MetaDE: Evolving Differential Evolution by Differential Evolution

Minyang Chen, Chenchen Feng, and Ran Cheng, *Senior Member, IEEE*

***Abstract*—As a cornerstone in the Evolutionary Computation (EC) domain, Differential Evolution (DE) is known for its simplicity and effectiveness in handling challenging black-box optimization problems. While the advantages of DE are well-recognized, achieving peak performance heavily depends on its hyperparameters such as the mutation factor, crossover probability, and the selection of specific DE strategies. Traditional approaches to this hyperparameter dilemma have leaned towards parameter tuning or adaptive mechanisms. However, identifying the optimal settings tailored for specific problems remains a persistent challenge. In response, we introduce MetaDE, an approach that evolves DE's intrinsic hyperparameters and strategies using DE itself at a meta-level. A pivotal aspect of MetaDE is a specialized parameterization technique, which endows it with the capability to dynamically modify DE's parameters and strategies throughout the evolutionary process. To augment computational efficiency, MetaDE incorporates a design that leverages parallel processing through a GPU-accelerated computing framework. Within such a framework, DE is not just a solver but also an optimizer for its own configurations, thus streamlining the process of hyperparameter optimization and problem-solving into a cohesive and automated workflow. Extensive evaluations on the CEC2022 benchmark suite demonstrate MetaDE's promising performance. Moreover, when applied to robot control via evolutionary reinforcement learning, MetaDE also demonstrates promising performance. The source code of MetaDE is publicly accessible at: https://github.com/EMI-Group/metade.**

***Index Terms*—Differential Evolution, Meta Evolutionary Algorithm, GPU Computing**

I. Introduction

The Differential Evolution (DE) [1–4] algorithm, introduced by Storn and Price in 1995, has emerged as a cornerstone in the realm of evolutionary computation (EC) for its prowess in addressing complex optimization problems across diverse domains of science and engineering. DE's comparative advantage over other evolutionary algorithms is evident in its streamlined design, robust performance, and ease of implementation. Notably, with just three primary control parameters, i.e., scaling factor, crossover rate, and population size, DE operates efficiently. This minimalistic design, paired with a lower algorithmic complexity, positions DE as an ideal candidate for large-scale optimization problems. Its influential role in the optimization community is further cemented by its extensive research attention and successful applications over the past decades [5–7], with DE and its derivatives often securing top positions in the IEEE Congress on Evolutionary Computation (CEC) competitions.

Despite the well recognized performance, DE is not without limitations. Particularly, some studies indicate that DE's optimization process may stagnate if it fails to generate offspring solutions superior to their parents [8, 9]. To avert this stagnation, selecting an appropriate parameter configuration to enhance DE's search capabilities becomes crucial.

However, the No Free Lunch (NFL) theorem [10] suggests that a universally optimal parameter configuration is unattainable. For example, while a higher mutation factor may aid in escaping local optima, a lower crossover probability might be preferable for problems with separability characteristics.

To address the intricate challenge of parameter configuration in DE, researchers often gravitate towards two predominant strategies: *parameter control* and *parameter tuning* [11–13]. Parameter control is a dynamic approach wherein the algorithm's parameters are adjusted on-the-fly during its execution. This adaptability allows the algorithm to respond to the evolving characteristics of the problem landscape, enhancing its chance of finding optimal or near-optimal solutions. Notably, DE has incorporated this strategy in several of its variants. For instance, jDE [14] adjusts the mutation factor and crossover rate during the run, while SaDE [15] dynamically chooses a mutation strategy based on its past success rates. Similarly, JaDE [16] and CoDE [17] employ adaptive mechanisms to modify control parameters and mutation strategies, respectively.

In contrast, parameter tuning is a more static methodology, wherein the optimal configuration is established prior to the algorithm's initiation. It aims to discover a parameter set that consistently demonstrates robust performance across various runs and problem instances. Despite its potential for reliable outcomes, parameter tuning is known for its computational intensity, often necessitating dedicated optimization efforts or experimental designs to identify the optimal parameters, which may explain its limited exploration in the field. Viewed as an optimization challenge, parameter tuning is also referred to as meta-optimization [18]. This perspective gave rise to *MetaEA*, which optimizes the parameters of an EA using another EA.

Despite MetaEA's methodological elegance and simplicity, it confronts the significant challenge of depending on extensive function evaluations. Fortunately, the inherent parallelism within MetaEA, across both meta-level and base-level popula-

This work was supported in part by Guangdong Basic and Applied Basic Research Foundation (No. 2024B1515020019). (*Corresponding author: Ran Cheng*).

Minyang Chen was with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. E-mail: cmy1223605455@gmail.com.

Chenchen Feng is with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. E-mail: chenchenfengcn@gmail.com.

Ran Cheng is with the Department of Data Science and Artificial Intelligence, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. E-mail: ranchengcn@gmail.com.

tions, renders it particularly amenable to parallel computing environments. However, a notable disparity exists between methodological innovations and the availability of advanced computational infrastructures, thus limiting MetaEA's potential due to the lack of advanced hardware accelerations such as GPUs. To bridge this gap, we introduce the *MetaDE* approach, which embodies the MetaEA paradigm by employing DE in a meta-level to guide the evolution of a specially tailored Parameterized Differential Evolution (PDE).

Designed with adaptability in mind, PDE can flexibly adjust its parameters and strategies, paving the way for a wide range of DE configurations. As PDE interacts with the optimization problem at hand, the meta-level DE observes and refines PDE's settings to better align with the problem's characteristics. Amplifying the efficiency of this nested optimization approach, MetaDE is integrated with a GPU-accelerated EC framework, thus weaving together parameter refinement and direct problem-solving into a seamless end-to-end approach to black-box optimization. In summary, our main contributions are as follows.

- **Parameterized Differential Evolution:** We have introduced Parameterized Differential Evolution (PDE), a variant of DE with augmented parameterization. Unlike traditional DE algorithms that come with fixed mutation and crossover strategies, PDE's architecture offers users the flexibility to adjust these parameters and strategies to fit the problem at hand. This design not only allows for the creation of diverse DE configurations tailored for specific challenges but also ensures efficient computation. To achieve this, all core operations of PDE, including mutation, crossover, and evaluation, have been optimized for parallel execution to harness advancement of GPU acceleration.
- **MetaDE:** Building on the MetaEA paradigm, we have designed the MetaDE approach. Specifically, MetaDE employs a meta-level DE as an `evolver` to iteratively refine PDE's hyperparameters, which is guided by performance feedback from multiple PDE instances acting as `executors`. This continuous optimization ensures PDE's configurations remain aligned with evolving problem landscapes. Moreover, we have incorporated several specialized methods to further enhance the performance of MetaDE.
- **GPU-accelerated Implementation:** Breaking away from the limitations of conventional parameter tuning, we integrate MetaDE with a GPU-accelerated computing framework – EvoX [19], which enhances MetaDE's computational prowess for facilitating swifter evaluations and algorithmic refinements. With this specialized implementation, MetaDE provides an efficient and automated end-to-end approach to black-box optimization.

The subsequent sections are organized as follows. Section II presents some preliminary knowledge for this work. Section III elucidates the intricacies of the proposed approach, including PDE and the MetaDE. Section IV showcases the experimental results. Finally, Section V wraps up the discourse and points towards avenues for future work.

