidates, with higher-ranked solutions contributing more strongly.

Covariance Matrix Adaptation `ES` (`CMA-ES`). `CMA-ES` is an adaptive variant of `ES` that updates not only the mean μ_t but also the covariance matrix Σ_t of the search distribution. Given a population $\{x_i\}_{i=1}^{\lambda}$ sampled from $\mathcal{N}(\mu_t, \Sigma_t)$, with normalized recombination weights w_i satisfying $\sum_{i=1}^{\lambda} w_i = 1$, the covariance update is

$$\Sigma_{t+1} \;=\; (1 - c_\Sigma)\Sigma_t + c_\Sigma \sum_{i=1}^{\lambda} w_i \, (x_i - \mu_t)(x_i - \mu_t)^\top,$$

where $c_\Sigma \in (0,1)$ is the covariance learning rate. The recombination weights w_i are normalized coefficients assigned according to the rank of each candidate, ensuring that better-performing individuals have larger influence on the mean and covariance update. This adaptation learns the principal directions of successful search steps, enabling the algorithm to align its sampling distribution with the local geometry of the objective. As a result, `CMA-ES` is scale-invariant and performs well on ill-conditioned, non-separable optimization landscapes, making it one of the most effective general-purpose black-box optimizers (see, e.g., [40, 41]).

Particle Swarm Optimization (`PSO`). `PSO` is a population-based algorithm where each particle i maintains a position $x_i^t \in \mathbb{R}^d$ and velocity $v_i^t \in \mathbb{R}^d$ at iteration t. The update rules are

$$v_i^{t+1} \;=\; \omega v_i^t + c_1 r_1 \,(p_i^* - x_i^t) + c_2 r_2 \,(g^* - x_i^t), \qquad x_i^{t+1} \;=\; x_i^t + v_i^{t+1},$$

where $\omega > 0$ is the inertia weight controlling momentum from the previous velocity, $c_1 > 0$ is the cognitive coefficient weighting attraction toward the particle's personal best p_i^*, $c_2 > 0$ is the social coefficient weighting attraction toward the global best g^* across all particles, $r_1, r_2 \sim \mathrm{Unif}(0,1)$ are independent random scalars introducing stochasticity. The balance of inertia, cognitive, and social terms enables `PSO` to explore broadly while converging toward high-quality regions of the search space.

Differential Evolution (`DE`). `DE` is a population-based evolutionary algorithm for continuous (and mixed-integer via adaptations) BBO (cf. [71]). Given a population $\{x_i^t\}_{i=1}^\lambda \subset \Omega$ at iteration t, a **mutation** vector is formed for each target x_i^t (e.g., `DE/rand/1`):

$$v_i^t \;=\; x_{r_1}^t \;+\; F\big(x_{r_2}^t - x_{r_3}^t\big), \quad r_1, r_2, r_3 \text{ distinct and } r_k \neq i,\; F \in (0,2).$$

A **crossover** (binomial) produces a trial u_i^t with rate $C_r \in [0,1]$:

$$u_{i,j}^t \;=\; \begin{cases} v_{i,j}^t, & \text{if } r_j \le C_r \text{ or } j = j_{\mathrm{rand}}, \\ x_{i,j}^t, & \text{otherwise,} \end{cases} \qquad r_j \sim \mathrm{Unif}(0,1).$$

Here, recombination (often called crossover) denotes the coordinate-wise mixing of the target vector x_i^t and the mutant vector v_i^t, controlled by the crossover rate C_r. Finally, **selection** is greedy (for minimization):

$$x_i^{t+1} \;=\; \begin{cases} u_i^t, & \text{if } f(u_i^t) \le f(x_i^t), \\ x_i^t, & \text{otherwise.} \end{cases}$$

Different strategies are specified by a naming convention of the form `DE/`x`/n/`c, where:

- $x \in \{\texttt{rand}, \texttt{best}\}$ indicates whether the base vector is chosen randomly (`rand`) or as the current best (`best`);
- $n \in \{1, 2, \dots\}$ is the number of difference vectors added (e.g., `1` uses a single ($x_{r_2} - x_{r_3}$), `2` uses two such differences summed);
- $c \in \{\texttt{bin}, \texttt{exp}\}$ specifies the crossover scheme: binomial (`bin`) or exponential (`exp`).

For example:

- `DE/rand/1/bin`: base vector is random, one difference vector is added, binomial crossover.
- `DE/best/1/bin`: base vector is the current best, one difference vector is added, binomial crossover.
- `DE/rand/2/bin`: base vector is random, two difference vectors are used, binomial crossover.

Natural Evolution Strategy (`NES`). `NES` is a principled subclass of evolution strategies that update the search distribution using the **natural gradient** of the expected fitness [74]. Rather than applying heuristic rank-based recombination, `NES` formulates the objective as

$$J(\theta) \;=\; \mathbb{E}_{x\sim\pi(x;\theta)}[f(x)],$$

and estimates its gradient via the log-likelihood trick:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim\pi(x;\theta)}[f(x)\,\nabla_\theta \log \pi(x;\theta)]\,.$$

In contrast to the weighted recombination step in standard `ES` (where new parameters are formed by averaging selected individuals), `NES` updates parameters by following the natural gradient of the expected fitness. The update then follows the **natural gradient**, $\tilde{\nabla}_\theta J(\theta) = F^{-1}\nabla_\theta J(\theta)$, where F is the Fisher information matrix of the distribution family. This natural-gradient correction accounts for the information geometry of the parameter space, yielding more stable and efficient adaptation of the mean and covariance compared to standard `ES`. As a result, `NES` provides a theoretically grounded alternative to heuristic strategies, and is closely related to `CMA-ES`, though derived explicitly from information-geometric principles.

3.5 Zero-Order Gradient Estimation

This section discusses the Zero-Order (ZO) methods that will be used in Subsection 4.1.3. ZO methods optimize black-box functions by constructing stochastic estimates of the gradient using only function evaluations. Since no explicit derivative information is available, ZO estimators play a central role in bridging BBO with gradient-based techniques from classical nonlinear programming and deep learning [9, 58, 68].

One-Point vs. Two-Point Estimators. A basic approach is the one-point estimator

$$\hat{\nabla} f(x) \;=\; \frac{f(x+\mu u)}{\mu}\, u,$$

where $x \in \mathbb{R}^d$ is the current iterate, $u \in \mathbb{R}^d$ is a random perturbation, and $\mu > 0$ a smoothing parameter. While simple, this estimator is biased for the gradient of the smoothed objective. The symmetric two-point estimator

$$\hat{\nabla} f(x) \;=\; \frac{f(x+\mu u) - f(x-\mu u)}{2\mu}\, u, \qquad u \sim \mathcal{N}(0, I_d),$$

is unbiased and generally preferred in practice, though it requires twice as many function queries per direction [58, 68], where $\mathcal{N}(0, I_d)$ denotes the standard multivariate normal distribution with mean zero and identity covariance.

Variance Reduction. The variance of ZO estimators can be reduced by:

- **Mini-batching:** averaging across multiple perturbation directions $\{u_j\}_{j=1}^m$,

$$\hat{\nabla} f(x) = \frac{1}{m} \sum_{j=1}^{m} \frac{f(x+\mu u_j) - f(x-\mu u_j)}{2\mu}\, u_j,$$

  at the cost of $2m$ function evaluations per iteration.

- **Antithetic sampling:** using pairs $(u, -u)$ to cancel odd-order terms in the estimator, improving accuracy with no extra cost.

- **Orthogonal perturbations:** sampling perturbation vectors that form an orthogonal basis, which reduces redundancy and improves coverage of the search space [4].

Choice of Perturbation Distribution. Randomized directions u can be sampled uniformly on the unit sphere

$$u \sim \mathrm{Unif}(\mathbb{S}^{d-1}),$$

ensuring isotropic exploration and unbiased estimates, or drawn from a Gaussian distribution $u \sim \mathcal{N}(0, I_d)$, which after normalization concentrates near the unit sphere. The choice affects the variance of the estimator and hence the convergence rate. In high dimensions, orthogonal Gaussian perturbations are often used for efficiency [17].

Representative `ZO` Methods. Beyond the estimators themselves, several gradient-free optimization algorithms have been developed:

- `ZO-SGD` (Zeroth-Order Stochastic Gradient Descent) [24]. Using the basic `ZO` gradient estimate $\hat{g}_t$, the update is

  $$x_{t+1} \;=\; x_t - \eta_t \hat{g}_t,$$

  where $\eta_t > 0$ is the learning rate and $\hat{g}_t$ is obtained by one-point or two-point random-direction queries.

- `ZO-SCD` (Zeroth-Order Stochastic Coordinate Descent) [57]. At iteration t, a coordinate $i_t \in \{1, \ldots, d\}$ is chosen uniformly at random. The coordinate-wise estimator is

  $$\hat{g}_{t,i_t} \;=\; \frac{f(x_t + \mu e_{i_t}) - f(x_t - \mu e_{i_t})}{2\mu},$$

  where e_{i_t} is the i_t-th standard basis vector in $\mathbb{R}^d$. The update is then

  $$x_{t+1} \;=\; x_t - \eta_t \hat{g}_{t,i_t} e_{i_t}.$$

- `ZO-signSGD` [4]. Instead of using raw estimates, the sign of each coordinate is taken:

  $$x_{t+1} \;=\; x_t - \eta_t \,\mathrm{sign}(\hat{g}_t),$$

  where $\mathrm{sign}(\cdot)$ is applied element-wise to the stochastic gradient estimate $\hat{g}_t$. This improves robustness to noise in high-dimensional settings.

- `ZO-AdaMM` (Zeroth-Order Adaptive Momentum Method) [9]. This method extends adaptive moment estimation (`Adam`/`AMSGrad`) to the zeroth-order case. At iteration t, with gradient estimate $\hat{g}_t$, the updates are:

  $$m_t \;=\; \beta_{1,t} m_{t-1} + (1 - \beta_{1,t})\hat{g}_t, \qquad v_t \;=\; \beta_2 v_{t-1} + (1 - \beta_2)(\hat{g}_t \odot \hat{g}_t),$$

where m_t and v_t are first- and second-moment accumulators, $\beta_{1,t}, \beta_2 \in (0,1)$ are decay factors, and $\odot$ denotes element-wise product. With $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$ (`AMSGrad` correction), the parameter update is

$$x_{t+1} \;=\; \Pi_{\mathcal{X},\sqrt{\hat{V}_t}}\Big(x_t - \alpha_t \hat{V}_t^{-1/2} m_t\Big)\,,$$

where α_t is the step size, $\hat{V}_t = \mathrm{diag}(\hat{v}_t)$, and $\Pi_{\mathcal{X},H}(\cdot)$ is projection onto feasible set $\mathcal{X}$ under the Mahalanobis norm induced by matrix H.

Applications. `ZO` estimators are widely used in:

- **Hyperparameter optimization:** gradient-free training of ML models when the loss is only available via cross-validation.
- **Adversarial ML:** generating adversarial examples for deep networks without access to gradients [8].
- **Simulation-based optimization:** optimizing objectives that are accessible only via expensive black-box simulators in engineering and scientific computing.

By combining unbiased gradient estimation with variance-reduction techniques, `ZO` methods extend the reach of adaptive stochastic gradient algorithms (e.g., `Adam`, `AMSGrad`) to BBO settings [9].

3.6 BBO and Its Applications

In this section, we illustrate how BBO methods are applied across operations research, engineering, and machine learning. Our goal is to highlight both classical DFO use cases in OR and the generic BBO workflow that underpins modern applications, thereby motivating the role of BBO as a unifying paradigm for complex real-world problems.

DFO Applications in OR. DFO methods are often used in OR to solve complex optimization problems where derivatives of the objective function or constraints are not available, expensive to calculate, or do not exist at all. These methods are particularly suitable for BBO problems where the objective function is evaluated by simulations, experiments, or computationally intensive models.

In robust and stochastic optimization, uncertainties in **decision-making models** can be addressed by surrogate-based DFO algorithms such as `BO` (cf. [23]).

Another interesting application in **energy systems optimization** is energy distribution networks, where black-box energy models with constraints (e.g., demand–supply balance and renewable integration) must be optimized using stochastic DFO techniques to handle uncertain energy outputs. DFO is also a powerful tool to capture the nonlinear, discrete, and uncertain properties of **supply chain problems** (cf. [30, 33]), where brute-force enumeration is infeasible.

Under the taxonomy of polling-based, surrogate-based, and local-approximation-based methods, different strategies are used in OR applications: polling-based methods (e.g., direct search, randomized line search) provide robustness in noisy or discontinuous simulation models; surrogate-based methods (e.g., `GP`s, `BO`) are effective for costly simulation-driven planning; and local-approximation-based methods (e.g., trust-region models, gradient approximations) are particularly suited to small- and medium-scale models with relatively smooth structure.

Beyond supply chains, DFO methods and their metaheuristic extensions are applied in **logistics and transportation** [30], **financial portfolio optimization** [19], **manufacturing and production scheduling** [36], and many other complex OR settings where explicit derivatives are unavailable or unreliable.

A Generic BBO Workflow. Kumagai and Yasuda [43] survey the landscape of BBO and its practical applications in Artificial Intelligence (AI) and industry. BBO refers to the optimization of an objective $f(x)$ that is expensive to evaluate and provides no gradient or structural information. Typical applications include hyperparameter tuning, simulation-based design, robotics control, real-time decision-making, and large-scale industrial systems (e.g., energy, medical, manufacturing).

A generic BBO workflow consists of the following steps:

- Defining the objective $f(x)$ and feasible domain Ω.
- Initializing with a set of exploratory samples, often using random or space-filling designs.
- Using an optimizer (`BO`, `ES`, DFO methods) to propose candidate solutions x_{trial}.
- Evaluating $f(x_{\text{trial}})$ and update the incumbent best solution.
- Optionally refining surrogate models or partition structures to guide future search.

This general framework is widely applicable because:

- It accommodates objectives that are non-differentiable, noisy, or simulation-based.
- Surrogates, AI-based models, and meta-learning methods enable efficient reuse of past evaluations.
- The same structure can integrate different optimizers and leverage HPC or simulation tools, making it suitable for industrial-scale systems.

Thus, BBO serves as a unifying paradigm that bridges optimization theory with real-world deployment in AI and engineering.

Algorithm 5 captures the generic workflow of BBO across different domains. In the **problem setup step** ($S0_5$), the objective function $f(x)$ and feasible domain Ω are specified, often representing a simulation or industrial process. The **initialization step** ($S1_5$) generates a small set of exploratory samples—through random sampling or space-filling designs—and may fit an initial surrogate model. In the **proposal step** ($S2_5$), an optimization strategy such as `BO`, `ES`, or DFO solver selects promising candidate points x_{trial}. These candidates are then tested in the **evaluation step** ($S3_5$) by querying the true objective, and the incumbent best solution x_{best} is updated. To improve efficiency, the **model/partition refinement step** ($S4_5$) can update surrogate models or adapt the search space based on past evaluations. This cycle continues until the evaluation budget is reached ($S5_5$). Because this structure is flexible and optimizer-agnostic, it can accommodate noisy or non-differentiable objectives, reuse past evaluations, and integrate with large-scale simulation tools—making it a unifying approach for both AI research and industrial applications.

Algorithm 5 Generic BBO Workflow

Problem setup: ($S0_5$) Define objective $f(x)$ and feasible domain Ω.

Initialization: ($S1_5$) Generate initial samples and, if applicable, fit a surrogate model.

Proposal: ($S2_5$) Use an optimizer (`BO`, `EA`, DFO) to propose candidate x_{trial}.

Evaluation: ($S3_5$) Evaluate $f(x_{\text{trial}})$ and update the current best solution x_{best}.

Model/Partition refinement: ($S4_5$) Optionally refine surrogate models or partitioning strategies to guide search.

Termination: ($S5_5$) Repeat until evaluation budget B is exhausted.

3.7 Recommendation and Conclusion

We classified DFO solvers into three main categories according to the referee's recommended taxonomy: **Polling-Based Methods**, **Surrogate-Based Methods**, and **Local-Approximation-Based Methods**. Table 1 summarizes representative algorithms in each category, together with their scalability by problem size, whether they provide exact guarantees, and their compatibility with high-performance computing (HPC).

Polling-based solvers (e.g., `NOMAD`, `BFO`) are robust and derivative-free, but can become slow on medium- and large-scale problems unless parallelized (e.g., via `MPI` [21] or `CUDA` [13]). Surrogate-based methods (e.g., `pySOT`, `Dakota`, `BOHB`) provide powerful global modeling capabilities but are typically too slow beyond dimension $d \geq 30$ unless combined with parallelization strategies [3, 69]. Local-approximation-based methods (e.g., `SSDFO`, `FMINUNC`, `BOBYQA`) rely on gradient or model approximations and are generally well-suited for small- to medium-scale problems. Their performance may deteriorate in noisy or very high-dimensional settings, although `SSDFO` remains effective in large-scale cases thanks to its use of subspace techniques. Some hybrid solvers, such as `NOMAD`, `VRBBO`, and `VRDFON`, include both model-free and model-based variants.

category	**problem size**			**HPC?**	**software/references**
	small	**medium**	**large**		
Polling-Based	+	+	+	±	`NOMAD` [16], `BFO` [62], `NMSMAX` [32], `PSM` [15], `DSPFD` [28], `DFLINIT` [48], `DFLBOX` [47], `DFNDFL` [26], `VRBBO` [38], `SDBOX` [49], `VRDFON` [39]
Surrogate-Based	+	+	±	±	`pySOT` [18], `Dakota` [1], `SPLINE` [31], `MISO` [56], `GPyOpt` [27], `BOHB` [20], `Scikit-Optimize` [12], `Spearmint` [69], `Dragonfly` [37], `SNOBFIT` [35], [69]
Local-Approximation-Based	+	+	±	±	`SSDFO` [42], `FMINUNC` [53], `BOBYQA` [63], `BCDFO` [29], `UOBYQA` [64], `MATRS` [41]

Table 1: Classification of DFO solvers under the taxonomy: polling-based, surrogate-based, and local-approximation-based methods.

4 Neural Networks as Enhancers for BBO

Neural networks (NNs) have emerged as powerful tools for enhancing BBO, where the objective and constraints can only be accessed through expensive function evaluations, and gradient information is unavailable. Classical DFO methods often struggle in such settings due to the combinatorial growth of discrete assignments, the complexity of continuous domains, and the high cost of evaluations.

As expressive function approximators, adaptive optimizers, meta-learners, and generative models, NNs introduce new mechanisms for guiding the search process, reusing data efficiently, and scaling optimization to complex domains. In the following, we survey representative NN-driven approaches that enhance BBO, including surrogate-based formulations solved via modular model-based frameworks [5], adaptive momentum methods [9], meta-learning portfolios [14, 54], and neural generative optimizers [45, 46].

The methods described in Section 3 (line search, direct search, and model-based solvers, with `BO` as a special case of the latter) constitute the **classical foundation** of BBO. They provide general strategies for searching without gradients, but each faces limitations in practice: line search and direct search may scale poorly in high dimensions; trust-region surrogates may struggle with non-smooth or categorical domains; and `BO` is limited by surrogate expressiveness and acquisition optimization.

ML and RL do not introduce entirely new classes of BBO solvers, but instead **enhance existing ones** by providing richer models, adaptive updates, and data-driven strategies. For example:

- ML-based surrogates (e.g., `mlrMBO` [5]) extend the surrogate modeling paradigm of `BO` and model-based DFO.
- Optimizer-inspired updates (e.g., `ZO-AdaMM` [9]) import adaptive learning-rate and momentum techniques from ML into gradient-free search, improving robustness over classical line/direct search.
- Meta-learning and portfolio methods (e.g., `ABBO` [54], `DiBB` [14], `SPBOpt` [67]) generalize the algorithm-selection problem already implicit in DFO, leveraging ML to choose or adapt optimizers across tasks.
- Generative ML models (e.g., `B2Opt` [45], `DiffBBO` [46]) offer new ways of sampling candidate solutions, complementing traditional acquisition optimization in surrogate-based methods.
- RL methods (e.g., `RBO` [10], `CAS-MORE` [34], `LB-SGD` [73], `Surr-RLDE` [52], `Q-Mamba` [50]) extend stochastic search and evolutionary strategies by for-

mulating optimization as sequential decision-making, enabling robustness under noise, constraint handling, and dynamic operator configuration.

Thus, the ML and RL sections can be viewed as a **second layer** built on top of the classical BBO taxonomy: classical BBO defines the optimization backbone, while ML and RL provide modern enhancements to make these solvers more scalable, robust, and adaptive.

4.1 ML Enhancements in BBO

ML provides a versatile set of tools to enhance BBO, where search must be conducted without explicit gradients and often under the complexity of combinatorial, continuous, or mixed domains. While classical approaches, such as `BO` and `ES`, offer general-purpose strategies, ML introduces richer models and adaptive mechanisms that improve efficiency, scalability, and robustness.

Broadly, ML contributes to BBO in four complementary ways: (i) through **surrogate modeling**, where expressive regressors approximate the expensive objective; (ii) through **optimizer-inspired updates**, which transfer techniques such as adaptive momentum from deep learning to the zeroth-order setting; (iii) through **meta-learning and algorithm portfolios**, which select or adapt optimizers across tasks; and (iv) through **generative neural models**, which directly learn distributions over promising solutions.

In the following subsections, we review eight representative methods that illustrate these roles, ranging from `mlrMBO` to recent advances such as `B2Opt` and `DiffBBO` (for more details see Table 2).

Method	Core Idea	Domain	Refs.
`mlrMBO`	Modular `SMBO` with flexible surrogates	Mixed/Continuous	[5]
`ZO-AdaMM`	Zeroth-order Adam with adaptive moments	Continuous/Noisy	[9]
`ABBO`	Algorithm selection and chaining	General BBO	[54]
`DiBB`	Distributed block-wise optimization	High-dimensional	[14]
`SPBOpt`	Partition-based local `BO` for low budgets	Low-budget BBO	[67]
`DFO-TR`	Trust-region DFO applied to ML objectives	Continuous/Noisy	[25]
`B2Opt`	Transformer crossover/mutation/selection	Low-budget BBO	[45]
`DiffBBO`	Reward-conditioned diffusion sampling	Offline/Data-driven	[46]

Table 2: Representative ML-enhanced methods for BBO.

4.1.1 Background

Modern BBO builds on a diverse set of concepts from optimization, ML, and RL. To provide a common foundation for the methods reviewed below, we high-

light a few recurring components: (i) surrogate models such as GPs, random forests, and radial basis functions, which approximate expensive objectives and provide uncertainty estimates; (ii) population-based optimizers including Evolution Strategies (ES), Covariance Matrix Adaptation (CMA-ES), Particle Swarm Optimization (PSO), and Differential Evolution (DE); (iii) acquisition functions in Bayesian optimization (EI, UCB, PI) that balance exploration and exploitation; (iv) distributed formulations such as block-wise optimization (DiBB), which scale solvers to high-dimensional problems; and (v) neural modules that enable generative optimizers, e.g., attention-based crossover, feed-forward mutation, and residual selection in B2Opt.

These concepts are not introduced here as basic definitions, but rather as building blocks that will be referenced in subsequent subsections on surrogate modeling, optimizer-inspired methods, meta-learning, and generative modeling.

In this survey, we do not aim to investigate how ML and RL improve classical evolution strategies; rather, we focus on hybrid methods, some of which incorporate ES as a component within broader ML- or RL-enhanced BBO frameworks.

Area Under the Receiver Operating Characteristic Curve (AUC). Ranking performance of a classifier is measured by AUC. Formally, if $s(x)$ is a real-valued scoring function, then

$$\texttt{AUC} \;=\; \Pr\big(s(x^+) > s(x^-)\big),$$

the probability that a randomly chosen positive example x^+ receives a higher score than a randomly chosen negative example x^-. Equivalently, AUC is the integral of the ROC (Receiver Operating Characteristic) curve, which plots true positive rate against false positive rate under varying thresholds. Here, the ROC curve is the function that shows how well a classifier separates positives from negatives at every possible threshold, and AUC is the area under that curve.

Distributed Block-Wise Optimization. In distributed block-wise optimization, the decision vector $x = (x^{(1)}, \ldots, x^{(k)})$ is partitioned into blocks. Each block B_j is optimized by a solver π_j, using access to the shared dataset $D_t = \{(x_i, f(x_i))\}_{i=1}^{t}$ of all past evaluations:

$$x_{t+1}^{(j)} = \pi_j(x_t^{(j)}; D_t).$$

A central coordinator then assembles the block-wise proposals into a full candidate x_{t+1}, evaluates the black-box objective $f(x_{t+1})$, and appends the result to D_t. This preserves the convergence properties of the base algorithm while achieving wall-clock scalability through parallelism.

Random Forest Surrogates. A **Random Forest** is an ensemble of M regression trees $\{T_j\}_{j=1}^{M}$, combined by averaging:

$$\hat{f}(x) \;=\; \frac{1}{M}\sum_{j=1}^{M} T_j(x).$$

Each tree T_j is trained on a bootstrap sample of the dataset, and at each split only a random subset of features is considered. Formally, a regression tree partitions the input space into regions $\{R_\ell\}$ with constant predictions c_ℓ, so that

$$T_j(x) = \sum_{\ell} c_\ell \, \mathbf{1}[x \in R_\ell].$$

The ensemble reduces variance by aggregating across diverse trees, while the random feature selection at splits improves robustness. In BBO, Random Forest surrogates are particularly effective in mixed-variable and categorical domains, since they natively handle discrete features without requiring embeddings, and scale well to high dimensions.

Gradient Boosting (`GB`). `GB` is an ensemble learning method that builds a strong predictor by sequentially combining weak learners, typically regression trees. Let $\{(x_i, y_i)\}_{i=1}^{N}$ be the training data and $F_m(x)$ the ensemble after m iterations. The procedure starts with an initial model $F_0(x)$, often chosen as the mean of y_i for regression. At iteration m, a weak learner $h_m(x)$ is trained to approximate the negative gradient of the loss $\ell(y, F_{m-1}(x))$ with respect to the predictions:

$$r_i^{(m)} \;=\; -\left.\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\right|_{F=F_{m-1}}.$$

The weak learner $h_m(x)$ fits these pseudo-residuals $\{r_i^{(m)}\}$, and the ensemble is updated as

$$F_m(x) \;=\; F_{m-1}(x) \;+\; \eta\, h_m(x),$$

where $\eta \in (0, 1]$ is a learning rate controlling the contribution of each stage. Over multiple iterations, `GB` gradually reduces the loss by directing new learners along the steepest descent direction in function space (for more detials, see [22]). In BBO, `GB` can be used as a surrogate model for non-linear objectives, especially in structured or tabular domains where tree ensembles (Random Forests, `GB`, XGBoost) often outperform neural surrogates

Modules in `B2Opt`. `B2Opt` employs specialized neural modules inspired by genetic operators:

- **Self-Attention Crossover (`SAC`):** Given a population embedding matrix $X \in \mathbb{R}^{n \times d}$, query, key, and value projections $Q, K, V \in \mathbb{R}^{n \times d}$ are computed as linear transforms of X. The attention-based crossover is

$$\texttt{SAC}(X) \;=\; \text{softmax}\Big(\tfrac{QK^\top}{\sqrt{d}}\Big)V,$$

  which recombines information across individuals, analogous to crossover in evolutionary algorithms but guided by learned attention weights.

- **Feed-Forward Mutation (`FM`):** Each candidate $x \in \mathbb{R}^d$ is perturbed through a feed-forward network:

$$\tilde{x} = W_2\, \sigma(W_1 x + b_1) + b_2,$$

  where W_1, W_2 are weight matrices, b_1, b_2 are biases, and $\sigma(\cdot)$ is a nonlinear activation (e.g., `ReLU`). This introduces nonlinear mutations beyond simple random perturbations.

- **Residual Selection Module (`RSSM`):** To preserve elite solutions, the residual selection mechanism interpolates between the mutated candidate $\tilde{x}$ and the original x:

$$x' \;=\; \alpha\tilde{x} + (1-\alpha)x,$$

  where $\alpha \in [0,1]$ is a learnable gating parameter. This balances exploration (through mutation) and exploitation (retaining high-quality individuals).

Classical DFO methods (line search, direct search, surrogate-based search) form the backbone of BBO. However, in challenging settings with combinatorial structures or mixed domains, their efficiency and robustness are limited by the absence of gradients and the exponential growth of the search space. ML offers complementary mechanisms to address these challenges. Below, we outline four main roles of ML in BBO and present representative formulations.

(i) Surrogate modeling. Let $f : \Omega \to \mathbb{R}$ with $\Omega \subseteq \mathbb{R}^d$ be the black-box objective. A surrogate $\hat{f}_\theta(x)$ is trained on data $D = \{(x_i, f(x_i))\}_{i=1}^N$ to approximate $f(x)$. Classical `BO` uses `GP` surrogates with posterior mean $\mu(x)$ and variance $\sigma^2(x)$. More general ML surrogates, such as Random Forests or `ReLU` neural networks (see definitions above), can capture non-smooth or categorical structures.

Examples. `mlrMBO` [5] generalizes sequential model-based optimization, iteratively updating a surrogate $\hat{f}_t$ and solving an acquisition function (e.g., `EI` or `UCB`, see above) to propose candidates. `SPBOpt` [67] partitions the domain into subregions, runs local `BO` in each region, and refines partitions adaptively,

which makes it particularly effective in low-budget scenarios. DFO-TR [25] applies model-based trust-region search to ML objectives such as AUC or SVM hyperparameter tuning (see above). Together, these approaches extend the classical Gaussian process paradigm by handling combinatorial constraints, partitioning the domain, or applying local regression in trust regions.

(ii) ML-inspired optimizers. Gradient-based optimizers from deep learning can be adapted to gradient-free contexts. Chen et al. [9] propose zeroth-order AdaMM, which replaces analytic gradients with zeroth-order estimates and applies Adam-style adaptive updates. Several ZO methods have already been defined in Subsection 3.5.