II. Preliminaries

A. DE and its Parameter Adaption

1) Overview of DE: As a typical EC algorithm, DE's essence lies in its differential mutation mechanism that drives the evolution of a population. The operational cycle of DE unfolds iteratively, with each iteration embodying specific phases, as elaborated in Algorithm 1:

1. **Initialization** (Line 1): The algorithm initializes a set of potential solutions. Each of these solutions, representing vectors of decision variables, is randomly generated within the search space boundaries.
2. **Mutation** (Lines 6-7): Each solution undergoes mutation to produce a mutant vector. This mutation process involves combinations of different individuals to form the mutant vectors.
3. **Crossover** (Line 8): The crossover operation interchanges components between mutants and the original solutions to generate a trial vector.
4. **Selection** (Lines 10-12): The trial vector competes against the original solution based on fitness, with the better solution progressing to the next generation.

DE progresses through cycles of mutation, crossover, and selection, persisting until it encounters a termination criterion. This could manifest as either reaching a predefined number of generations or achieving a target fitness threshold. The algorithm's adaptability allows for the spawning of myriad DE variants by merely tweaking its mutation and crossover operations. Specifically, DE variants follow a unified naming convention: `DE/x/y/z`, where `x` identifies the base vector used for mutation, `y` quantifies the number of difference involved, and `z` typifies the crossover method employed. For example, the DE variant as presented in Algorithm 1 is named as `DE/rand/1/bin`.

Algorithm 1 DE

```
Input: $D$, $NP$, $F$, $CR$, $G_{max}$
1: Initialize population $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{NP}\}$
2: Evaluate the fitness of each individual in the population
3: $g = 0$
4: while $g \leq G_{max}$ do
5:     for $i = 1$ to $NP$ do
6:         Randomly select $\mathbf{x}_{r_1}$, $\mathbf{x}_{r_2}$, and $\mathbf{x}_{r_3}$ from $\mathbf{X}$,
           such that $r_1 \neq r_2 \neq r_3 \neq i$
7:         Compute the mutant vector: $\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3})$
8:         Perform crossover for each variable between $\mathbf{x}_i$ and $\mathbf{v}_i$:
           $u_{i,j} = \begin{cases} v_{i,j}, & \text{if rand}(0,1) \leq CR \text{ or } j = \text{randint}(1, D) \\ x_{i,j}, & \text{otherwise} \end{cases}$
9:         Evaluate the fitness of $\mathbf{u}_i$
10:        if $\text{f}(\mathbf{u}_i) \leq \text{f}(\mathbf{x}_i)$ then
11:            Replace $\mathbf{x}_i$ with $\mathbf{u}_i$ in the population
12:        end if
13:    end for
14:    $g = g + 1$
15: end while
16: return the best fitness
```

2) Parameter Modulation in DE: DE employs a unique mutation mechanism, which adapts to the problem's natural scaling. By adjusting the mutation step's size and orientation to the objective function landscape, DE embraces the *contour matching principle* [20], which promotes basin-to-basin transfer for enhancing the convergence of the algorithm.

At the core of DE's mutation is the scaling factor F. This factor not only determines the mutation's intensity but also governs its trajectory and ability to bypass local optima. Commonly, F is set within the $[0.5, 1]$ interval, with a starting point often at 0.5. While values outside the $[0.4, 1]$ range can sometimes yield good results, an F greater than 1 tends to slow convergence. Conversely, values up to 1 generally promise swifter and more stable outcomes [21]. Nonetheless, to deter settling at suboptimal solutions too early, F should be adequately elevated.

Parallel to mutation, DE incorporates a uniform crossover operator, which is often labeled as discrete recombination or binomial crossover in the GA lexicon. The crossover constant CR also plays a pivotal role, which determines the proportion of decision variables to be exchanged during the generation of offspring. A low value for CR ensures only a small portion of decision variables are modified per iteration, thus leading to axis-aligned search steps. As CR increases closer to 1, offspring tend to increasingly reflect their mutant parent, thereby curbing the generation of orthogonal search steps [5].

For classical DE configurations, such as `DE/rand/1/bin`, rotational invariance is achieved only when CR is maxed out at 1. Here, the crossover becomes wholly vector-driven, and offspring effectively mirror their mutants. However, the optimal CR is intrinsically problem-dependent. Empirical studies recommend a CR setting within the $[0, 0.2]$ range for problems characterized by separable decision variables. Conversely, for problems with non-separable decision variables, a CR in the proximity of $[0.9, 1]$ is more effective [5].

The adaptability of DE is evident in its wide spectrum of variants, each distinct in its mutation and crossover strategies with delicate parameter modulations. In the following, we will detail seven mutation strategies and three crossover strategies, all of which are widely-recognized in state-of-the-art DE variants. Here, the subscript notation in $\mathbf{x}$ specifies the individual selection technique. For instance, $\mathbf{x}_r$ and $\mathbf{x}_{best}$ correspond to randomly selected and best-performing individuals respectively, whereas $\mathbf{x}_i$ represents the currently evaluated individual.

Mutation Strategies:

1. `DE/rand/1`:

$$\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \,. \tag{1}$$

2. `DE/best/1`:

$$\mathbf{v}_i = \mathbf{x}_{\text{best}} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \,. \tag{2}$$

3. `DE/rand/2`:

$$\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) + F \cdot (\mathbf{x}_{r_4} - \mathbf{x}_{r_5}) \,. \tag{3}$$

4. `DE/best/2`:

$$\mathbf{v}_i = \mathbf{x}_{\text{best}} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) + F \cdot (\mathbf{x}_{r_3} - \mathbf{x}_{r_4}) \,. \tag{4}$$

5. `DE/current-to-best/1`:

$$\mathbf{v}_i = \mathbf{x}_i + F \cdot (\mathbf{x}_{\text{best}} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \,. \tag{5}$$

The above five classical mutation strategies, introduced by Storn and Price [20], cater to various problem landscapes. For instance, the 'rand' variants help maintain population diversity, while strategies using two differences typically produce more diverse offspring than those relying on a single difference.

6. `DE/current-to-pbest/1`:

$$\mathbf{v}_i = \mathbf{x}_i + F \cdot (\mathbf{x}_{\text{pbest}} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \,. \tag{6}$$

This strategy originates from JaDE [16]. $\mathbf{x}_{\text{pbest}}$ is randomly selected from the top *p%* of individuals in the population (typically the top 10%) to strike a balance between exploration and exploitation.

7. `DE/current-to-rand/1`:

$$\mathbf{u}_i = \mathbf{x}_i + K_i \cdot (\mathbf{x}_{r_1} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \,. \tag{7}$$

Here, K_i is a random number from $U(0, 1)$. This strategy, originally proposed in [22], emphasizes rotational invariance. By bypassing the crossover phase, it directly yields the trial vector $\mathbf{u}_i$. Thus, it is ideal for addressing non-separable rotation challenges and has been a cornerstone for multiple adaptive DE variations.

Crossover Strategies:

1. Binomial Crossover:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } r \leq CR \text{ or } j = j_{\text{rand}} \\ x_{i,j}, & \text{otherwise,} \end{cases} \tag{8}$$

where j_{rand} is a random integer between 1 and D. This strategy is a cornerstone in DE.

2. Exponential Crossover:

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if } j = \langle n \rangle_d, \langle n+1 \rangle_d, ..., \langle n+L-1 \rangle_d \\ x_{i,j} & \text{otherwise,} \end{cases} \tag{9}$$

where $\langle\rangle_d$ is a modulo operation with D and L representing the crossover length, following a censored geometric distribution with a limit of D and probability of CR. By focusing on consecutive variables, this strategy excels in handling problems with contiguous variable dependencies.