(iii) Meta-learning and portfolios. When optimizing across a distribution of tasks $\mathcal{T}$, meta-black-box optimization (MetaBBO) seeks to minimize the expected regret

$$\min_{\pi} \ \mathbb{E}_{T\sim\mathcal{T}}\big[f_T(x_\pi) - f_T(x^\star)\big],$$

where π is a meta-policy, x_π is the solution proposed by π for task T, and $x^\star$ is the task-specific global optimum. The goal is to learn strategies that generalize across tasks rather than optimizing each problem instance from scratch.

The Automated Black-Box Optimizer (ABBO) [54] leverages large-scale benchmarking to select or chain optimizers $\alpha \in \mathcal{A}$ from a portfolio, based on task meta-features (e.g., dimension, variable types) or short exploratory runs. The Distributed Black-Box Optimization (DiBB) framework [14] takes a structural approach: the decision vector is partitioned into blocks $\{B_j\}$, each block is optimized by a dedicated solver π_j, and the block-wise solutions are combined into global candidates $x = (x^{(1)}, \ldots, x^{(k)})$.

Together, these methods illustrate two complementary meta-learning strategies: solver selection at the portfolio level (ABBO) and problem decomposition at the structural level (DiBB).

(iv) Generative modeling. Generative models directly learn a distribution $p_\theta(x)$ over promising solutions, where θ denotes learnable parameters of the generative network. This approach replaces acquisition optimization with direct sampling from a learned search distribution.

B2Opt [45] employs a Transformer architecture with specialized modules for crossover, mutation, and selection (see SAC, FM, RSSM definitions above), which evolve a population of candidates by iteratively transforming their embeddings.

Diffusion-based BBO (DiffBBO) [46] learns conditional densities $p_\theta(x \mid r)$, where r is a reward signal or pseudo-label indicating solution quality. New candidates

are then generated by iterative denoising:

$$x_T \sim \mathcal{N}(0, I), \qquad x_{t-1} = x_t - \alpha_t \nabla_x \log p_\theta(x_t \mid r),$$

where x_T is an initial Gaussian noise sample, $\alpha_t > 0$ is a time-dependent step size, and $\nabla_x \log p_\theta(x_t \mid r)$ is the score function guiding the reverse diffusion.

These generative approaches represent a shift from classical surrogate-based `BO`, where the acquisition function must be optimized in an inner loop, to direct modeling of the distribution of good solutions. By learning $p_\theta(x)$ or $p_\theta(x \mid r)$, they enable efficient, data-driven search and often outperform acquisition-based methods in low-budget or offline settings.

4.1.2 `mlrMBO` – Modular Model-Based Optimization

The Modular Model-Based Optimization (`mlrMBO`) framework [5] is a general-purpose implementation of surrogate-based optimization, also known as Sequential Model-Based Optimization (`SMBO`). `BO` is a prominent special case within this broader `SMBO` paradigm. `mlrMBO` was designed for real-world settings with mixed-variable domains (continuous, integer, categorical, and conditional parameters), multi-objective tasks, and parallel batch evaluations.

The ML contribution of `mlrMBO` lies in its highly flexible surrogate modeling:

- Any regression learner from the modular machine learning toolbox can serve as the surrogate model (Random Forests, `GP`, `GB`, etc.).
- Random Forests are often used for heterogeneous domains (continuous and categorical), as they naturally handle non-continuous inputs without explicit embeddings.
- `GP`s with custom kernels (`RBF`, categorical, conditional) are supported for continuous and smooth problems.
- The modular design allows swapping out the surrogate, acquisition function, or optimizer depending on problem characteristics.

This surrogate flexibility makes `mlrMBO` robust across diverse BBO problems, from engineering simulations to hyperparameter tuning and algorithm configuration.

Main features of `mlrMBO` include:

- **Mixed-domain support:** handling continuous, integer, categorical, and hierarchical (conditional) parameters.

- **Batch proposals:** generating multiple candidate points per iteration, enabling efficient parallelization.
- **Multi-objective optimization:** integrating dominance-based and scalarization-based acquisitions.
- **Error handling:** supporting noisy evaluations, failed runs, or missing data.
- **Visualization and logging:** tracking optimization trajectories with integrated ML tooling.

Algorithm 6 proceeds through a sequence of steps ($S0_6$)–($S6_6$). In the **initialization step** ($S0_6$), an initial design D_0 is generated with a space-filling method such as Latin Hypercube or Sobol sampling, and a first surrogate model (e.g., a Random Forest or `GP`) is fitted to approximate the expensive black-box function. The best solution found so far x_{best} is initialized from these evaluations. In the **surrogate update step** ($S1_6$), the surrogate is retrained with all data collected up to the current iteration. Next, in the **acquisition design step** ($S2_6$), an acquisition function such as `EI` or `UCB` is defined to balance exploration and exploitation. The **inner optimization step** ($S3_6$) then seeks promising candidates by optimizing the acquisition function over the mixed-variable domain, using solvers capable of handling categorical or conditional variables. The candidate solution x_{trial} is subsequently tested in the **evaluation step** ($S4_6$) on the expensive objective function. This new observation is added to the dataset in the **augmentation step** ($S5_6$). Finally, in the **best point update step** ($S6_6$), the algorithm checks whether the new evaluation improves on the incumbent best solution, and updates x_{best} accordingly. This loop is repeated until a stopping criterion, such as convergence or evaluation budget, is met.

`mlrMBO` has been successfully applied to:

- hyperparameter optimization in ML pipelines (`SVMs`, `NNs`, ensembles),
- multi-objective tuning of runtime–accuracy trade-offs in solvers,
- algorithm configuration for combinatorial optimization problems,
- engineering design with expensive simulation-based evaluations.

4.1.3 `ZO-AdaMM` – Zeroth-Order Adaptive Momentum Method

A prominent example of how ideas from **ML optimizers** can enhance black-box optimization is the Zeroth-Order Adaptive Momentum Method (`ZO-AdaMM`) by Chen et al. [9]. `ZO-AdaMM` transfers `Adam`/`AMSGrad`-style **adaptive moment estimation** to the gradient-free setting by replacing analytic gradients with

Algorithm 6 `mlrMBO`**, Modular Model-Based Optimization**

Initialization: ($S0_6$) Generate an initial design D_0 using a space-filling method (e.g., Latin Hypercube, Sobol sequence). Fit surrogate model $\hat{f}_0(x)$ using a chosen ML regression learner (e.g., Random Forest or `GP`). Set best solution $x_{\text{best}} = \text{argmin}\, f(x_i)$.

repeat

Surrogate Update: ($S1_6$) Retrain surrogate $\hat{f}_t$ on all data D_t.

Acquisition Design: ($S2_6$) Define acquisition function $a(x|\hat{f}_t)$ (e.g., `EI`, `UCB`, `PI`).

Inner Optimization: ($S3_6$) Optimize $a(x)$ over the mixed domain Ω using optimizers aware of categorical/conditional variables (e.g., evolutionary search, mixed-integer local search). Obtain candidate x_{trial}.

Evaluating f: ($S4_6$) Compute $f(x_{\text{trial}})$ on the expensive black-box.

Augmentation: ($S5_6$) Add $(x_{\text{trial}}, f(x_{\text{trial}}))$ to dataset D_t.

Updating Best Point: ($S6_6$) If $f(x_{\text{trial}}) < f_{\text{best}}$, update $x_{\text{best}} = x_{\text{trial}}$.

until stopping criterion (evaluation budget, convergence).

randomized zeroth-order (`ZO`) estimates and using an `AMSGrad` second-moment cap together with a **Mahalanobis** projection for constraints.

This is motivated by black-box ML tasks (adversarial example generation, RL, hyperparameter tuning) where gradients are unavailable or unreliable. Classical `ZO` methods (e.g., `ZO-SGD`, coordinate descent) use fixed steps and suffer high variance in high dimensions. By importing adaptive moments (m_t, v_t), `ZO-AdaMM` improves robustness and stability.

Key enhancements:
• Random-direction gradient surrogates are built from function queries at perturbed inputs (no analytic gradients).
• `Adam`/`AMSGrad`-style moving averages (m_t, v_t) with a max-capped second moment reduce variance and enable adaptive scaling.
• For constraints, Mahalanobis-distance projections are used; Euclidean projections can fail to converge.
• Theory shows rates that are roughly $O(\sqrt{d})$ worse (in d-dependence) than first-order adaptive methods, which is expected in `ZO` settings.

Empirically (e.g., on ImageNet attacks, both per-image and universal), `ZO-AdaMM` converges faster and attains higher success rates than `ZO` baselines (`ZO-SGD`, `ZO-signSGD`, `ZO-SCD`), exhibiting the practical value of bringing deep-learning optimizer design into gradient-free ML.

Algorithm 7 follows the same adaptive moment estimation principles as `Adam`, but replaces analytic gradients with stochastic `ZO` estimates. In the **initializa-**

tion step (S0$_7$), an initial point, learning rate, and smoothing parameter are set. At each iteration, the algorithm first performs **gradient estimation** (S1$_7$) by querying the black-box function at randomly perturbed inputs to construct a one-point estimator of the gradient. These noisy estimates are then smoothed using an **adaptive momentum update** (S2$_7$), where moving averages of the first and second moments (m_t, v_t) are updated in the same way as `AMSGrad`. Next, the **adaptive step** (S3$_7$) computes the update direction by scaling m_t with $\sqrt{v_t}$, yielding stable progress even in high dimensions. If the optimization is constrained, a final **projection step** (S4$_7$) maps the iterate back into the feasible set using a Mahalanobis-distance projection, which ensures convergence where naive Euclidean projection can fail. This loop repeats until convergence, effectively transferring the robustness of `Adam`-type optimizers to the gradient-free black-box setting.

Algorithm 7 `ZO-AdaMM`**, Zeroth-Order Adaptive Momentum Method**

Initialization: (S0$_7$) Initialize x_0, step sizes $\{\alpha_t\}$, momentum parameters $\{\beta_{1,t}\}$, β_2, and set $m_0 = 0$, $v_0 = 0$, $\hat{v}_0 = 0$.

repeat

Gradient Estimation: (S1$_7$) Compute stochastic zeroth-order gradient estimate $\hat{g}_t$ at x_t using random perturbations $u \sim \mathrm{Unif}(\mathbb{S}^{d-1})$:

$$\hat{g}_t \;=\; \tfrac{d}{\mu}\big(f(x_t + \mu u) - f(x_t)\big)u$$

Momentum Update: (S2$_7$) Update moving averages:

$$m_t \;=\; \beta_{1,t} m_{t-1} + (1 - \beta_{1,t})\,\hat{g}_t,$$

$$v_t \;=\; \beta_2 v_{t-1} + (1 - \beta_2)\,(\hat{g}_t \odot \hat{g}_t).$$

Here $\odot$ denotes element-wise product.

Adaptive Step: (S3$_7$) Maintain $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$ and update:

$$x_{t+1} \;=\; \Pi_{\mathcal{X}, \sqrt{\hat{V}_t}}\Big(x_t - \alpha_t\, \hat{V}_t^{-1/2} m_t\Big)\,,$$

where $\hat{V}_t = \mathrm{diag}(\hat{v}_t)$ and $\Pi_{\mathcal{X},H}(\cdot)$ denotes projection onto $\mathcal{X}$ under Mahalanobis norm induced by H.

until convergence criterion is met

While Algorithm 7 is defined with a one-point estimator using random directions $u \sim \mathrm{Unif}(\mathbb{S}^{d-1})$, in practice variants sometimes use Gaussian directions $u \sim \mathcal{N}(0, I)$ or a symmetric two-point estimator for reduced bias. These alternatives yield similar updates but are not part of the official `ZO-AdaMM` algorithm as proposed by Chen et al. [9].

4.1.4 ABBO – Algorithm Selection Wizard for BBO

BBO covers a broad class of problems, from continuous functions to mixed-integer, noisy, multi-objective, or dynamic problems. No single optimizer (CMA-ES, BO, ES, etc.) performs best across all these scenarios — this is known as the **no free lunch theorem**.

To address this, Meunier et al. [54] propose the **Automated Black-Box Optimizer (ABBO)**, implemented in the Nevergrad platform (also called NGOpt). ABBO is a meta-algorithm that leverages massive benchmarking data to select, combine, or adapt optimizers during a run.

ABBO uses data-driven insights from benchmarking to guide algorithm choice and parameter control:

- **Passive algorithm selection:** Based on simple problem meta-features (dimension, variable types, noise, parallelism, budget), ABBO chooses optimizers that have shown strong performance in similar settings.
- **Active testing (bet-and-run):** ABBO can quickly try multiple algorithms on the target problem for a few iterations, then continue the most promising one — similar to an exploration step.
- **Chaining:** ABBO can switch between algorithms during a run (e.g., random search at the start, then CMA-ES for refinement), using rules derived from large-scale benchmarking.

In this way, ABBO relies on benchmarking-informed algorithm selection and adaptive portfolios, instead of expert intuition or static heuristics.

Algorithm 8 operates as an automated wizard that chooses and adapts optimizers to the characteristics of a given BBO problem. In the **problem characterization step** ($S0_8$), basic meta-features such as dimensionality, variable types, evaluation budget, noise, and parallelism are extracted. Based on these descriptors, the **passive selection step** ($S1_8$) identifies a set of promising optimizers from benchmarking knowledge. To reduce risk, ABBO can then perform an **active selection step** ($S2_8$), running each candidate briefly (bet-and-run) and keeping only the best-performing one. In the **selection/chaining step** ($S3_8$), ABBO either fixes on the strongest optimizer or designs a sequence of optimizers (for instance, starting with random search and switching to CMA-ES for refinement). The chosen optimizer(s) are then executed in the **main optimization step** ($S4_8$) with the remaining evaluation budget. Finally, in the **update step** ($S5_8$), the wizard tracks the incumbent best solution throughout the run. This adaptive process enables ABBO to automatically tailor its search strategy to a wide range of problem classes, achieving strong performance without manual tuning.

Algorithm 8 `ABBO`**, Algorithm Selection Wizard for BBO**

Problem characterization: ($S0_8$) Extract meta-features: dimension d, variable types (continuous, integer, categorical), evaluation budget B, presence of noise, and degree of parallelism p.
Passive selection: ($S1_8$) From benchmarking knowledge, choose candidate optimizers likely to fit the problem class.
Active selection (bet-and-run): ($S2_8$) Optionally run each candidate optimizer for a short budget $b \ll B$ to empirically assess performance.
Selection/Chaining: ($S3_8$) Choose the best optimizer or design a sequence of optimizers (chaining strategy).
Run optimization: ($S4_8$) Allocate remaining budget $B - b$ to the chosen optimizer(s).
Update best: ($S5_8$) Track $x_{\text{best}}, f_{\text{best}}$ during execution.

`ABBO` has been shown to:

- match or outperform state-of-the-art optimizers on `COCO`, `Pyomo`, `Photonics`, `LSGO`, and `MuJoCo` benchmarks,
- automatically select between `CMA-ES`, `BO`, `DF`, `PSO`, or hybrid methods,
- generalize across hundreds of heterogeneous problems without problem-specific tuning.

4.1.5 `DFO-TR` – BBO in ML with Trust-Region DFO

Ghanbari and Scheinberg [25] study the application of the **Derivative-Free Optimization with Trust Regions (`DFO-TR`)** algorithm to ML tasks where explicit gradients are unavailable or unreliable. Their motivating examples include optimizing non-smooth objectives such as `AUC` and hyperparameter tuning of ML models.

`DFO-TR` belongs to the class of model-based trust-region algorithms:

- At each iteration, a local surrogate model $m(x)$ is constructed within a trust region centered at the current iterate.
- The surrogate is typically a quadratic regression model, fitted to past function evaluations (via regression rather than strict interpolation) in the trust region.
- The surrogate is optimized (approximately) within the trust region to generate a trial point x_{trial}.

- The trial point is then evaluated on the true black-box function $f(x)$, and the trust-region radius is adjusted depending on the agreement between surrogate prediction and actual improvement.

This approach is particularly well suited for ML optimization tasks because:

- It can optimize noisy and non-differentiable objectives such as `AUC` or validation error, where gradients are undefined.
- It reuses past evaluations efficiently, unlike purely random or grid-based search.
- It requires relatively few function evaluations, which is crucial when training ML models is computationally expensive.

Empirical results show that `DFO-TR` can efficiently maximize `AUC` in classification tasks (e.g., linear classifiers, `SVMs`) without requiring gradients, and can tune hyperparameters (regularization, kernel width) with performance competitive to `BO`, random search, and gradient-based heuristics.

Algorithm 9 applies a trust-region strategy to machine learning BBO problems where gradients are unavailable. In the **initialization step** ($S0_9$), a starting point and an initial trust-region radius are chosen. At each iteration, the **surrogate fitting step** ($S1_9$) builds a local quadratic regression model from function values near the current point. The surrogate is then optimized inside the trust region in the **candidate generation step** ($S2_9$), producing a trial solution x_{trial}. This candidate is evaluated on the true black-box function in the **evaluation step** ($S3_9$). The **trust-region update step** ($S4_9$) compares the surrogate's predicted improvement against the actual improvement: if the agreement is good, the trial point is accepted and the trust-region radius is expanded; if poor, the trial point is rejected and the radius is contracted. The incumbent best solution is updated in ($S5_9$). This loop repeats until the evaluation budget is used up or convergence is detected ($S6_9$). By balancing local surrogate modeling with adaptive trust-region control, `DFO-TR` efficiently searches expensive, noisy, and non-smooth ML objectives such as `AUC` or validation error.

Algorithm 9 `DFO-TR`**, BBO in ML with Trust-Region DFO**

Initialization: (S0$_9$) Initialize starting point x_0 and trust-region radius Δ_0.
Surrogate fitting: (S1$_9$) Fit a local quadratic surrogate model $m_t(x)$ using evaluated points within the current trust-region.
Candidate generation: (S2$_9$) Compute candidate point

$$x_{\text{trial}} = \underset{x \in B(x_t, \Delta_t)}{\operatorname{argmin}} \; m_t(x).$$

Evaluation: (S3$_9$) Evaluate the true objective at the trial point: $f_{\text{trial}} = f(x_{\text{trial}})$.
Trust-region update: (S4$_9$) Compare predicted vs. actual improvement:

- If agreement is good: accept x_{trial} and expand Δ_t.
- If agreement is poor: reject x_{trial} and shrink Δ_t.

Best solution update: (S5$_9$) Update x_{best} if f_{trial} yields an objective improvement (e.g., higher `AUC` for maximization tasks or lower error for minimization tasks).
Termination: (S6$_9$) Repeat steps (S1$_9$)–(S5$_9$) until evaluation budget is exhausted or convergence is reached.

4.1.6 `SPBOpt` – Solving BBO via Learning Search Space Partition

Sazanovich et al. [67] propose the **Search Partition for Bayesian Optimization (`SPBOpt`)** algorithm, developed during the NeurIPS 2020 BBO Challenge. `SPBOpt` addresses the challenge of **low-budget optimization**, where only a small number of function evaluations is allowed. The key idea is to partition the search space into regions and apply local `BO` within each region, with the partitioning refined adaptively as evaluations proceed.

The main workflow of `SPBOpt` is:

- Partition the domain Ω into multiple subregions.
- For each region, run a local `BO` routine to propose candidate points.
- Merge all regional candidates and evaluate them on the true objective.
- Adaptively update the partitioning using feedback on performance, reinforcing promising areas while discarding poor ones.

This approach is particularly effective in competition and benchmark settings because:

- Local `BO` is sample-efficient, crucial when the evaluation budget is very limited.
- Partitioning balances exploration and exploitation by distributing queries across regions.
- Adaptive refinement of partitions enables focus on promising subspaces as evidence accumulates.

`SPBOpt` ranked third in the NeurIPS 2020 BBO Challenge finals, demonstrating strong empirical performance under strict evaluation budgets.

Algorithm 10 is designed for low-budget BBO by combining partitioning with local `BO`. This algorithm runs for K iterations, proposing B candidate points per iteration (as in the NeurIPS 2020 BBO Challenge, $K = 16$ and $B = 8$, giving a total budget of $K \times B = 128$ evaluations). In the **partitioning step** ($\mathrm{S0}_{10}$), the domain Ω is divided into several subregions, which allows the search to cover the space systematically. Each region is then explored independently in the **local optimization step** ($\mathrm{S1}_{10}$), where a `BO` routine proposes candidate points based on a surrogate model fit to the evaluations inside that region. All proposed candidates are aggregated and tested on the true objective in the **candidate evaluation step** ($\mathrm{S2}_{10}$). The results are used to adapt the space partitioning in the **partition update step** ($\mathrm{S3}_{10}$), reinforcing promising regions with finer

granularity and de-emphasizing or discarding poor regions. This loop repeats until the evaluation budget is exhausted ($S4_{10}$). By combining global coverage through partitions with sample-efficient local `BO`, `SPBOpt` balances exploration and exploitation and quickly concentrates evaluations in promising areas, making it effective under tight evaluation limits.

Algorithm 10 `SPBOpt`**, Solving BBO via Learning Search Space Partition**

Partitioning: ($S0_{10}$) Divide the search space Ω into multiple subregions.

Local optimization: ($S1_{10}$) Within each subregion, run a local `BO` routine to propose candidate points.

Candidate evaluation: ($S2_{10}$) Evaluate the proposed candidates from the selected region(s) on the true objective $f(x)$.

Partition update: ($S3_{10}$) Refine partitions based on observed performance feedback.

Termination: ($S4_{10}$) Repeat until the total evaluation budget (e.g., $K \times B$ evaluations in the challenge) is exhausted.

Empirical results reported by Sazanovich et al. [67] show that:

- `SPBOpt` ranked third overall in the NeurIPS 2020 BBO Challenge finals, outperforming many strong baselines under a strict 128-evaluation budget.
- The partition-and-local-BO strategy was especially effective in low-dimensional and highly multi-modal problems, where global coverage combined with adaptive refinement allowed rapid focus on promising regions.
- Compared to vanilla `BO`, `SPBOpt` achieved higher solution quality within the same limited budget by balancing exploration across partitions and exploitation within each subregion.

4.1.7 `B2Opt` – Learning to Optimize BBO with Little Budget

Li et al. [45] propose `B2Opt`, a deep learning framework for BBO under extremely limited evaluation budgets. `B2Opt` builds on ideas from genetic algorithms and Transformers, learning parameterized optimization strategies that map random initial populations to near-optimal solutions with few queries.

The main workflow is:

- Encode candidate populations into a transformer-based network.

- Within each `B2Opt` **Block (OB)**, apply three modules: a Self-Attention-based Crossover module (`SAC`), a Feed-Forward Mutation module (`FM`), and a Residual Selection module (`RSSM`).
- Stack OBs sequentially to update the population across iterations.
- Train `B2Opt` on cheap surrogate problems to approximate target BBO tasks.
- Deploy the learned optimizer on expensive black-box functions.

Algorithm 11 illustrates the workflow of the `B2Opt` framework. In the **initialization step** ($S0_{11}$), a random population of candidate solutions is generated. These candidates are then processed in the **encoding step** ($S1_{11}$), where the population is passed through one or more OBs composed of `SAC`, `FM`, and `RSSM`. The resulting new population X_{t+1} is produced in the **population update step** ($S2_{11}$). To make the optimizer effective under limited evaluations, the framework undergoes offline training in the **learning step** ($S3_{11}$), where it learns optimization strategies on surrogate tasks using gradient-based methods such as `SGD` or `Adam`. Finally, in the **deployment step** ($S4_{11}$), the trained optimizer is applied directly to expensive black-box problems, enabling it to generate near-optimal solutions with only a few queries. By learning optimization strategies offline and transferring them, `B2Opt` reduces the need for costly evaluations and outperforms many hand-designed algorithms in low-budget scenarios.

Algorithm 11 `B2Opt`, Learning to Optimize BBO with Little Budget

($S0_{11}$) Initialize random population X_0.
($S1_{11}$) Update X_t via `B2Opt` Block(s) (`SAC`, `FM`, `RSSM`) to produce new representations.
($S2_{11}$) Decode the updated representations to form the next population X_{t+1}.
($S3_{11}$) Train network parameters on surrogate tasks using `SGD`/`Adam`.
($S4_{11}$) Deploy trained optimizer to expensive BBO tasks.

`B2Opt` achieves **state-of-the-art performance** in strict low-budget regimes:

- On the standard BBOB benchmarks, `B2Opt` outperforms strong baselines including `DE`, `CMA-ES`, and `BO`-based methods.
- In 10-dimensional tasks, `B2Opt` leads on 20 out of 24 `BBOB` functions; in 100 dimensions, it leads on 19 out of 24 functions, showing strong scalability.
- Compared to other learning-to-optimize frameworks (`LGA`, `LES`, `L2O-swarm`), `B2Opt` consistently achieves higher solution quality within the same evaluation budget.
- On real tasks such as robotics control and NN training, `B2Opt` finds high-quality solutions with only tens of queries, whereas classical baselines often stagnate under such tight budgets.

4.1.8 `DiffBBO` – Diffusion Model for Data-Driven BBO

Li et al. [46] cast offline BBO as conditional sampling with diffusion models: learn $p(x \mid r)$ from mixed labeled/unlabeled data via reward (or preference) modeling and pseudo-labeling; then generate near-optimal x by conditioning on high reward. They provide sub-optimality bounds close to off-policy bandits and show latent subspace fidelity.

Algorithm 12 frames offline BBO as conditional generation with diffusion models guided by rewards. We call this algorithm `DiffBBO`. In the **initialization step** ($S0_{12}$), a reward or preference model is trained on the labeled data, and pseudo-labels $\hat{r}(x)$ are assigned to the unlabeled pool. Next, in the **diffusion training step** ($S1_{12}$), a conditional diffusion model $p_\theta(x \mid r)$ is learned to approximate the distribution of candidate solutions given reward signals. At the **acquisition design step** ($S2_{12}$), a high reward target $r^\dagger$ is set, and a denoising generation schedule is defined. The **generation step** ($S3_{12}$) then samples new candidates x_{trial} by conditioning the diffusion process on $r^\dagger$, optionally reranking samples using the learned reward model $\hat{r}(x)$. In the **evaluation step** ($S4_{12}$), the surrogate reward $\hat{r}(x)$ is used for selection; in semi-offline cases the true black-box $f(x)$ may also be queried. The buffer is expanded in the **augmentation step** ($S5_{12}$), and both the reward and diffusion models can be refined with the new data. Finally, in the **update step** ($S6_{12}$), the incumbent best solution is updated whenever an improved candidate is found. This loop repeats until stopping criteria are met, enabling the diffusion model to generate near-optimal candidates from offline data while providing sub-optimality guarantees close to those in off-policy bandits.

Empirical validation of `DiffBBO` shows:

- On synthetic benchmarks, it achieves solution quality competitive with off-policy bandit algorithms under the same offline data.
- On high-dimensional problems (up to 100D), the method preserves fidelity in latent subspaces, enabling effective sampling in lower-dimensional manifolds.
- In real-world tasks such as hyperparameter tuning and offline robotics control, `DiffBBO` outperforms strong baselines including `ES` and `BO` when only offline data are available.

4.1.9 `DiBB` – Distributed Partially-Separable BBO

Cuccu et al. [14] present `DiBB`, which turns a base BBO into a **partially separable** distributed algorithm by partitioning parameters into blocks (e.g., by

Algorithm 12 `DiffBBO`**, Diffusion Model for Data-Driven BBO**

Initialization: ($S0_{12}$) Fit reward/preference model on labeled set; pseudo-label unlabeled pool with $\hat{r}(x)$.

repeat

Diffusion Training: ($S1_{12}$) Train reward-conditioned diffusion (score model) $p_\theta(x \mid r)$ on pseudo-labeled data.

Acquisition Design: ($S2_{12}$) Set high reward conditioning $r^\dagger$; define denoising schedule.

Generation: ($S3_{12}$) Sample $x_{\text{trial}} \sim p_\theta(\cdot \mid r^\dagger)$ via denoising; optionally rerank by $\hat{r}(x)$.

Evaluation: ($S4_{12}$) Evaluate with $\hat{r}(x_{\text{trial}})$ (default offline); query $f(x_{\text{trial}})$ if feasible.

Augmentation: ($S5_{12}$) Add to buffer; refine reward and diffusion models if needed.

Updating Best Point: ($S6_{12}$) If the new candidate improves upon the current best (i.e., $\hat{r}(x_{\text{trial}}) < \hat{r}(x_{\text{best}})$ in minimization, or $\hat{r}(x_{\text{trial}}) > \hat{r}(x_{\text{best}})$ in maximization), update $x_{\text{best}} := x_{\text{trial}}$.

until stop.

neural layers) and running independent solver instances per block, while assembling/evaluating full candidates centrally. It preserves base-algorithm properties and scales in wall-clock time.

Algorithm 13 distributes BBO across parameter blocks to improve scalability while retaining the behavior of the base optimizer. In the **initialization step** ($S0_{13}$), the decision variables are partitioned into blocks $\{B_j\}$ (e.g., NN layers), and an instance of the chosen base solver is launched for each block, all sharing a central dataset D_0. During each iteration, the **surrogate update step** ($S1_{13}$) lets each block optimizer refine its local state or surrogate using shared feedback. In the **acquisition design step** ($S2_{13}$), each block proposes a local component of the candidate, which the coordinator assembles into a full solution x_{trial}. The **inner optimization step** ($S3_{13}$) involves running the base solver within each block, with synchronization ensuring consistent candidate construction. The **evaluation step** ($S4_{13}$) centrally computes the black-box objective $f(x_{\text{trial}})$. In the **augmentation step** ($S5_{13}$), the result $(x_{\text{trial}}, f(x_{\text{trial}}))$ is broadcast back to all blocks, updating the shared dataset. Finally, in the **best point update step** ($S6_{13}$), the global incumbent x_{best} is replaced if improvement is observed. This loop repeats until the budget is exhausted, allowing `DiBB` to preserve the theoretical properties of the base optimizer while achieving wall-clock speedups through distributed, partially separable optimization.