3. Arithmetic Recombination:

$$\mathbf{u}_i = \mathbf{x}_i + K_i \cdot (\mathbf{v}_i - \mathbf{x}_i), \tag{10}$$

where K_i is a random value from $U(0, 1)$. Exhibiting rotational invariance, this strategy, when combined with the `DE/rand/1` mutation, results in the `DE/current-to-rand/1` strategy [5], as described by:

$$\begin{aligned} \mathbf{u}_i &= \mathbf{x}_i + K_i \cdot (\mathbf{v}_i - \mathbf{x}_i) \\ &= \mathbf{x}_i + K_i(\mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) - \mathbf{x}_i) \\ &= \mathbf{x}_i + K_i(\mathbf{x}_{r_1} - \mathbf{x}_i) + K_i \cdot F(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}), \end{aligned} \tag{11}$$

which is equivalent to Eq. (7).

3) Adaptive DE: The development of parameter adaption in DE has witnessed significant advancements over time, from initial endeavors in parameter adaptation to recent sophisticated methods that merge multiple strategies. This subsection traces the chronological advancements, emphasizing the pivotal contributions and their respective impacts on adaptive DE.

The earliest phase in DE's adaption centered on the modification of the crossover rate CR . Pioneering algorithms such as SPDE [23] incorporated CR within the parameter set of individuals, enabling its simultaneous evolution with the decision variables of the problem to be solved. This strategy was further refined by SDE [24], which assigned CR for each individual based on a normal distribution. Subsequent research efforts shifted focus to the scaling factor F. In this context, DETVSF [25] dynamically adjusted F, fostering exploration during the algorithm's nascent stages and pivoting to exploitation in later iterations. Building on this, FaDE [26] employed fuzzy logic controllers to optimize mutation and crossover parameters.

The DESAP [27] algorithm marked a significant paradigm shift by introducing self-adapting populations and encapsulating control parameters within individuals. Successive contributions like jDE [14], SaDE [15], and JaDE [16] accentuated the significance of parameter encoding, integrated innovative mutation strategies, and emphasized archiving optimization trajectories using external repositories. Further, EPSDE [21] and CoDE [17] enhanced the offspring generation process, amalgamating multiple strategies with randomized parameters.

The contemporary landscape of adaptive DE is characterized by complex methodologies and refined strategies. Algorithms such as SHADE [28] and LSHADE [29] championed the utilization of success-history mechanisms and dynamic population size modifications. Notable developments like ADE [30] introduced a biphasic parameter adaptation mechanism. The domain further expanded with algorithms like LSHADE-RSP [31], IMODE [32], and LADE [33], emphasizing mechanisms such as selective pressure, the integration of multiple DE variants, and the automation of the learning process.

Undoubtedly, the adaptive DE domain has witnessed transformative growth, with each phase of its evolution contributing to its current sophistication. However, despite these advancements, many adaptive strategies remain empirical and hinge on manual designs, while their effectiveness is not universally guaranteed.

B. Distributed DE

The integration of distributed (i.e., multi-population) strategies also significantly enhances the efficacy of DE. Leading this advancement, Weber *et al.* conducted extensive research on scale factor interactions and mechanisms within a distributed DE framework [34–36], followed by ongoing developments along the pathway [7]. For example, some works such as EDEV [37], MPEDE [38] and IMPEDE [39] adopted multi-population frameworks to ensemble various DE variants/operators, while the other works such as DDE-AMS [40] and DDE-ARA [41] employed multiple populations for adaptive resource allocations.

Despite the achievements, current implementations of distributed DE often focus predominantly on algorithmic improvements, while overlooking potential enhancements from advanced hardware accelerations such as GPU computing. Besides, the design of these distributed strategies often features intricate and rigid configurations that lack proper flexibility.

C. MetaEAs

Generally, the term *meta* refers to a higher-level abstraction of an underlying concept, often characterized by its *recursive* nature. In the context of EC, inception of the Meta Evolutionary Algorithms (MetaEAs) can be traced back to the pioneering works of Mercer and Sampson [42] in the late 1970s. Under the initiative termed *meta-plan*, their pioneering efforts aimed at enhancing EA performance by optimizing its parameters through another EA. Although sharing similarities with hyperheuristics [43–45], a major difference distinguishes MetaEAs: while hyperheuristics often delve into selecting and fine-tuning a set of predefined algorithms, MetaEAs concentrate on the paradigm of refining the parameters of EAs by EAs. Notably, MetaEAs are also akin to ensemble of algorithms, such as EDEV [37] and CoDE [17], which amalgamate diverse algorithms to ascertain the most efficacious among them.

Advancing the meta-plan concept, MetaGA [46] emerged as a significant milestone. Here, a genetic algorithm (GA) was deployed to fine-tune six intrinsic control parameters, namely: population size, crossover rate, mutation rate, generation gap, scaling window, and selection strategy. The efficacy of this approach was gauged using dual metrics: online and offline performance.

The evolution of the concept continued with MetaEP [47], which offers a meta-level evolutionary programming (EP) that could concurrently evolve optimal parameter settings. Another pivotal contribution was the Parameter Relevance Estimation and Value Calibration (REVAC) [48], which served as a meta estimation of distribution algorithm (MetaEDA). Utilizing a GA at its core, REVAC iteratively discerned promising parameter value distributions within the configuration space.

Innovations in the domain persisted with the Gender-based GA (GGA) [49], inspired by natural gender differentiation, and other notable methods like MetaCMAES [50]. As articulated in the PhD thesis by Pedersen [18], a profound insight into MetaEA revealed that while contemporary optimizers endowed with adaptive behavioral parameters offered advantages, they were often eclipsed by streamlined optimizers under appropriate parameter tuning. This thesis, which embraced DE as one of its optimization tools, employed the Local Unimodal Sampling (LUS) heuristic for tuning parameters such as NP, F, and CR.

Culminating the discourse, the work in [51] demonstrated the scalability of MetaEAs by harnessing it within a large-scale distributed computing environment. With the $(\mu, \lambda)-$ES steering the meta-level tuning, base-level algorithms like GA, ES, and DE were adeptly optimized. For DE, parameters optimized encompassed NP, mutation operator, F, CR, and PF (parameter for the *either-or* strategy), enhancing MetaEAs' prowess in addressing intricate, large-scale optimization problems. Recently, the MetaEA paradigm has also been employed for automated design of ensemble DE [52].

The field of MetaEAs has shown steady progress since its inception in the 1970s. However, despite the achievements, the landscape of MetaEAs research still confronts certain limitations. Notably, the research, while promising, has predominantly remained confined to smaller-scale implementations. The anticipated leap to large-scale experiments, especially those that might benefit from GPU acceleration, remains largely uncharted. This underscores an imperative need for more extensive empirical validations and the exploration of contemporary computational resources to fully realize the potential of MetaEAs.

D. GPU-accelerated EC Framework

To capitalize on the advancements of modern computing infrastructures, we have seamlessly integrated our proposed MetaDE with EvoX [19], a distributed GPU-accelerated computing framework for scalable EC. This integration ensures that MetaDE enables efficient execution and optimization for large-scale evaluations.

The EvoX framework provides several distinctive features. Primarily, it is designed for optimal performance across diverse distributed systems and is tailored to manage large-scale challenges. Its user-friendly functional programming model simplifies the EC algorithm development process, reducing inherent complexities. The framework cohesively integrates data streams and functional elements into a comprehensive workflow, underpinned by a sophisticated hierarchical state management system. Moreover, EvoX features a rich library of EC algorithms, proficient in addressing a wide array of tasks, from black-box optimization to advanced areas such as deep neuroevolution and evolutionary reinforcement learning.