Empirical results reported by Cuccu et al. [14] show that:

Algorithm 13 `DiBB`, Distributed Partially-Separable BBO

Initialization: ($S0_{13}$) Partition variables into blocks $\{B_j\}$; instantiate base solver per block; initialize shared dataset D_0.

repeat

Surrogate Update: ($S1_{13}$) Each block updates its local model/state from shared feedback.

Acquisition Design: ($S2_{13}$) Each block proposes local component $x_{\text{trial}}^{(j)}$; the coordinator assembles full x_{trial}.

Inner Optimization: ($S3_{13}$) Run base inner optimization within each block; synchronize candidates.

Evaluating f: ($S4_{13}$) Compute $f(x_{\text{trial}})$ centrally.

Augmentation: ($S5_{13}$) Broadcast $(x_{\text{trial}}, f(x_{\text{trial}}))$ to all blocks; update shared D_{t+1}.

Updating Best Point: ($S6_{13}$) If the trial candidate improves upon the global incumbent (i.e., $f(x_{\text{trial}}) < f(x_{\text{best}})$ for minimization or $f(x_{\text{trial}}) > f(x_{\text{best}})$ for maximization), then set $x_{\text{best}} := x_{\text{trial}}$.

until budget.

- On `COCO/BBOB` benchmarks, `DiBB` achieves up to 5× wall-clock speed-ups compared to the base solvers while preserving their performance profiles.
- In neuroevolution tasks (e.g., `Walker2D` with over 11,000 weights), `DiBB` scales effectively by distributing parameters by neural layers, enabling efficient optimization of very high-dimensional controllers.
- The method consistently matches or exceeds the performance of the underlying optimizer while providing strong scalability through parallelization.

4.2 RL Enhancements in BBO

RL offers complementary strategies for enhancing BBO, particularly in settings where optimization must adapt online, handle stochasticity, or balance safety with exploration. Unlike static model-based approaches, RL formulates optimization as a sequential decision-making process, where an agent interacts with the black-box objective to configure operators, select candidate solutions, or adapt algorithmic behavior over time.

Contributions of RL to BBO can be grouped into four main categories:

- **Robustness and safety**: robust regression, log-barrier formulations, and KL-decoupled updates stabilize optimization under noise and constraints, as in `RBO` [10], `LB-SGD` [73], and `CAS-MORE` [34].
- **Dynamic operator configuration**: RL policies adapt evolutionary operators such as crossover and mutation during the search, exemplified by `Surr-RLDE` [52].
- **Policy search equivalence**: RL policy-gradient updates can be expressed as black-box search heuristics, revealing close ties to evolution strategies, as shown by `PI2/PIBB` [72].
- **Meta-RL for algorithm configuration**: learned controllers generalize across heterogeneous tasks, with offline training and benchmarks enabling zero-shot adaptation, as in `Q-Mamba` [50], supported by `MetaBox` [51] and `MetaBBO-RL` [76].

The following subsections review six representative methods that illustrate these roles: `Surr-RLDE`, `RBO`, `CAS-MORE`, `LB-SGD`, `PI2/PIBB`, and `Q-Mamba` (cf. Table 3).

Method	Core Idea	Domain	Refs.
`RBO`	Robust regression for noisy gradient estimates	Noisy BBO/RL	[10]
`LB-SGD`	Log-barrier `SGD` ensuring feasibility	Constrained BBO/RL	[73]
`CAS-MORE`	KL-decoupled mean/covariance and entropy control	Stochastic BBO / Episodic RL	[34]
`Surr-RLDE`	RL configures `DE` operators with surrogate pseudo-rewards	`MetaBBO`	[52]
`PI2/PIBB`	Reward-weighted policy updates	Evolutionary BBO / Policy search	[72]
`Q-Mamba`	Offline decomposed Q-learning and Mamba backbone	Large-action-space `MetaBBO`	[50]

Table 3: Representative RL-enhanced methods for BBO.

4.2.1 Background

RL provides a complementary perspective on BBO, casting search as a sequential decision-making process. To make this survey self-contained, we introduce the main RL-related concepts that recur in hybrid BBO frameworks. These include surrogate architectures such as the Kolmogorov–Arnold Network (`KAN`) and training criteria like the Order-Aware Loss (`OAL`); core algorithmic families such as Policy Optimization (`PO`), Meta-Policy Online training, and Dynamic Algorithm Configuration (`DAC`); scalable formulations including Decomposed Q-Functions (`DQF`); and representative RL-based `MetaBBO` approaches such as `DEDQN`, `GLEET`, and `SYMBOL`. We also recall common benchmarks and platforms for evaluation, including `COCO`, `Pyomo`, `Photonics`, `LSGO`, and `MuJoCo`. By defining these concepts upfront, we establish a foundation that will be referenced in the subsequent subsections on robustness, dynamic operator control, RL–policy search equivalence, and meta-RL for BBO.

Kolmogorov–Arnold Network (`KAN`). The Kolmogorov–Arnold superposition theorem states that any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ can be expressed as a finite superposition of univariate functions:

$$f(x_1, \ldots, x_d) \;=\; \sum_{q=1}^{2d+1} \Phi_q\!\left(\sum_{p=1}^{d} \phi_{q,p}(x_p) \right),$$

where $\{\phi_{q,p} : \mathbb{R} \to \mathbb{R}\}$ are inner functions applied to each input dimension, and $\{\Phi_q : \mathbb{R} \to \mathbb{R}\}$ are outer functions combining the results. A Kolmogorov–Arnold Network (`KAN`) is a neural architecture that parameterizes $\phi_{q,p}$ and Φ_q with learnable weights, effectively instantiating this constructive representation.

Because `KAN`s approximate multivariate functions using only sums of univariate functions, they can capture highly nonlinear landscapes with relatively few parameters and limited training data. This makes them attractive as surrogates in BBO, and they are employed in `Surr-RLDE` [52] to replace costly function evaluations during RL of evolutionary operators.

Order-Aware Loss. Surrogate training can enforce rank preservation via an order-aware loss:

$$\mathcal{L}_{\text{order}} \;=\; \sum_{i,j} \mathbf{1}[f(x_i) < f(x_j)]\, \ell\big(\hat{f}(x_i), \hat{f}(x_j)\big),$$

where $\ell(\hat{f}(x_i), \hat{f}(x_j))$ is a pairwise ranking loss that penalizes the surrogate $\hat{f}$ whenever it assigns a higher value to a truly worse point. Typical choices include the hinge ranking loss $\ell(u, v) = \max\{0, 1 - (u - v)\}$, the logistic loss

$\ell(u, v) = \log(1+\exp(-(u-v)))$, and the squared loss $\ell(u, v) = (u-v-1)^2$. These losses ensure that if $f(x_i) < f(x_j)$ (i.e., x_i is better), then the surrogate satisfies $\hat{f}(x_i) < \hat{f}(x_j)$, preserving the correct ordering of candidate solutions. This criterion is central in `Surr-RLDE`, where ranking consistency is more important than absolute value approximation.

Policy Optimization (`PO`). In RL, the goal is to maximize the expected return

$$J(\theta) \;=\; \mathbb{E}_{\tau\sim\pi_\theta}\big[R(\tau)\big],$$

where $\pi_\theta(a|s)$ is a parameterized policy with parameters θ, $\tau = (s_0, a_0, s_1, a_1, \dots)$ denotes a trajectory generated by π_θ and the environment dynamics, and

$$R(\tau) \;=\; \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

is the discounted cumulative reward with discount factor $\gamma \in [0, 1)$. **Policy optimization** (`PO`) methods directly adjust θ to improve $J(\theta)$. This can be done by gradient-based techniques such as `REINFORCE` (Monte Carlo policy gradients) or `PPO` (proximal policy optimization), or by gradient-free black-box search approaches such as `ES`.

Meta-Policy Online. A **meta-policy** π_ω is a higher-level controller that adapts the behavior of an optimizer during its run. Here ω are the meta-parameters of the controller, updated online as

$$\omega \;\leftarrow\; \omega + \eta\, \nabla_\omega\, \mathbb{E}_{T\sim\mathcal{T}}\left[\sum_t r_t\right],$$

where $\eta > 0$ is a learning rate, $\mathcal{T}$ is a distribution of tasks, and r_t is the reward observed at iteration t. While conceptually powerful, online training of π_ω is often sample-inefficient compared to offline `MetaBBO` approaches (e.g., `Q-Mamba`), which leverage pre-collected trajectories.

Dynamic Algorithm Configuration (`DAC`). `DAC` frames optimizer control as a Markov Decision Process (MDP) $\langle S, A, T, R\rangle$, where S is the state space, A the action space, $T(s'|s, a)$ the transition dynamics, and $R(s, a)$ the reward function. At time t, the optimizer's progress is summarized by a state $s_t \in S$ (e.g., population statistics or surrogate uncertainty), an action $a_t \in A$ adjusts solver parameters (e.g., step size, mutation rate), the transition kernel T describes the effect of this change on the optimizer's trajectory, and r_t is the immediate reward (e.g., improvement in objective value). This `MDP` view underpins meta-optimization in `Q-Mamba`.

Decomposed Q-Functions. Large action spaces in DAC can be intractable if a single monolithic state-action value function $Q(s, a)$ is used. A decomposed formulation instead splits the action vector $a = (a^1, \ldots, a^d)$ into components, approximating

$$Q(s, a) \;=\; \sum_{k=1}^{d} Q^k(s, a^k),$$

where Q^k is a dimension-specific Q-function. This decomposition reduces complexity and enables tractable argmax action selection in high-dimensional DAC problems, and is employed in Q-Mamba.

RL-based MetaBBO Variants. Several hybrid frameworks adapt evolutionary optimizers using RL: DEDQN configures DE operators via deep Q-learning; GLEET learns transferable policies that select evolutionary operators across multiple tasks; and SYMBOL discovers symbolic mutation and crossover operators guided by RL. These approaches illustrate how RL can automate operator selection and generalize evolutionary strategies beyond fixed heuristics.

Benchmarks and Platforms. RL-enhanced BBO methods are often evaluated on standardized benchmarks:

- COCO: Comparing Continuous Optimizers, standard BBO test suite.
- Pyomo: algebraic modeling and optimization in Python.
- Photonics: photonic circuit design tasks.
- LSGO: Large-Scale Global Optimization problems ($d \geq 1000$).
- MuJoCo: continuous-control physics simulator for RL.

While DFO methods (line search, direct search, surrogate-based search) form the backbone of BBO, recent developments in RL have introduced new ways of improving their efficiency and robustness. We group the roles of RL in BBO into four categories:

(i) Robustness and safety. RL-inspired black-box optimizers often need to operate reliably in noisy or constrained domains.

The Robust Black-box Optimizer (RBO) [10] estimates gradients via robust regression. Given perturbation vectors $\{g_i\}_{i=1}^m$ and noisy function evaluations $y_i = f(\theta + g_i) + \epsilon_i$, where ϵ_i denotes noise, the gradient is recovered as

$$\hat{\nabla} f(\theta) \;=\; \arg\min_{v \in \mathbb{R}^d} \sum_{i=1}^{m} \rho\big(y_i - f(\theta) - g_i^\top v\big),$$

with $\rho(\cdot)$ a robust loss function (e.g., Huber, L_1). This procedure tolerates adversarial or heavy-tailed noise and enables stable updates.

Log-Barrier Stochastic Gradient Descent (`LB-SGD`) [73] ensures feasibility by embedding constraints into a log-barrier augmented objective

$$\tilde{f}(\theta) \;=\; f(\theta) - \mu \sum_{j=1}^{p} \log(-h_j(\theta)),$$

where $\{h_j(\theta)\}_{j=1}^{p} \leq 0$ are inequality constraints, and $\mu > 0$ is a barrier parameter controlling strictness. Stochastic gradient descent on $\tilde{f}(\theta)$ ensures that all iterates remain feasible.

Coordinate-Ascent Model-based Relative Entropy Search (`CAS-MORE`) [34] stabilizes Gaussian search distributions by separately constraining updates of the mean μ_t and covariance Σ_t via Kullback–Leibler (KL) divergences:

$$\mathrm{KL}(\mu_{t+1} \| \mu_t) \;\leq\; \epsilon_\mu, \qquad \mathrm{KL}(\Sigma_{t+1} \| \Sigma_t) \;\leq\; \epsilon_\Sigma,$$

where $\epsilon_\mu, \epsilon_\Sigma > 0$ are trust parameters. Decoupling exploitation (mean shift) from exploration (covariance adaptation) yields more robust optimization in stochastic or high-variance environments.

(ii) Dynamic operator configuration. RL can adaptively control evolutionary operators during black-box search. The `Surr-RLDE` framework [52] combines a surrogate model $s(x)$ with an RL policy π_ϕ. At iteration t, the state s_t encodes population-level features such as diversity or fitness statistics. The policy $\pi_\phi(s_t)$ outputs operator parameters a_t (e.g., crossover rate, mutation factor), which govern the generation of trial solutions in `DE`. This enables operator parameters to be tuned online, rather than relying on fixed heuristics.

(iii) RL–policy search equivalence. Certain RL policy update rules can be interpreted as black-box search heuristics. Policy Improvement with Path Integrals (`PI2`) and Policy Improvement with Black-Box (`PIBB`) [72] both update policy parameters $\theta \in \mathbb{R}^d$ by reward-weighted averaging of perturbations:

$$\theta_{t+1} \;=\; \theta_t + \sum_{i=1}^{K} w_i \epsilon_i, \qquad w_i \;=\; \frac{R_i}{\sum_{j=1}^{K} R_j},$$

where $\{\epsilon_i\}_{i=1}^{K}$ are sampled perturbations of the policy, R_i are the corresponding returns, and w_i are normalized weights. This update is mathematically equivalent to rank-based evolution strategies such as `CMA-ES` or `NES`, establishing a bridge between RL and evolutionary BBO.

(iv) Meta-RL for BBO. RL can also be used at the meta-level, where a controller adapts optimizers online and generalizes across heterogeneous tasks. `Q-Mamba` [50] addresses large action spaces by discretizing continuous solver parameters into bins and applying decomposed Q-learning, where the state-action value function is factorized dimension-wise. Beyond single methods, standardized platforms enable systematic evaluation: `MetaBox` [51] provides a benchmark for meta-optimization across hundreds of black-box tasks, while `MetaBBO-RL` [76] introduces evaluation protocols and metrics (e.g., generalization decay, transfer efficiency) tailored to RL-based optimizers. These frameworks ensure fair comparison and reproducibility of meta-RL methods.

With these definitions and roles in place, the following subsections review representative RL-enhanced methods in more detail.

4.2.2 `Surr-RLDE` – RL-Configured `DE` with Surrogate Training

Ma et al. [52] propose the `Surr-RLDE` framework, which unifies surrogate learning with RL for Meta-Black-Box Optimization (`MetaBBO`). The central idea is to train a `KAN` surrogate with a relative-order-aware loss to preserve the ranking of candidate solutions. `KAN`s are chosen because they capture complex functional landscapes more effectively than standard neural surrogates in low-data regimes. This surrogate, trained per problem instance, replaces most costly black-box evaluations during policy training, allowing an RL agent to learn how to dynamically configure the mutation operators and parameters in `DE`.

The main workflow of `Surr-RLDE` is:

- Train a `KAN` surrogate with order-aware loss on an initial dataset.
- Use an RL policy π_ϕ to map population state features (e.g., diversity, fitness trends) to `DE` operator choices and parameter settings.
- Generate trial populations using `DE` with RL-configured parameters, evaluating most candidates via the surrogate and occasionally with the true objective.
- Provide pseudo-rewards based on surrogate ranking consistency (with occasional true rewards) to train the RL agent off-policy.
- Continuously update the surrogate as more true evaluations are observed.

This approach is particularly effective because:

- RL learns adaptive operator-selection policies that exploit search dynamics.

- Crossover and mutation rates are tuned online instead of using fixed heuristics.
- Surrogate-based pseudo-rewards drastically reduce the number of true function queries.

Extensive experiments show that `Surr-RLDE` significantly reduces evaluation cost while maintaining competitive performance. It generalizes to higher-dimensional problems, matching or outperforming recent `MetaBBO` baselines such as `DEDQN`, `GLEET`, and `SYMBOL` [51, 52, 70].

Algorithm 14 integrates surrogate learning with RL to dynamically configure evolutionary operators in `DE`. In the **initialization step** ($S0_{14}$), an initial population P_0 is sampled and evaluated with the true objective f, forming dataset D_0 and setting the incumbent best solution $(x_{\text{best}}, f_{\text{best}})$. A `KAN` surrogate $s(x)$ is trained with an order-aware loss to preserve the ranking of solutions, and an RL policy π_ϕ is initialized to map population-level features (e.g., diversity, fitness statistics) to `DE` strategy and parameter choices. At each iteration, the **operator configuration step** ($S1_{14}$) uses the policy to select crossover and mutation parameters. The **trial generation step** ($S2_{14}$) applies `DE` to form a trial population P^{trial}, which is primarily evaluated with the surrogate $s(\cdot)$, with occasional queries to the true $f(x)$. In the **reward computation step** ($S3_{14}$), pseudo-rewards are calculated from surrogate ranking consistency, supplemented by true rewards when available, and stored in a replay buffer. The **surrogate update step** ($S4_{14}$) adds any true evaluations to D and periodically retrains $s(x)$. The **policy update and selection step** ($S5_{14}$) forms the next generation P_{t+1} by `DE` selection, updates the incumbent $(x_{\text{best}}, f_{\text{best}})$ whenever a true evaluation yields improvement, and trains the RL policy π_ϕ off-policy using accumulated experience. This cycle repeats until the evaluation budget is exhausted ($S6_{14}$), enabling `Surr-RLDE` to adapt operator choices online while drastically reducing expensive queries, achieving competitive performance in `MetaBBO` benchmarks.

Algorithm 14 `Surr-RLDE`**, RL-Configured** `DE` **with Surrogate Training**

Initialization: ($S0_{14}$) Generate initial population $P_0 \subset \Omega$, evaluate with the true objective f to obtain dataset D_0. Set incumbent best $x_{\text{best}} = \text{argmin}_{x \in P_0} f(x)$ and $f_{\text{best}} = f(x_{\text{best}})$. Train surrogate $s(x)$ (`KAN` with order-aware loss). Initialize RL policy π_ϕ mapping population features of P_t to `DE` operator parameters.

repeat

Operator configuration: ($S1_{14}$) Use π_ϕ to select `DE` strategy (mutation, crossover) and parameters for P_t.

Trial generation: ($S2_{14}$) Generate trial population P^{trial} using `DE` with RL-configured operators. Evaluate candidates primarily with surrogate $s(x)$; query true $f(x)$ only occasionally within the evaluation budget.

Reward computation: ($S3_{14}$) Compute pseudo-rewards based on surrogate ranking consistency, supplemented by true rewards when available. Store transitions (s_t, a_t, r_t) in the replay buffer.

Surrogate update: ($S4_{14}$) When true evaluations $f(x)$ are obtained, add $(x, f(x))$ to dataset D and periodically retrain surrogate $s(x)$.

Policy update and selection: ($S5_{14}$) Form next generation P_{t+1} by `DE` selection: replace individuals in P_t with better candidates from P^{trial}. Update incumbent best: if any candidate evaluated with $f(x)$ satisfies $f(x) < f_{\text{best}}$, set $x_{\text{best}} := x$, $f_{\text{best}} := f(x)$. Train RL policy π_ϕ off-policy using stored experiences.

Termination: ($S6_{14}$) Repeat steps ($S1_{14}$)–($S5_{14}$) until the evaluation budget is exhausted.

until budget exhausted.

4.2.3 RBO – Provably Robust BBO for RL

Choromanski et al. [10] propose **Robust Black-box Optimization (RBO)**, a DFO method for RL that remains effective under adversarial or stochastic noise. RBO addresses the challenge of estimating reliable gradients when function evaluations are corrupted or unstable.

The main workflow of RBO is:

- Sample perturbation directions $\{g_i\}_{i=1}^m$.
- Evaluate the reward function at $\theta \pm g_i$, possibly with corrupted or noisy measurements.
- Formulate a robust regression problem to reconstruct an estimate of the local gradient field $\hat{\nabla} F(\theta)$.
- Update policy parameters with an off-policy gradient ascent step, reusing past samples.

Robust gradient estimation. Given noisy or corrupted measurements

$$y_i = F(\theta + g_i) + \epsilon_i,$$

RBO observes that the local linear approximation satisfies

$$y_i - F(\theta) \;\approx\; g_i^\top \nabla F(\theta), \qquad i = 1, \ldots, m.$$

Stacking all m perturbations gives

$$\mathbf{y} - F(\theta)\mathbf{1} \;=\; G\nabla F(\theta) + \epsilon,$$

where $G \in \mathbb{R}^{m \times d}$ is the perturbation matrix, $\mathbf{y} \in \mathbb{R}^m$ are observed rewards, and ϵ is noise.

RBO estimates the gradient by solving a robust regression (LP-decoding) problem:

$$\hat{\nabla F}(\theta) \;=\; \arg\min_{v \in \mathbb{R}^d} \sum_{i=1}^{m} \rho\big(y_i - F(\theta) - g_i^\top v\big)\,,$$

with $\rho(\cdot)$ a robust loss (e.g., Huber or L_1). From an error-correcting code perspective, this estimator can provably recover the gradient to high accuracy even if up to 23% of function evaluations are arbitrarily corrupted. Moreover, RBO reuses past perturbations to estimate an entire local gradient field, yielding continuous gradient-flow estimates and improving sample efficiency compared to standard ES methods.

Policy update. With the robust gradient estimate, policy parameters are updated by

$$\theta \leftarrow \theta + \eta \hat{\nabla} F(\theta),$$

where η is a learning rate. Importantly, `RBO` reuses past perturbations and their evaluations to form continuous gradient-flow estimates, improving sample efficiency in noisy settings.

This approach is particularly effective because:

- It provably tolerates nearly one quarter of function evaluations being arbitrarily corrupted.
- It reuses past samples to construct stable gradient flows, reducing sample complexity.
- It ensures reliable optimization in noisy RL environments where evolution strategies and other DFO methods often fail.

Experiments on `MuJoCo` robot control and quadruped locomotion confirm that `RBO` continues to train policies successfully under heavy noise, where other black-box methods fail.

Algorithm 15 implements Robust BBO (`RBO`), a derivative-free method designed to withstand corrupted or noisy evaluations in RL. In the **initialization step** ($S0_{15}$), policy parameters θ are set. The **perturbation sampling step** ($S1_{15}$) generates random search directions $\{g_i\}$. Each perturbation is tested in the **evaluation step** ($S2_{15}$) by computing the reward at $\theta \pm g_i$, even if some outcomes are corrupted or highly noisy. To obtain reliable search directions, the **gradient estimation step** ($S3_{15}$) formulates these measurements as a regression problem and solves it with robust techniques to reconstruct an estimate of the gradient $\hat{\nabla} F(\theta)$. The **update step** ($S4_{15}$) then adjusts policy parameters with a gradient ascent step, reusing past samples in an off-policy fashion to improve sample efficiency. This loop repeats until convergence or the evaluation budget is exhausted ($S5_{15}$). By tolerating up to 23% of arbitrarily corrupted evaluations and leveraging past experience, `RBO` provides provably stable learning in noisy RL environments where standard evolution strategies or DFO methods often fail.

4.2.4 `CAS-MORE` – Coordinate-Ascent Model-based Relative Entropy

Hüttenrauch and Neumann [34] present **Coordinate-Ascent MORE (`CAS-MORE`)**, an improved version of the Model-based Relative Entropy Stochastic Search (`MORE`) algorithm. `CAS-MORE` extends `MORE` to episodic RL with stochastic rewards, where ranking-based optimizers often fail.

Algorithm 15 `RBO`**, Provably Robust BBO for RL**

Initialization: (S0$_{15}$) Initialize policy parameters θ.

Perturbation sampling: (S1$_{15}$) Sample perturbations $\{g_i\}$.

Evaluation: (S2$_{15}$) Evaluate rewards $y_i = F(\theta \pm g_i)$, possibly corrupted or noisy.

Gradient estimation (robust regression / LP-decoding): (S3$_{15}$) Estimate $\hat{\nabla} F(\theta)$ by solving the robust regression

$$\hat{\nabla F}(\theta) = \arg\min_{v} \sum_{i=1}^{m} \rho\big(y_i - F(\theta) - g_i^\top v\big)\,,$$

where ρ is a robust loss (e.g., Huber or L_1). This LP-decoding view guarantees accurate gradient recovery even if up to 23% of evaluations are arbitrarily corrupted.

Update: (S4$_{15}$) Update policy parameters with

$$\theta \leftarrow \theta + \eta \hat{\nabla} F(\theta),$$

reusing past samples in an off-policy fashion to form continuous gradient flows.

Termination: (S5$_{15}$) Repeat until convergence or evaluation budget exhausted.

The main workflow of `CAS-MORE` is:

- Sample candidate solutions from a Gaussian search distribution.
- Fit a quadratic surrogate model of the objective using ordinary least squares with preprocessing.
- Update the mean and covariance separately, each under its own KL-divergence bound.
- Adapt the entropy dynamically using an evolution path heuristic.

This approach is particularly effective because:

- Decoupling mean and covariance updates improves stability and convergence.
- Adaptive entropy scheduling accelerates learning and prevents premature collapse of exploration.
- The simplified surrogate fitting method is more reliable under noisy, stochastic evaluations.

`CAS-MORE` outperforms ranking-based methods such as `CMA-ES` and other policy-gradient-inspired black-box algorithms in episodic RL, achieving faster convergence and greater robustness to noise.

Algorithm 16 extends the `MORE` algorithm to handle noisy, episodic RL tasks by stabilizing distribution updates and improving exploration. In the **initialization step** ($S0_{16}$), a Gaussian search distribution $\pi(x;\theta)$ is set with an initial mean and covariance. At each iteration, the **sampling step** ($S1_{16}$) draws a batch of candidate solutions from this distribution, which are evaluated on the stochastic objective. A quadratic surrogate model of the objective is then fit in the **surrogate fitting step** ($S2_{16}$) using ordinary least squares with preprocessing, providing a smooth local approximation even under noisy returns. In the **distribution update step** ($S3_{16}$), the mean and covariance are updated separately, each constrained by its own KL-divergence bound, which decouples exploration (covariance) from exploitation (mean shift) and prevents instability. The **entropy adaptation step** ($S4_{16}$) dynamically adjusts the distribution's entropy using an evolution-path heuristic, encouraging exploration early and annealing it as progress is made. This process repeats until convergence or the evaluation budget is exhausted ($S5_{16}$). By separating mean and covariance updates, improving surrogate reliability, and adaptively controlling entropy, `CAS-MORE` achieves faster convergence and greater robustness than ranking-based methods such as `CMA-ES` in episodic RL.

Algorithm 16 `CAS-MORE`**, Coordinate-Ascent Model-based Relative Entropy**

Initialization: ($S0_{16}$) Initialize Gaussian search distribution $\pi(x;\theta)$.
Sampling: ($S1_{16}$) Sample candidate solutions from $\pi(x;\theta)$.
Surrogate fitting: ($S2_{16}$) Fit a quadratic surrogate model of the objective using ordinary least squares with preprocessing.
Distribution update: ($S3_{16}$) Update mean and covariance with separate KL-divergence bounds.
Entropy adaptation: ($S4_{16}$) Adapt entropy schedule using the evolution path heuristic.
Termination: ($S5_{16}$) Repeat until convergence or budget exhausted.

4.2.5 `LB-SGD` – Log Barriers for Safe RL

Usmanova et al. [73] propose **Log-Barrier Stochastic Gradient Descent (`LB-SGD`)**, a BBO method tailored for constrained and safe RL. `LB-SGD` ensures that iterates remain feasible throughout the optimization process by embedding constraints into a log-barrier objective and applying `SGD`.

The main workflow of `LB-SGD` is:

- Initialize policy parameters and define the log-barrier augmented objective.
- Query a stochastic oracle (zeroth-order function values or first-order noisy gradients).
- Update policy parameters using `SGD` with an adaptive step size tuned to the smoothness constant.
- Maintain feasibility at every step through the barrier formulation.

This approach is particularly effective because:

- Log-barrier terms guarantee constraint satisfaction during all iterations.
- Adaptive step-size rules enable provable convergence in convex, strongly convex, and nonconvex cases.
- It scales efficiently to high-dimensional policy spaces, where safe `BO` is impractical.
- It provides a lightweight alternative for safe RL without requiring explicit constraint models.

LB-SGD achieves state-of-the-art safe policy optimization, balancing exploration and safety in constrained RL, and outperforms prior safe BO methods in high-dimensional tasks.

Algorithm 17 implements log-barrier stochastic gradient descent (LB-SGD) to safely optimize policies under constraints in RL. In the **initialization step** ($S0_{17}$), policy parameters are set. The **barrier formulation step** ($S1_{17}$) augments the true black-box objective with log-barrier terms that penalize approaching constraint boundaries, so that the optimization is restricted to the feasible region. At each iteration, the **oracle query step** ($S2_{17}$) obtains a stochastic estimate of the gradient of this augmented objective, either from noisy first-order gradients or from zeroth-order function queries. The **update step** ($S3_{17}$) then adjusts parameters using SGD with an adaptive step size tuned to problem smoothness. Because of the barrier terms, the **safety enforcement step** ($S4_{17}$) guarantees that all iterates remain feasible throughout the optimization. This process repeats until convergence or the evaluation budget is exhausted ($S5_{17}$). By combining log-barrier constraints with stochastic optimization, LB-SGD provides a lightweight, scalable approach to safe BBO, achieving state-of-the-art results in high-dimensional safe RL tasks where BO is impractical.

Algorithm 17 LB-SGD**, Log Barriers for Safe RL**

Initialization: ($S0_{17}$) Initialize policy parameters θ_0.
Barrier formulation: ($S1_{17}$) Define log-barrier augmented objective $\tilde{f}(\theta)$.
Oracle query: ($S2_{17}$) Obtain stochastic gradient estimate $\hat{\nabla}\tilde{f}(\theta_t)$ using zeroth-order or first-order feedback.
Update: ($S3_{17}$) Update parameters: $\theta_{t+1} = \theta_t - \eta_t \hat{\nabla}\tilde{f}(\theta_t)$ with adaptive step size η_t.
Safety enforcement: ($S4_{17}$) Barrier terms ensure all iterates remain feasible.
Termination: ($S5_{17}$) Repeat until convergence or budget exhausted.