III. Proposed Approach

The foundational premise of MetaDE is to utilize a core DE algorithm to evolve an ensemble of parameterized DE variants. Within this framework, the core DE, which tunes the parameters, is termed the `evolver`. In contrast, each parameterized DE variant, which optimizes the problem at hand, is termed the `executor`. This section commences by augmenting the parameterization of DE in a more general manner, such that DE is made *evolvable* by spawning various DE variants by modulating the parameters. Then, this section details the integration of proposed MetaDE within a meta-framework, together with a brief introduction to the GPU-accelerated implementation.

A. Augmented Parameterization of DE

To make DE evolvable, this subsection introduces the Parameterized Differential Evolution (PDE), an extension of the standard DE designed to augment its flexibility through the parameterization of mutation and crossover strategies. While PDE retains the foundational principles of standard DE, its distinctiveness lies in its capability to generate a multitude of strategies by modulating the parameters.

In the standard DE framework, tunability is constrained to the adjustments of the F and CR parameters, and strategies are bound by predefined rules. To augment this limited flexibility, PDE introduces a more granular parameterization. Building upon DE's notation `DE/x/y/z`, PDE encompasses six parameters: F (scale factor), CR (crossover rate), bl (base vector left), br (base vector right), dn (difference number), and cs (crossover scheme). The combined roles of bl and br determine the base vector, leading to a strategy notation for PDE expressed as `DE/bl-to-br/dn/cs`.

Each parameter's nuances and the array of strategy combinations they enable are elaborated upon in the subsequent sections. Fig. 1 provides a visual representation of these parameters, delineated within dashed boxes.

1) Scale Factor F and Crossover Rate CR: The scale factor F and crossover rate CR serve as pivotal parameters in DE, both represented as real numbers.

The F parameter regulates the differential variation among population entities. Elevated values induce exploratory search behaviors, while lower values encourage more exploitation. Although Storn and Price originally identified [0, 2] as an effective domain for F [4], contemporary DE variants deem $F \leq 1$ as more judicious [16, 17, 21, 28, 29]. Consequently, PDE constrains F within [0,1].

On the other hand, CR controls the recombination extent during crossover. Higher CR make it more likely to try out new gene combinations from changes, while lower values keep the genes more stable and similar to the original. As a rate (or probability), CR is confined to the [0, 1] interval.

2) Augmented Parameterization of Mutation: The distinctiveness of PDE lies in its ability to generate a multitude of mutation strategies by modulating parameters bl, br, and dn. Parameters bl (base vector left) and br (base vector right) select one of four possible vectors:

- `rand`: A randomly chosen individual from the population.
- `best`: The best individual in the population.
- `pbest`: A random selection from the top $p\%$ of individuals.
- `current`: The present parent individual.

Parameter dn, controls the number of differences in mutation, assumes values in the set $\{1, 2, 3, 4\}$. Specifically, each difference Δ captures the difference between two unique and randomly selected individuals from the population. Therefore, the mutation formulation `DE/bl-to-br/dn` is:

$$\mathbf{v} = \mathbf{x}_{bl} + F \cdot (\mathbf{x}_{br} - \mathbf{x}_{bl}) + F \cdot (\Delta_1 + ... + \Delta_{dn}). \quad (12)$$

In the implementation of PDE, bl and br are encoded as: 1: `rand`, 2: `best`, 3: `pbest`, and 4: `current`. In addition, if both bl and br assume identical values, the term $F \cdot (\mathbf{x}_{br} - \mathbf{x}_{bl})$ will disappear. The base vector takes the value of $\mathbf{x}_{bl}$ directly and the mutation strategy then becomes non-directional (e.g., `DE/rand/1`).

3) Augmented Parameterization of Crossover: Parameter cs defines the crossover strategies in PDE, comprising those elaborated in Section II-A: binomial crossover, exponential crossover, and arithmetic recombination. Specifically, cs is encoded as: 1: `bin`, 2: `exp`, and 3: `arith`, representing

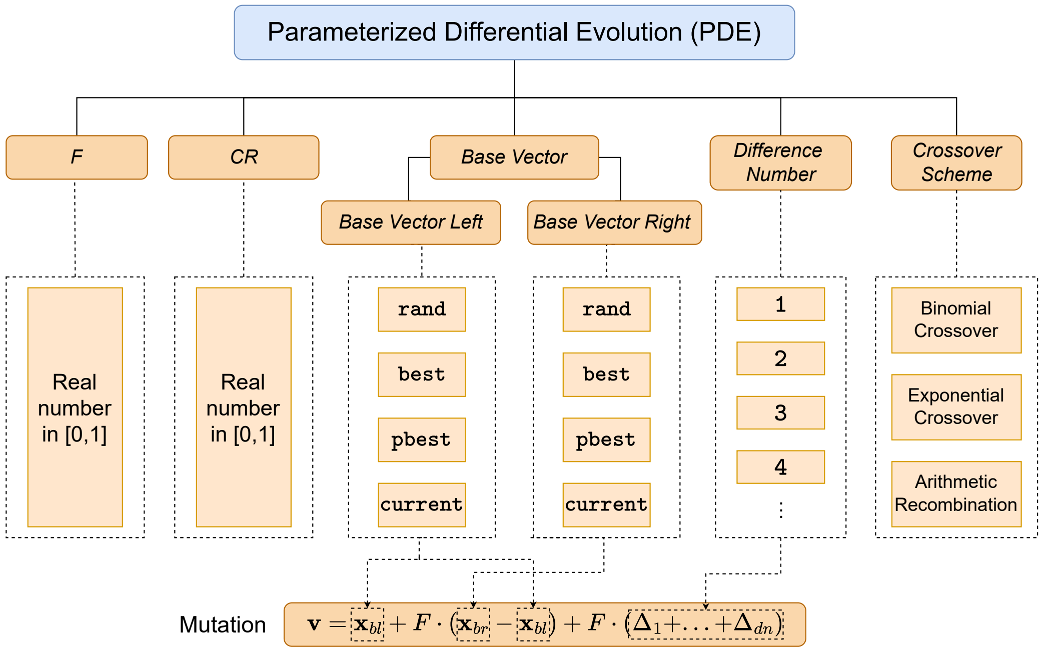


Fig. 1: Parameter delineation of PDE and their respective domains. PDE comprehensively parameterizes DE, endorsing unrestricted parameter and strategy modifications. In this schema, F and CR are continuous parameters, whereas others are categorical. The dashed-line boxes exhibit their specific value ranges. The mutation function is derived from the base vector left, base vector right, and difference number parameters.

binomial crossover, exponential crossover, and arithmetic recombination respectively.

As a result, the augmented parameterizations for mutation and crossover, denoted as `DE/bl-to-br/dn/cs`, will give rise to a spectrum of 192 distinctive strategies in total. This breadth allows for the encapsulation of mainstream strategies detailed in Section II-A, as illustrated in Table I. When synergized with F and CR, this culminates in a comprehensive parameter configuration landscape.

TABLE I
THE ENCODING OF TYPICAL DE VARIANTS BY THE PROPOSED PDE.

strategy	bl	br	dn	cs
`DE/rand/1/bin`	1	1	1	1
`DE/best/1/bin`	2	2	1	1
`DE/current-to-best/1/bin`	4	2	1	1
`DE/rand/2/bin`	1	1	2	1
`DE/best/2/bin`	2	2	2	1
`DE/current-to-pbest/1/bin`	4	3	1	1
`DE/current-to-rand/1*`	1	1	1	3

[*] As per Eq. (11), `DE/current-to-rand/1` is equivalent to `DE/rand/1/arith`.

Algorithm 2 details PDE's procedure. Its distinct attribute is Line 3, where the mutation function is shaped by bl, br, and dn. Line 4 designates the crossover function based on cs. The evolutionary phase commences at Line 6. Notably, PDE's concurrent mutation and crossover operations (Lines 7-10) are tensorized to facilitate parallel offspring generation, which deviate from the conventional sequential operations in standard DE. With such a tailored procedure, the computational efficiency can be substantially improved.

Algorithm 2 Parameterized DE (PDE)

```
Input: $D$, $NP$, $G_{max}$, $F$, $CR$, $bl$ (base vector left), $br$ (base vector
    right), $dn$ (difference number), $cs$ (crossover scheme)
 1: Initialize population $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{NP}\}$
 2: Evaluate the fitness of each individual in the population
 3: Generate mutation function $M(\mathbf{X})$ according to $bl$, $br$, $dn$:
    $\mathbf{v} = \mathbf{x}_{bl} + F \cdot (\mathbf{x}_{br} - \mathbf{x}_{bl}) + F \cdot (\Delta_1 + ... + \Delta_{dn})$
 4: According to $cs$, choose a crossover function $C(\mathbf{V}, \mathbf{X})$ in (8-10)
 5: $g = 0$
 6: while $g \le G_{max}$ do
 7:     Generate $NP$ mutant vectors: $\mathbf{V} = M(\mathbf{X})$
 8:     Perform crossover for all mutant vectors: $\mathbf{U} = C(\mathbf{V}, \mathbf{X})$
 9:     Evaluate the fitness of $\mathbf{U}$
10:     Make selection between $\mathbf{U}$ and $\mathbf{X}$
11:     $g = g + 1$
12: end while
13: return the best fitness
```

B. Architecture of MetaDE

Atop the proposed PDE, this subsection further introduces the architecture of MetaDE. The main target of MetaDE is to evolve the parameters of PDE through an external DE, empowering PDE to identify optimal parameters tailored to the target problem.

As illustrated in Fig. 2, MetaDE is structured with a two-tiered optimization architecture. The upper tier, termed the `evolver`, leverages DE to evolve the parameters of PDE. In contrast, the lower tier consists of a collection of `executors` that each run the parameterized PDE instance

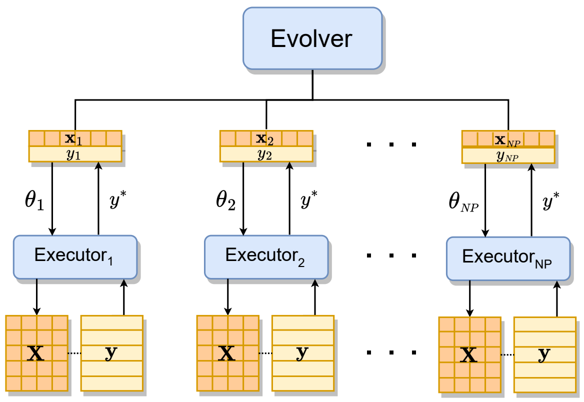


Fig. 2: Architecture of MetaDE. Within this architecture, a conventional DE algorithm operates as an `evolver`, where its individual $\mathbf{x}_i$ represents a distinct parameter configuration θ_i. These configurations are relayed to PDE to instantiate diverse DE variants as the `executors`. Each `executor` then evolves its distinct population and returns the best fitness y^* as identified, which is subsequently set as the fitness of $\mathbf{x}_i$.

to optimize the objective function. Every individual in the `evolver`, represented as $\mathbf{x}_i$, is decoded into a parameter configuration $\boldsymbol{\theta}_i$ with six elements: F, CR, bl, br, dn, and cs. For the evaluation of each individual, the configuration $\boldsymbol{\theta}_i$ is directed to its respective `executor` PDE$_i$ for objective function optimization. The final fitness y^* as identified by each `executor`, is subsequently set as the fitness of the corresponding $\mathbf{x}_i$ individual.

The architecture of MetaDE is streamlined for simplicity. Building upon this architecture, MetaDE integrates two tailored components: the *one-shot evaluation method* and the *power-up strategy*. These components further enhance the adaptability and efficiency of the `executors`, thereby elevating the overall performance of MetaDE.

C. One-shot Evaluation Method

Within the context of an `executor` driven by DE itself, the inherent stochastic nature can lead to variability in the optimal fitness values returned. Historically, several evaluation techniques, such as repeated evaluation [53], F-racing [54], and intensification [55], have been put forth to tackle this inconsistency. Yet, these often come at the cost of an exorbitant number of functional evaluations (FEs). To address this issue, we introduce the one-shot evaluation method.

Specifically, the method mandates each `executor` to undertake a singular, comprehensive independent run, subsequently returning its best-found solution. A distinguishing aspect of this method is the consistent allocation of the same initial random seed to every `executor`. As the algorithm progresses, this uniform seed ensures that the PDE fine-tunes its parameters in a consistent manner, thereby identifying optimal parameters tailored to the given seed environment. Essentially, this strategy embeds the seed as an integral facet of the problem domain.

D. Power-up Strategy

During the independent runs of an `executor`, the allocation of FEs plays a pivotal role in determining both the quality of solutions and computational efficiency. Allocating an excessive number of FEs indiscriminately can lead to undue computational resource consumption without necessarily improving solution quality. To address this issue, we propose the power-up strategy.

The essence of this strategy is dynamic FE allocation: while earlier iterations receive a moderate number of FEs to ensure resource efficiency, a more generous allocation (fivefold) is reserved for the terminal iteration within the evolutionary process. This strategy ensures that the `executor` has the resources for a thorough and comprehensive evaluation during its most crucial phase – the final generation of the `evolver`.

E. Implementation

As outlined in Algorithm 3, MetaDE draws its simple algorithmic workflow from conventional DE. MetaDE adopts `DE/rand/1/bin` as the `evolver`. The initialization phase (Line 1) spawns the MetaDE population within the parameter boundaries $[\mathbf{lb}, \mathbf{ub}]$. During the evaluation phase (Lines 6-11), each individual is decoded into a parameter blueprint and directed to an independent PDE instance (`executor`) for problem resolution. Running for a predetermined iteration count G', each `executor` subsequently reports the best fitness. Notably, Line 10 encapsulates the essence of the power-up strategy: for MetaDE's concluding iteration ($g == G_{max}$), the evaluation quota is amplified to $5 \times G'$ for the `executors`.

Algorithm 3 MetaDE

```
Input: $D$, $NP$, $G_{max}$, $\mathbf{lb}$ (lower boundaries of PDE's parameters),
    $\mathbf{ub}$ (upper boundaries of PDE's parameters), $NP'$ (population
    size of PDE), $G'$ (max generations of PDE)
 1: Initialize population $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{NP}\}$ between $[\mathbf{lb}, \mathbf{ub}]$
 2: Initialize the fitness of $\mathbf{X}$: $\mathbf{y} = \mathbf{inf}$
 3: $g = 0$
 4: while $g \le G_{max}$ do
 5:     Generate trial vectors $\mathbf{U}$ by mutation and crossover
        /* The mutation and crossover scheme used is rand/1/bin */
 6:     Decode each trial vector $\mathbf{u}$ into parameters:
        $F = \mathbf{u}[1], CR = \mathbf{u}[2], bl = \text{floor}(\mathbf{u}[3]), br = \text{floor}(\mathbf{u}[4]),$
        $dn = \text{floor}(\mathbf{u}[5]), cs = \text{floor}(\mathbf{u}[6])$
 7:     if $g < G_{max}$ then
 8:         $\mathbf{y} = \text{PDE}(D, NP', G', F, CR, bl, br, dn, cs)$
 9:     else
10:         $\mathbf{y} = \text{PDE}(D, NP', 5 * G', F, CR, bl, br, dn, cs)$
11:     end if
12:     Make selection between $\mathbf{U}$ and $\mathbf{X}$
13:     $g = g + 1$
14: end while
15: return the best individual and fitness
```

Evidently, the algorithmic design of MetaDE provides an automated end-to-end approach to black-box optimization. However, the computational demands of MetaDE, particularly in terms of FEs, cannot be understated. In historical computational contexts, such intensive demands might have posed significant impediments. Fortunately, contemporary advancements in computational infrastructures, coupled with the ubiquity of high-performance computational apparatuses such as GPUs, have substantially alleviated such a challenge.