4.2.6 PI2 vs PIBB – Policy Improvement between BBO and RL

Stulp and Sigaud [72] investigate the connection between **Policy Improvement with Path Integrals (PI2)** and its simplified form **Policy Improvement with Black-Box (PIBB)**. They show that these methods are closely related to CMA-ES and natural evolution strategies, revealing a bridge between policy search in RL and BBO.

The main workflow of PIBB is:

- Initialize policy parameters θ.

- Generate K perturbed rollouts of the policy with parameters $\theta + \epsilon_i$.
- Evaluate cumulative rewards of each rollout.
- Update parameters by direct reward-weighted averaging of perturbations.

The difference from `PI2` is in the weighting scheme: `PI2` uses exponential weighting derived from path integral control, while `PIBB` employs simpler direct reward-proportional weighting.

This approach is particularly effective because:

- It shares the simplicity and parallelism of evolutionary strategies.
- Reward-weighted averaging naturally implements a policy gradient update without value functions.
- It unifies RL-style learning rules with BBO heuristics, showing `PIBB` is a special case of `CMA-ES`.

`PI2` and `PIBB` demonstrate how concepts from BBO and RL converge, offering theoretical insights and practical hybridization opportunities.

Algorithm 18 illustrates the generic loop of Policy Improvement with Black-Box (`PIBB`), a simple yet powerful connection between RL and evolutionary BBO. In the **initialization step** ($S0_{18}$), the policy parameters θ are set. The **rollout sampling step** ($S1_{18}$) then generates K trajectories by perturbing the policy parameters with noise vectors ϵ_i. Each perturbed policy is executed in the environment, and in the **reward evaluation step** ($S2_{18}$), the cumulative reward of each rollout is measured. In the **policy update step** ($S3_{18}$), the parameters θ are adjusted by a direct reward-weighted average of the perturbations, which effectively performs a gradient-free policy gradient update. This process repeats until convergence ($S4_{18}$). The difference to `PI2` lies in the weighting: `PI2` uses exponential weights derived from path integral control theory, while `PIBB` applies simpler direct reward-proportional weights. Both methods unify ideas from RL and BBO, showing equivalence to natural evolution strategies, with `PIBB` being a special case of `CMA-ES`, while retaining parallelism, simplicity, and robustness in policy search.

Algorithm 18 `PI2` **vs** `PIBB`**, Policy Improvement between BBO and RL**

Initialization: ($S0_{18}$) Initialize policy parameters θ.

Rollout sampling: ($S1_{18}$) Sample K rollouts with perturbed parameters $\theta + \epsilon_i$.

Reward evaluation: ($S2_{18}$) Compute cumulative rewards for each rollout.

Policy update: ($S3_{18}$) Update θ by direct reward-weighted averaging of perturbations.

Termination: ($S4_{18}$) Repeat until policy converges.

4.2.7 `Q-Mamba` – Offline `MetaBBO` via Decomposed Q-Learning

Classical `MetaBBO` methods often train the meta-policy online and struggle with efficiency and large action spaces. Ma et al. [50] propose `Q-Mamba`, an *offline* `MetaBBO` framework that reformulates Dynamic Algorithm Configuration (`DAC`) as a long-sequence decision problem, learned via a decomposed Q-function with a selective state-space (`Mamba`) backbone.

The key ML component is the surrogate Q-function:
• Instead of a monolithic Q over high-dimensional actions, `Q-Mamba` uses **decomposed Q-learning**, with one head per action dimension.
• Continuous hyperparameters are discretized into bins, ensuring tractable argmax action selection.
• A `Mamba` sequence model captures long-horizon population dynamics and stabilizes training with a conservative Q-loss.

`Q-Mamba` is trained once offline on a dataset of `DAC` trajectories. In the **initialization step** ($S0_{19}$), the dataset is collected and continuous actions are discretized. In each **Q-function training step** ($S1_{19}$), the decomposed Q-heads are fitted with a `Mamba` backbone, updated by a conservative Bellman loss. The **action reconstruction step** ($S2_{19}$) selects actions dimension-wise by argmax, ensuring tractability despite large action spaces. Finally, in the **deployment step** ($S3_{19}$), the learned policy is executed: for each population state, an action is selected, applied to configure the base optimizer, and the state is updated from population dynamics. This loop continues until the evaluation budget is exhausted.

Algorithm 19 `Q-Mamba`**, Offline** `MetaBBO` **via Decomposed Q-Learning**

Initialization: ($S0_{19}$) Collect an offline `DAC` dataset $D = \{(s_t, a_t, r_t, s_{t+1})\}$. Discretize each action dimension into bins.
repeat
  Q-Function Training: ($S1_{19}$) Fit decomposed Q-heads $\{Q_\theta^k(s, a^k)\}$ with a Mamba backbone. Update with a conservative Bellman error to mitigate distributional shift.
  Action Reconstruction: ($S2_{19}$) For each state s, select action dimension-wise: $a^k = \text{argmax}_{a^k} Q_\theta^k(s, a^k)$. Concatenate $\{a^k\}$ to reconstruct the full `DAC` control vector.
  Deployment: ($S3_{19}$) At runtime, observe population state s_t, compute a_t, apply it to configure the base optimizer, and roll out one step. Update s_{t+1} from population dynamics.
until evaluation budget exhausted.

`Q-Mamba` is thus both **expressive** (via `Mamba` sequence modeling) and **tractable** (via decomposed Q-learning). Ma et al. evaluate `Q-Mamba` on BBOB test func-

tions and neuroevolution control tasks, showing strong zero-shot transfer to unseen tasks and outperforming online `MetaBBO` baselines in both efficiency and final performance.

5 Benchmarking on ML and RL for BBO Methods

Research on ML- and RL-enhanced BBO has grown rapidly, producing a diverse set of methods ranging from surrogate-based solvers to meta-RL controllers. To place these advances in context, it is essential to review prior surveys and benchmarking efforts that systematically evaluate BBO algorithms across domains. Competitions like the NeurIPS 2020 BBO Challenge [7] establish strict evaluation budgets and standardized protocols for fair comparison. Frameworks such as `MetaBox` [51] offer reproducible environments for RL-based meta-optimizers, and comprehensive studies such as Zhou et al. [76] benchmark the generalization ability of `MetaBBO-RL` methods across heterogeneous tasks.

Together, these efforts highlight both the strengths of current approaches and the challenges that remain in scaling, generalization, and reproducibility. The following subsections summarize these contributions in more detail, covering surveys of BBO and its applications, large-scale competitions, standardized benchmark platforms, and recent evaluation studies of `MetaBBO-RL` algorithms.

5.1 Benchmark Applications

Neural Architecture Search (`NAS`). `NAS` formulates the automated design of NNs as a BBO problem over a discrete or mixed-variable space. An architecture $a \in \mathcal{A}$ is typically parameterized by operation choices $o_\ell \in \mathcal{O}$ for each layer $\ell = 1, \ldots, L$, and connectivity decisions $c_m \in \{0, 1\}$ that indicate whether an edge between two nodes in the computation graph is present. The main optimization problem is

$$a^\star = \operatorname*{argmin}_{a \in \mathcal{A}} \ \mathcal{L}(a; D_{\text{train}}, D_{\text{val}}),$$

where $\mathcal{L}$ denotes the validation loss after training a on data D_{train} and evaluating on D_{val}.

In NAS-Bench-101 [75], three constraints are enforced to ensure well-formed architectures: (i) **acyclicity**, by restricting edges to the upper triangle of the adjacency matrix, ensuring the graph is a **directed acyclic graph (DAG)**, i.e., a graph with directed edges but no cycles; (ii) **connectivity**, requiring that the input and output nodes are linked by at least one directed path and that every non-null intermediate node has both incoming and outgoing edges; (iii)

null operations, introduced to allow variable-sized graphs with fixed encoding length, with the additional constraint that null nodes cannot have edges and must appear only after all non-null nodes.

Key parameters of `NAS` are the operator set $\mathcal{O}$ (e.g., convolutions, pooling, or null), the maximum search depth L (number of nodes), the binary connectivity variables c_m encoding graph edges, and the evaluation budget N that constrains the number of architectures trained or approximated. In BBO terms, `NAS` is a high-dimensional discrete optimization problem with structural constraints, often tackled with RL, evolutionary strategies, or surrogate-based optimization (e.g., `NN+MILP` [60]).

DNA Binding Optimization. This problem formulates the design of nucleotide sequences to maximize or minimize the binding affinity of a protein (e.g., transcription factors) for a DNA site. The search space is

$$\Omega = \{A, C, G, T\}^L,$$

where L is the fixed sequence length and each position takes one of the four nucleotides. A candidate sequence $x \in \Omega$ is evaluated by an objective function

$$f(x) \in \mathbb{R},$$

which returns a binding score, typically derived from biophysical models, machine-learned predictors, or costly wet-lab assays. The key parameters of the problem are: the sequence length L (dimensionality of the discrete search space), the nucleotide alphabet $\{A, C, G, T\}$ (decision domain), the binding affinity function $f(x)$ (black-box objective), structural or biological constraints $\mathcal{C}(x)$ (e.g., GC-content, motif inclusion, avoidance of repeats or secondary structures), and the evaluation budget N (maximum number of costly affinity queries). In BBO, DNA binding tasks are difficult due to the exponential size of Ω (4^L possible sequences) and the intractability of gradients. Surrogate-assisted frameworks such as `NN+MILP` [60] have been applied, where a neural surrogate predicts $f(x)$ and mixed-integer programming ensures feasible search under $\mathcal{C}(x)$.

Constrained Binary Quadratic Problems (`CBQP`). A constrained binary quadratic problem is an optimization task of the form

$$\min_{x \in \{0,1\}^n} \; x^\top Q x + c^\top x \quad \text{s.t.} \quad Ax \leq b,$$

where $x \in \{0, 1\}^n$ is a binary decision vector, $Q \in \mathbb{R}^{n \times n}$ is a symmetric quadratic cost matrix, $c \in \mathbb{R}^n$ is a linear coefficient vector, and $Ax \leq b$ encodes additional linear constraints. Such problems are NP-hard and arise in diverse applications including portfolio selection, scheduling, and network design. In BBO,

CBQPs are challenging because the quadratic objective is combinatorial and constraints must be satisfied exactly. The MINLPLib benchmark [6] provides a curated library of real-world CBQPs and mixed-integer nonlinear programs used to evaluate BBO and mixed-integer solvers. The key parameters of a CBQP instance are the number of binary variables n, the quadratic cost matrix Q, the linear term c, the constraint matrix A, and the right-hand side vector b. In ML-enhanced BBO, frameworks such as `NN+MILP` [60] have been applied to these problems, using piecewise-linear surrogates and mixed-integer programming to search the discrete feasible set efficiently.

Linear classifiers predict labels using

$$\hat{y} = \text{sign}(w^\top x + b),$$

where $w \in \mathbb{R}^d$ is a weight vector and $b \in \mathbb{R}$ is a bias term. Training typically minimizes a convex surrogate loss $\ell(y, w^\top x + b)$ that upper-bounds the 0–1 misclassification loss. Common examples include the hinge loss $\ell(y, z) = \max\{0,\, 1 - yz\}$ (used in `SVM`), the logistic loss $\ell(y, z) = \log(1 + \exp(-yz))$, or the squared loss $(y - z)^2$.

In the context of BBO, `AUC` is often the objective for tasks like classifier tuning, since it is non-differentiable and difficult to optimize directly with gradient-based methods.

5.2 BBO Challenge 2020

Candelieri et al. [7] report on the **NeurIPS 2020 BBO Challenge**, a large-scale competition designed to evaluate algorithms for derivative-free optimization under strict evaluation budgets. The challenge included diverse benchmark functions and realistic tasks (e.g., hyperparameter tuning, water distribution optimization).

The main workflow of the BBO Challenge setting was:

- Provide participants with problem domains (continuous, integer, categorical).
- Require algorithms to optimize unknown objectives within a fixed evaluation budget.
- Use unified APIs for querying black-box functions.
- Evaluate performance via normalized regret across multiple tasks.

Key insights from the challenge include:

- Portfolio and meta-learning approaches outperformed single optimizers.
- Evolutionary and Bayesian optimization methods were widely applied.
- Benchmarks fostered further development and benchmarking of general-purpose optimizers such as `HEBO` and `Squirrel`, while also comparing against baselines like `OpenTuner`

Algorithm 20 summarizes the protocol followed in the NeurIPS 2020 BBO Challenge. In the **initialization step** ($S0_{20}$), a set of benchmark problems with varying domains (continuous, integer, categorical) is prepared, and participants interact through a unified Application Programming Interface (`API`) that hides the objective functions (here, the `API` is the standardized interface through which optimizers query the hidden benchmark problems). During each iteration, a participant's optimizer proposes candidate solutions in the **submission step** ($S1_{20}$), which are then evaluated on the hidden functions in the **evaluation step** ($S2_{20}$). The performance is logged in the **scoring step** ($S3_{20}$) by computing normalized regret, allowing fair comparison across tasks of different scales. This loop repeats until the evaluation budget is exhausted ($S4_{20}$). By enforcing strict budgets and diverse problem types, the challenge created a level playing field for algorithms and highlighted the benefits of adaptive, portfolio-based, and meta-learning strategies over single solvers.

Algorithm 20 BBO Challenge Protocol (NeurIPS 2020)

($S0_{20}$) Initialize benchmark problems via the standardized `API`.
($S1_{20}$) Submit candidate solutions from participant optimizer.
($S2_{20}$) Evaluate objective values with hidden black-boxes.
($S3_{20}$) Record normalized regret performance.
($S4_{20}$) Repeat until evaluation budget exhausted.

5.3 `MetaBox` – A Benchmark for `MetaBBO-RL`

Ma et al. [51] introduce `MetaBox`, a standardized benchmark platform for RL-based meta-optimizers (`MetaBBO-RL`). The goal of `MetaBox` is to unify evaluation across a diverse set of tasks, providing fair baselines and reproducible metrics for meta-optimizers that dynamically configure black-box optimizers.

The main workflow of `MetaBox` is:

- Initialize a base optimizer (e.g., `EA`, `BO`) and an RL meta-controller.
- Observe the trajectory of the optimizer to form a state s_t.
- Select a configuration a_t according to the policy $\pi_\omega(s_t)$.

- Apply the configuration, run the optimizer, and update its state.
- Train the RL policy from accumulated rewards over time.

This benchmark is particularly valuable because:

- It covers more than 300 tasks from synthetic benchmarks to realistic domains.
- It provides a baseline library of 19 optimizers (`EA`, `BO`, `MetaBBO-RL`, etc.) implemented under a unified template for fair comparison.
- It introduces three standardized metrics: Aggregated Evaluation Indicator (`AEI`), Meta Generalization Decay (`MGD`), and Meta Transfer Efficiency (`MTE`).
- It automates the Train–Test–Log workflow, enabling reproducible evaluation.