Hence, we leverage the GPU-accelerated framework of EvoX [19] for the implementation of MetaDE. Thanks to the inherently parallel nature of MetaDE, computational tasks can be judiciously delegated to GPUs to engender optimized runtime performance. Specifically, the parallelism in MetaDE manifests in three distinct facets:

- **Parallel Initialization and Execution:** The multiple `executors` are instantiated and operated concurrently, each tailored by a unique parameter configuration derived from the MetaDE ensemble. This simultaneous operation enables comprehensive exploration across varied parameter landscapes.
- **Parallel Offspring Generation:** Both the `evolver` and `executors` adhere to parallel strategies for offspring inception. By synchronizing and coordinating mutations and crossover operations in their respective populations, MetaDE is able to rapidly produce offspring, thereby accelerating the evolutionary process.
- **Parallel Fitness Evaluations:** Each `executor` conducts fitness evaluations concurrently across its member individuals. Given the substantial number of the individuals within the populations of the `executors`, this parallel strategy significantly enhances the overall efficiency of MetaDE.

MetaDE adheres rigorously to the functional programming paradigm, capitalizing on automatic vectorization for parallel execution. Core algorithmic components, including crossover, mutation, and evaluation, are constructed using pure functions. Subsequently, the entire program is mapped to a GPU-based computation graph, ushering in accelerated processing. EvoX's adept state management ensures a seamless transfer of the algorithm's prevailing state, encompassing aspects like population, fitness, hyperparameters, and auxiliary data.

Listing 1 elucidates a representative implementation of MetaDE underpinned by the EvoX framework, which is meticulously segmented into four salient phases:

- **Initialization**: Herein, primary entities like the `evolver` (employing the traditional DE) and the `executor` (utilizing the proposed PDE) are instantiated. Concurrently, the target optimization problem is defined.
- **Meta Problem Transformation**: Within this phase, the original optimization problem is transformed to align with the meta framework. This metamorphosis is realized via the **`MetaProblem`** class, where the evaluation function undergoes vectorization, priming it for efficient batch assessments and facilitating concurrent evaluations of manifold configurations.
- **Computing Workflow Creation**: Post transformation, the workflow is architected to seamlessly amalgamate the initialized components. The `batch_executor` is crafted for batched operations of DE variants, and the **`MetaProblem`** is instantiated therewith. The holistic workflow, embodied by the `UniWorkflow` class, is then constructed, weaving together the `evolver`, the transformed problem, and a (`decoder`) which transforms the `evolver`'s population into specific hyperparameters for instantiating the DE variant of each `executor`.
- **Execution**: Having established the groundwork, MetaDE's execution phase is triggered, autonomously driving the computing workflow across distributed GPUs. This workflow is traversed iteratively, culminating once a predefined termination criterion is met.

```python
from evox import algorithms, problems, ...

### Initialization ###
evolver = algorithm.DE() # specify evolver
executor = algorithm.PDE() # specify executor
problem = ... # specify optimization problem

### Meta Problem Transformation ###
class MetaProblem(Problem):
   def __init__(self, batch_executor, ... ):
      # vectorize fitness evaluations
      self.batch_evaluate =
          vectorize(vectorize(problem.evaluate))

   def evaluate(self, state, ...):
      ...
      # run executors
      while ...:
         ...
         batch_fits, ... =
             self.batch_evaluate(...)
      # return fitness
      return min(min(batch_fits))

### Computing Workflow Creation ###
batch_executor = create_batch_executor(...)
meta_problem = MetaProblem(batch_executor, ...)
workflow = workflow.UniWorkflow(
         algorithm = evolver,
         pop_transform = decoder,
         problem = meta_problem,
      )

### Execution ###
while ...:
   state = workflow.step(state)
```

Listing 1: Demonstrative implementation of MetaDE leveraging the computational workflow of EvoX. The implementation is distinctly divided into four pivotal components: Workflow Initialization, Meta Problem Transformation, Computing Workflow Creation, and Execution.

IV. Experimental Study

In this section, we conduct detailed experimental assessments of MetaDE's capabilities. First, we comprehensively benchmark MetaDE against several representative DE variants and CEC2022 top algorithms to gauge its relative performance on the CEC2022 benchmark suite [56]. Then, we investigate the optimal DE variants obtained by MetaDE in the benchmark experiment. Finally, we apply MetaDE to robot control tasks. All experiments were conducted on a system equipped with an Intel Core i9-10900X CPU and an NVIDIA RTX 3090 GPU. For GPU acceleration, all the algorithms and test functions were implemented within EvoX [19].

A. Benchmarks against Representative DE Variants

1) Experimental Setup: The CEC2022 benchmark suite for single-objective black-box optimization was utilized for this study. This suite includes basic ($F_1 - F_5$), hybrid ($F_6 - F_8$), and composition functions ($F_9 - F_{12}$), catering to various optimization characteristics such as unimodality/multimodality and separability/non-separability.

For benchmark comparisons, we selected seven representative DE variants: DE (rand/1/bin) [4], SaDE [15], JaDE [16], CoDE [17], SHADE [28], LSHADE-RSP [31], and EDEV [37], which encapsulate a spectrum of mutation, crossover, and adaptation strategies. All algorithms were reimplemented using EvoX, with each capable of running in parallel, including the concurrent evaluation and reproduction. Their respective descriptions are as follows:

- DE/rand/1/bin is a foundational DE variant, which leverages a random mutation strategy coupled with binomial crossover.
- SaDE maintains an archive for tracking successful strategies and CR values and exhibits adaptability in strategy selection and parameter adjustments throughout the optimization process.
- JaDE relies on the current-to-pbest mutation strategy and dynamically adjusts its F and CR parameters during the optimization trajectory.
- CoDE infuses generational diversity by composing three disparate strategies, each complemented with randomized parameters, for offspring generation.
- SHADE employs the current-to-pbest mutation strategy and integrates a success-history mechanism to fine-tune its F and CR parameters adaptively.
- LSHADE-RSP, as one of the most competitive DE variants, employs delicate strategies such as linear population size reduction and ranking-based mutation.
- EDEV adopts a distributed framework that ensembles three classic DE variants: JaDE, CoDE, and EPSDE.

The population size for all comparative algorithms was uniformly set to 100, except for experiments involving large populations. The other parameters for these algorithms were adopted as per their default settings described in their respective publications.

In our MetaDE configuration, on one hand, the evolver had a population size of 100 and adopted the vanilla rand/1/bin strategy with $F = 0.5$ and $CR = 0.9$; On the other hand, each executor maintained a population of 100, iterating 1000 times for all the problems. For simplicity, any result exceeding the precision of 10^{-8} was truncated to 0. All statistical results were obtained via 31 independent runs[1].

2) Performance under Equal Wall-clock Time: In this part, we set equal wall-clock time (60 s) as the termination condition for running each test. This approach aligns with the practical constraints of modern GPU computing, where execution time serves as a more meaningful and comparable measure of performance across algorithms. Since all algorithms in our experiments are implemented with GPU parallelism, this setup ensures fairness by standardizing the computational resources and focusing on efficiency within the same time budget.

[1]Full results, including the statistical results applying Wilcoxon rank-sum tests with a a significance level of 0.05, can be found in the Supplementary Document.

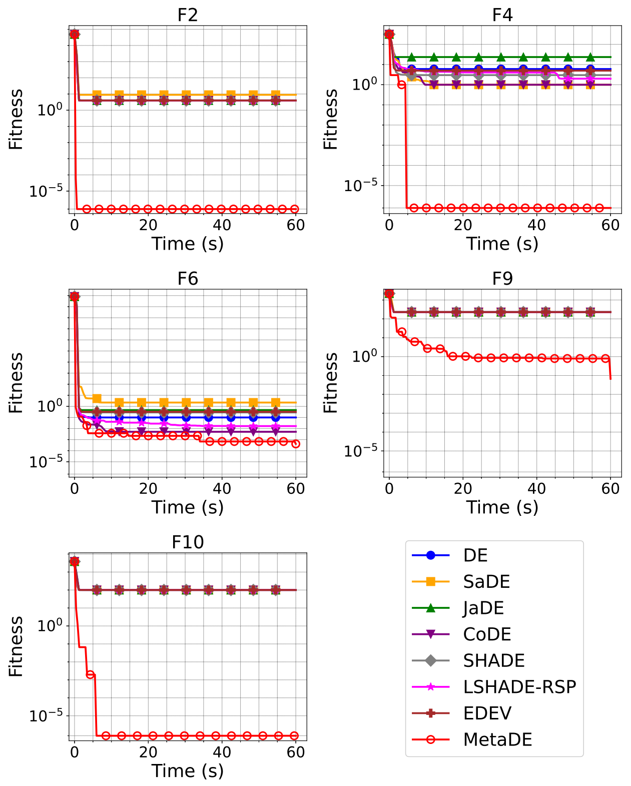


Fig. 3: Convergence curves on 10D problems in CEC2022 benchmark suite. The peer DE variants are set with population size of 100.

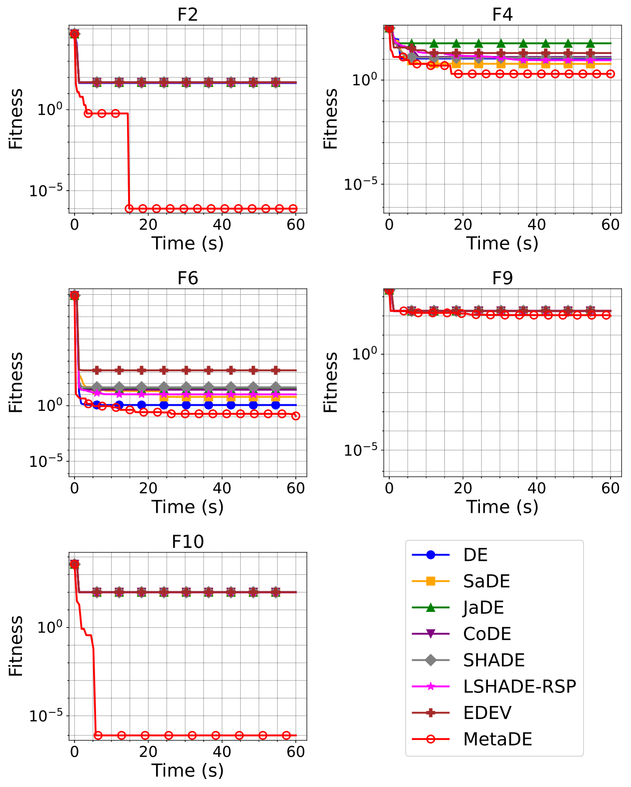


Fig. 4: Convergence curves on 20D problems in CEC2022 benchmark suite. The peer DE variants are set with population size of 100.

As shown in Figs. 3 and 4, we selected five challenging problems, specifically F_2, F_4, F_6, F_9, and F_{10}, to demonstrate the convergence profiles. Notably, MetaDE's convergence curve is observably more favorable, consistently registering lower errors than its counterparts across the majority of the problems. Particularly, on F_2, F_4, F_9, and F_{10}, MetaDE exhibits resilience against local optima entrapment and subsequent convergence stagnation. This is attributed to MetaDE's capability to identify optimal algorithm settings tailored for diverse problems, rather than merely tweaking parameters based on isolated segments of the optimization trajectory, as is the case with some DE variants. An intriguing characteristic of MetaDE's convergence, evident in functions like F_9 (refer to Fig. 3), is its pronounced performance surge in the optimization's terminal phase. This enhancement can be linked to MetaDE's power-up strategy of allocating bonus computational resources in its final phase (as per Line 10 of Algorithm 3).

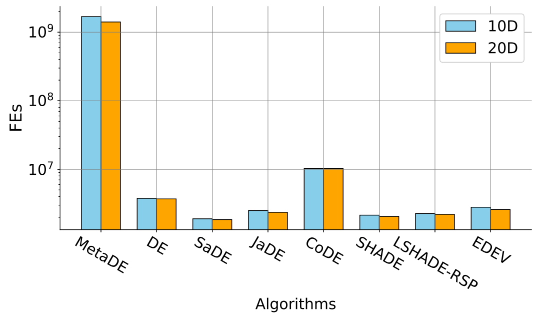


Fig. 5: The number of FEs achieved by each algorithm within 60 s. The results are averaged on all 10D and 20D problems in the CEC2022 benchmark suite.

Furthermore, to assess the concurrency of the algorithms, the number of FEs achieved by each algorithm within 60 seconds is shown in Table II and Fig 5. The results indicate that MetaDE achieves approximately 10^9 FEs within 60 seconds, while the other algorithms manage to attain only around 10^7 FEs in the same time frame. The results demonstrate the high concurrency of MetaDE, which is particularly favorable in GPU computing.

TABLE II
THE NUMBER OF FEs ACHIEVED BY EACH ALGORITHM WITHIN 60 s.