`MetaBox` establishes one of the first standardized large-scale evaluation platforms for `MetaBBO-RL`, accelerating research on RL-based meta-optimization.

Algorithm 21 represents the generic interaction loop used in `MetaBox` to evaluate RL-based meta-optimizers. In the **initialization step** ($S0_{21}$), a base optimizer such as an evolutionary algorithm or Bayesian optimizer is instantiated, together with an RL meta-controller that will configure it. At each iteration, the **observation step** ($S1_{21}$) extracts a state s_t from the trajectory of the optimizer, summarizing features like population diversity, progress, or uncertainty. In the **configuration selection step** ($S2_{21}$), the RL policy π_ω outputs a configuration a_t that adapts the behavior of the base optimizer (e.g., adjusting parameters, operator choices, or acquisition functions). The **apply and update step** ($S3_{21}$) executes the base optimizer under the chosen configuration and updates its state accordingly. Meanwhile, the **policy learning step** ($S4_{21}$) updates the RL controller using rewards that reflect optimizer performance, enabling the policy to improve over time. This process continues until the evaluation budget or task horizon is reached ($S5_{21}$). By standardizing this loop across hundreds of tasks and dozens of optimizers, `MetaBox` enables reproducible, large-scale evaluation of `MetaBBO-RL` methods with consistent metrics such as `AEI`, `MGD`, and `MTE`.

5.4 Benchmarking `MetaBBO-RL` Approaches

Zhou et al. [76] introduce a comprehensive benchmarking study of **Meta-Black-Box Optimization with Reinforcement Learning (MetaBBO-RL)**. While recent methods such as `DEDQN`, `LDE`, and `RLPSO` show strong results

Algorithm 21 `MetaBBO-RL` **Generic Loop**

Initialization: ($S0_{21}$) Initialize a base optimizer (e.g., `EA`/`BO`) and an RL meta-controller.
Observation: ($S1_{21}$) Observe the optimizer trajectory or state s_t.
Configuration selection: ($S2_{21}$) Choose configuration $a_t \sim \pi_\omega(s_t)$ using the RL policy.
Apply and update: ($S3_{21}$) Apply configuration, run the optimizer, and update state.
Policy learning: ($S4_{21}$) Train the RL policy from accumulated rewards.
Termination: ($S5_{21}$) Repeat until the budget is exhausted or until task termination.

on selected families of problems, their generalization ability across heterogeneous tasks remains uncertain.

The main workflow of the benchmark is:

- Define task suites: synthetic noisy benchmarks and protein docking problems.
- Train `MetaBBO-RL` optimizers (e.g., `DEDQN`, `LDE`, `RLPSO`) with `REINFORCE` or `PPO` controllers.
- Evaluate across 51 test instances, measuring `AEI`, cost curves, and robustness.
- Compare against classic optimizers (`CMA-ES`, `DE`, `PSO`) and supervised meta-learners.

Algorithm 22 outlines the evaluation protocol for benchmarking reinforcement-learning-based meta-optimizers in black-box optimization. In the **task definition step** ($S0_{22}$), suites of benchmark problems are specified, including synthetic noisy functions and real-world protein docking tasks. Next, in the **training step** ($S1_{22}$), candidate `MetaBBO-RL` methods such as `DEDQN`, `LDE`, and `RLPSO` are trained using RL controllers like `REINFORCE` or `PPO`. The **evaluation step** ($S2_{22}$) tests these optimizers on held-out problem instances under strict evaluation budgets. Performance is then measured in the **metric recording step** ($S3_{22}$), where `AEI`, cost curves, and robustness measures are logged. Finally, in the **comparison step** ($S4_{22}$), the results are benchmarked against classic black-box optimizers such as `CMA-ES`, `DE`, `PSO`, and also against supervised meta-learners. This standardized workflow highlights both the strengths and generalization gaps of current `MetaBBO-RL` approaches, providing a fair basis for comparison across heterogeneous tasks.

Algorithm 22 `MetaBBO-RL` **Benchmark Evaluation**

($S0_{22}$) Define benchmark tasks: synthetic and protein docking.
($S1_{22}$) Train candidate `MetaBBO-RL` optimizers.
($S2_{22}$) Evaluate on held-out tasks with limited budgets.
($S3_{22}$) Record `AEI`, normalized cost, and robustness metrics.
($S4_{22}$) Compare against baselines (`CMA-ES`, `BO`, `PSO`).

6 Conclusion

Black-box optimization (BBO) provides a framework for problems where gradients or explicit structure are unavailable, but practical solutions must be found under limited evaluations. Classical derivative-free methods form the foundation of BBO, yet they often struggle with scalability, noise, and combinatorial complexity. Machine learning (ML) and reinforcement learning (RL) have emerged as enhancers that **complement classical solvers**, integrating into **inexact solution methods** that do not guarantee global optimality but deliver high-quality solutions under strict time or budget constraints.

Our survey shows how ML contributes through surrogate modeling, optimizer-inspired updates, meta-learning portfolios, and generative models, while RL introduces robustness, adaptive operator configuration, and meta-optimization across tasks. Benchmarking efforts such as the NeurIPS BBO Challenge, `MetaBox`, and `MetaBBO-RL` evaluation protocols provide reproducible environments to compare these approaches fairly.

In summary, ML and RL do not replace classical solvers but transform them into more scalable, robust, and adaptive frameworks for real-world optimization. Future work should deepen their integration in mixed-integer domains, improve generalization across heterogeneous tasks, and develop interpretable and efficient methods for decision-making under uncertainty.

Acknowledgements This work received funding from the National Centre for Energy II (TN02000025).

References

[1] Brian M Adams, Leif E Bauman, William J Bohnhoff, et al. Dakota: A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis, 2020. Version 6.13 User's Manual, Sandia National Laboratories.

[2] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing, 2017.

[3] A.P. Bartók and J.D. Sparks. A parallel surrogate model approach for derivative-free optimization with high-dimensional outputs. *Proceedings of the 2010 Conference on High Performance Computing (HPC)*, pages 188–194, 2010. This paper presents surrogate-based optimization and how it can be parallelized for large, high-dimensional optimization problems using HPC resources.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pages 560–569. PMLR, 2018.

[5] Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions, 2017.

[6] Michael R. Bussieck and Stefan Vigerske. MINLPLib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[7] Antonio Candelieri, Raffaele Perego, and Francesco Archetti. Black-box optimization challenge 2020. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, 2020.

[8] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *ACM Workshop on Artificial Intelligence and Security*, 2017.

[9] Xiangyi Chen, Sijia Liu, Kaidi Xu, Xingguo Li, Xue Lin, Mingyi Hong, and David Cox. Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization. *Advances in neural information processing systems*, 32, 2019.

[10] Krzysztof Choromanski, Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Deepali Jain, Yuxiang Yang, Atil Iscen, Jasmine Hsu, and Vikas Sindhwani. Provably robust blackbox optimization for reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Koushil Sreenath, editors, *Proceedings of The 3rd Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1534–1547. PMLR, 2020.

[11] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, 2009.

[12] The Scikit-Optimize contributors. Scikit-optimize: Efficient and reusable implementations of bayesian optimization, 2018.

[13] NVIDIA Corporation. *CUDA C Programming Guide*, 2021.

[14] Giuseppe Cuccu, Luca Rolshoven, Fabien Vorpe, Philippe Cudré-Mauroux, and Tobias Glasmachers. DiBB: distributing black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 341–349, New York, NY, USA, 2022. Association for Computing Machinery.

[15] Ana Luísa Custódio and Luís Nunes Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM Journal on Optimization*, 18(2):537–555, 2007.

[16] Sébastien Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the mads algorithm. *ACM Transactions on Mathematical Software*, 37(4):44:1–44:15, 2011.

[17] John C. Duchi, Michael I. Jordan, Martin J. Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Trans. Inf. Theor.*, 61(5):2788–2806, May 2015.

[18] David Eriksson, David Bindel, and Christine A Shoemaker. pySOT and POAP: An event-driven asynchronous framework for surrogate optimization. *Optimization and Engineering*, 20(4):739–775, 2019.

[19] Kyle Erwin and Andries Engelbrecht. Meta-heuristics for portfolio optimization. *Soft Computing*, 27(24):19045–19073, 2023.

[20] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436–1445, 2018.

[21] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. University of Tennessee, 1994.

[22] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[23] Roman Garnett. *Bayesian Optimization*. Machine Learning Foundations. Cambridge University Press, Cambridge, UK, 2023.

[24] Saeed Ghadimi and Guanghui Lan. Stochastic zeroth-order optimization. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[25] Hiva Ghanbari and Katya Scheinberg. Black-box optimization in machine learning with trust region based derivative free algorithm. *arXiv preprint arXiv:1703.06925*, 2017.

[26] T. Giovannelli, G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for mixed-integer nonsmooth constrained optimization. *Comput. Optim. Appl.*, 82(2):293–327, 2022.

[27] The GPyOpt authors. GPyOpt: A bayesian optimization framework in python, 2016.

[28] S. Gratton, C.W. Royer, L.N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM Journal on Optimization*, 25(3):1515–1541, 2015.

[29] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.*, 26(4-5):873–894, 2011.

[30] Stanley E Griffis, John E Bell, and David J Closs. Metaheuristics in logistics and supply chain management. *Journal of Business Logistics*, 33(2):90–106, 2012.

[31] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[32] Nicholas J. Higham. Optimization by direct search in matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 14(2):317–333, 1993.

[33] Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. OR-Gym: A reinforcement learning library for operations research problems, 2020.

[34] Maximilian Hüttenrauch and Gerhard Neumann. Robust black-box optimization for stochastic search and episodic reinforcement learning. *Journal of Machine Learning Research*, 25(153):1–44, 2024.

[35] W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.*, 35(2):1–25, 2008.

[36] Bassem Jarboui, Patrick Siarry, and Jacques Teghem. *Metaheuristics for production scheduling*. John Wiley & Sons, 2013.

[37] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21:1–27, 2019.

[38] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.*, pages 365–414, 2022.

[39] Morteza Kimiaei. An improved randomized algorithm with noise level tuning for large-scale noisy unconstrained DFO problems. *Numerical Algorithms*, January 2025.

[40] Morteza Kimiaei and Arnold Neumaier. Effective matrix adaptation strategy for noisy derivative-free optimization. *Mathematical Programming Computation*, 16(3):459–501, July 2024.

[41] Morteza Kimiaei and Arnold Neumaier. MATRS: heuristic methods for noisy derivative-free bound-constrained mixed-integer optimization. *Mathematical Programming Computation*, 17(3):505–546, May 2025.

[42] Morteza Kimiaei, Arnold Neumaier, and Parvaneh Faramarzi. New subspace method for unconstrained derivative-free optimization. *ACM Transactions on Mathematical Software*, 49(4):1–28, December 2023.

[43] Wataru Kumagai and Keiichiro Yasuda. Black-box optimization and its applications. In *Innovative Systems Approach for Facilitating Smarter World*, pages 81–95. Springer, 2023.

[44] Jeffrey Larson, Matt Menickelly, and Stefan M. Wild. Derivative-free optimization methods. *Acta Numer.*, 28:287–404, 2019.

[45] Xiaobin Li, Kai Wu, Xiaoyu Zhang, and Handing Wang. B2opt: Learning to optimize black-box optimization with little budget. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 18502–18510, 2025.

[46] Zihao Li, Hui Yuan, Kaixuan Huang, Chengzhuo Ni, Yinyu Ye, Minshuo Chen, and Mengdi Wang. Diffusion model for data-driven black-box optimization, 2024.

[47] G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for bound constrained mixed-integer optimization. *Comput. Optim. Appl.*, 53(2):505–526, 2011.

[48] G. Liuzzi, S. Lucidi, and F. Rinaldi. An algorithmic framework based on primitive directions and nonmonotone line searches for black-box optimization problems with integer variables. *Math. Program. Comput.*, 12(4):673–702, 2020.

[49] Stefano Lucidi and Marco Sciandrone. A derivative-free algorithm for bound constrained optimization. *Computational Optimization and Applications*, 21:119–142, 2002.

[50] Zeyuan Ma, Zhiguang Cao, Zhou Jiang, Hongshu Guo, and Yue-Jiao Gong. Meta-black-box-optimization through offline q-function learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *ICML '25*, Vancouver, Canada, 2025. PMLR.

[51] Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Zhenrui Li, Guojun Peng, Yue-Jiao Gong, Yining Ma, and Zhiguang Cao. Metabox: A benchmark platform for meta-black-box optimization with reinforcement learning. In *NeurIPS 2023 Track on Datasets and Benchmarks*, 2023.

[52] Zeyuan Ma, Zhiyang Huang, Jiacheng Chen, Zhiguang Cao, and Yue-Jiao Gong. Surrogate learning in meta-black-box optimization: A preliminary study. 2025.

[53] The MathWorks. *Optimization Toolbox*. The MathWorks, Inc., Natick, MA, USA, 2024.

[54] Laurent Meunier, Herilalaina Rakotoarison, Pak Kan Wong, Baptiste Roziere, Jeremy Rapin, Olivier Teytaud, Antoine Moreau, and Carola Doerr. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. *IEEE Transactions on Evolutionary Computation*, 26(3):490–500, June 2022.

[55] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20(1):172–191, 2009.

[56] J. Müller. MISO: mixed-integer surrogate optimization framework. *Optim. Eng.*, 17(1):177–203, 2015.

[57] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free coordinate descent methods for minimizing convex functions. *CoreGRID Technical Report*, 2017.

[58] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):559–595, 2017.

[59] Arnold Neumaier and Morteza Kimiaei. An improvement of the goldstein line search. *Optimization Letters*, 18:1313–1333, 2024.

[60] Theodore Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[61] N. Ploskas and N. V. Sahinidis. Review and comparison of algorithms and software for mixed-integer derivative-free optimization. *J. Glob. Optim.*, 82(3):433–462, 2021.

[62] Margherita Porcelli and Philippe L. Toint. Exploiting problem structure in derivative-free optimization with BFO. *ACM Transactions on Mathematical Software (TOMS)*, 47(3):1–26, 2021.

[63] Michael JD Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge, 2009.

[64] M.J.D. Powell. UOBYQA: Unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[65] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

[66] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.*, 56(3):1247–1293, 2012.

[67] Mikita Sazanovich, Anastasiya Nikolskaya, Yury Belousov, and Aleksei Shpilman. Solving black-box optimization challenge via learning search space partition for local bayesian optimization. In Hugo Jair Escalante and Katja Hofmann, editors, *NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 77–85. PMLR, 2021.

[68] Ohad Shamir. An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, 18(52):1–11, 2017.

[69] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25 (NeurIPS)*, pages 2951–2959, 2012.

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

[71] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[72] Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. 2012.

[73] Ilnura Usmanova, Yarden As, Maryam Kamgarpour, and Andreas Krause. Log barriers for safe black-box optimization with application to safe reinforcement learning. *Journal of Machine Learning Research*, 25(171):1–54, 2024.

[74] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

[75] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR, 2019.

[76] Mingxuan Zhou, Kai Li, Jiarui Zhang, Chen-Xiao Gao, and Chao Qian. Benchmarking meta-black-box optimization with reinforcement learning approaches. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2024.