Dim	Func	MetaDE	DE	SaDE	JaDE	CoDE	SHADE	LSHADE-RSP	EDEV
10D	F_1	**1.85E+09**	4.28E+06	1.96E+06	2.55E+06	1.09E+07	2.24E+06	2.57E+06	2.79E+06
	F_2	**1.84E+09**	4.19E+06	1.89E+06	2.42E+06	1.11E+07	2.18E+06	2.58E+06	2.82E+06
	F_3	**1.50E+09**	4.00E+06	1.89E+06	2.50E+06	1.11E+07	2.10E+06	2.46E+06	2.68E+06
	F_4	**1.84E+09**	4.11E+06	2.02E+06	2.61E+06	1.11E+07	2.15E+06	2.60E+06	2.88E+06
	F_5	**1.83E+09**	4.13E+06	2.00E+06	2.60E+06	1.15E+07	2.17E+06	2.53E+06	2.96E+06
	F_6	**1.84E+09**	4.31E+06	1.96E+06	2.65E+06	1.07E+07	2.14E+06	2.55E+06	2.95E+06
	F_7	**1.74E+09**	3.35E+06	1.90E+06	2.55E+06	9.96E+06	2.14E+06	2.41E+06	2.87E+06
	F_8	**1.72E+09**	3.34E+06	1.83E+06	2.53E+06	9.60E+06	2.17E+06	2.34E+06	2.74E+06
	F_9	**1.78E+09**	3.35E+06	1.84E+06	2.52E+06	9.69E+06	2.18E+06	2.44E+06	2.82E+06
	F_{10}	**1.44E+09**	3.32E+06	1.83E+06	2.46E+06	9.00E+06	2.12E+06	2.30E+06	2.70E+06
	F_{11}	**1.46E+09**	3.55E+06	1.88E+06	2.34E+06	9.66E+06	2.13E+06	2.34E+06	2.63E+06
	F_{12}	**1.43E+09**	3.46E+06	1.83E+06	2.41E+06	9.51E+06	2.09E+06	2.32E+06	2.67E+06
20D	F_1	**1.66E+09**	4.32E+06	1.92E+06	2.46E+06	1.13E+07	2.21E+06	2.68E+06	2.80E+06
	F_2	**1.66E+09**	3.91E+06	1.88E+06	2.37E+06	1.17E+07	2.09E+06	2.66E+06	2.74E+06
	F_3	**1.18E+09**	3.76E+06	1.77E+06	2.33E+06	9.75E+06	1.95E+06	2.62E+06	2.64E+06
	F_4	**1.65E+09**	3.50E+06	1.87E+06	2.28E+06	1.07E+07	2.00E+06	2.63E+06	2.80E+06
	F_5	**1.64E+09**	3.57E+06	1.86E+06	2.32E+06	1.07E+07	2.05E+06	2.55E+06	2.74E+06
	F_6	**1.64E+09**	3.89E+06	1.90E+06	2.34E+06	1.16E+07	2.09E+06	2.62E+06	2.80E+06
	F_7	**1.45E+09**	4.15E+06	1.84E+06	2.36E+06	1.01E+07	1.99E+06	2.32E+06	2.80E+06
	F_8	**1.44E+09**	3.42E+06	1.82E+06	2.31E+06	9.51E+06	2.12E+06	2.18E+06	2.30E+06
	F_9	**1.57E+09**	3.30E+06	1.77E+06	2.33E+06	9.48E+06	2.03E+06	2.59E+06	2.75E+06
	F_{10}	**9.80E+08**	3.67E+06	1.82E+06	2.43E+06	1.01E+07	2.08E+06	2.09E+06	2.35E+06
	F_{11}	**1.00E+09**	3.51E+06	1.95E+06	2.41E+06	9.81E+06	2.07E+06	2.17E+06	2.37E+06
	F_{12}	**9.90E+08**	3.44E+06	1.85E+06	2.37E+06	9.51E+06	2.07E+06	2.17E+06	2.39E+06

3) Performance under Equal FEs: In the preceding part, the performance benchmarking of MetaDE with other algorithms was anchored to equal wall-clock durations. However, to ensure a comprehensive assessment, it is imperative to evaluate their performances under equivalent FEs. In this part, we run each algorithm using the FEs achieved by MetaDE in 60 s (i.e., $1.84 \times 10^9/1.66 \times 10^9$, $1.84 \times 10^9/1.64 \times 10^9$, and $1.44 \times 10^9/9.8 \times 10^8$) on F_2, F_6, and F_{10} for 10D/20D cases. These selected functions collectively epitomize the basic, hybrid, and composition challenges within the CEC2022 benchmark suite.

As summarized in Table III, MetaDE consistently demonstrates the best performance, even when other algorithms are endowed with comparable FEs. The reason can be traced to the inherent stagnation tendencies of other algorithms: after a certain point, additional FEs may not contribute to performance improvements. This behavioral pattern is also lucidly captured in the convergence curves as presented in Figs. 3 and 4.

Another noteworthy observation is the extended computation time required for a singular run of the comparison algorithms under these enhanced FEs, often extending to several hours or even transcending a day (e.g., running a single run of DE can take up to seven hours). This elongated computational span can largely be attributed to their low concurrency, which struggles to benefit the parallelism of GPU computing.

4) Performance with Large Populations: Since a large population size could potentially increase the concurrency of fitness evaluations, for rigorousness, we further investigate the performance of the algorithms with large populations.

Specifically, MetaDE adopted the same population size setting as in previous experiments (i.e., 100 for both `evolver` and `executor`), while the population size of the other DE variants was increased to 1,000. This adjustment significantly enhances the concurrency of the other DE variants when utilizing GPU accelerations, thereby preventing insufficient convergence.

As evidenced in Figs. 6-7, MetaDE still outperforms the other DE variants across all problems. However, the performances of the other DE variants did not show significant improvements, which can be attributed to two factors. First, since the conventional DE variants were not tailored for large populations, simply enlarging the populations may not help. Second, since the sorting and archiving operations in some DE variants (e.g., SaDE) suffer from high computational complexities related to the population size, enlarging the populations brings additional computation overheads, thus limiting their performances under fixed wall-clock time.

By contrast, the large population in MetaDE is delicately organized in a *hierarchical* manner: the `executor` maintains a population of moderate size, with each individual initializing an `executor` with a normal population. This strategy not only capitalizes on the small-population advantage of conventional DE, but also benefits the concurrency brought by large populations.

TABLE III
COMPARISONS BETWEEN METADE AND OTHER DE VARIANTS UNDER EQUAL FES. THE MEAN AND STANDARD DEVIATION (IN PARENTHESES) OF THE RESULTS OVER MULTIPLE RUNS ARE DISPLAYED IN PAIRS. RESULTS WITH THE BEST MEAN VALUES ARE HIGHLIGHTED.

Func		MetaDE	DE	SaDE	JaDE	CoDE	SHADE	LSHADE-RSP	EVDE
10D	F_2	**0.00E+00 (0.00E+00)**	4.52E+00 (2.36E+00)−	6.85E+00 (3.52E+00)−	6.31E+00 (3.05E+00)−	5.78E+00 (2.37E+00)−	4.33E+00 (3.81E+00)−	2.35E+00 (3.44E+00)−	5.86E+00 (2.99E+00)−
	F_6	**5.50E-04 (3.96E-04)**	1.13E-01 (7.83E-02)−	3.54E+01 (1.11E+02)−	2.02E+00 (3.34E+00)−	6.96E-03 (5.99E-03)−	9.27E-01 (1.31E+00)−	3.10E-02 (4.83E-02)−	1.46E+00 (2.76E+00)−
	F_{10}	**0.00E+00 (0.00E+00)**	1.00E+02 (4.40E-02)−	1.00E+02 (6.10E-02)−	1.21E+02 (4.35E+01)−	1.00E+02 (6.88E-02)−	1.29E+02 (4.67E+01)−	1.09E+02 (2.87E+01)−	1.10E+02 (3.05E+01)−
20D	F_2	**1.26E-02 (3.74E-02)**	4.72E+01 (2.09E+00)−	4.76E+01 (2.02E+00)−	1.34E+01 (2.19E+01)−	4.91E+01 (1.70E-06)−	4.91E+01 (3.40E-06)−	4.84E+01 (1.62E+00)−	4.47E+01 (1.41E+01)−
	F_6	**1.16E-01 (2.79E-02)**	7.28E-01 (5.22E-01)−	3.20E+01 (1.61E+01)−	4.90E+01 (3.31E+01)−	2.26E+01 (1.80E+01)−	5.67E+01 (3.90E+01)−	1.15E+01 (8.38E+00)−	2.92E+03 (5.82E+03)−
	F_{10}	**0.00E+00 (0.00E+00)**	1.07E+02 (2.07E+01)−	1.00E+02 (2.72E-02)−	1.01E+02 (3.64E-02)−	1.00E+02 (3.71E-02)−	1.43E+02 (5.64E+01)−	1.21E+02 (4.52E+01)−	1.14E+02 (4.65E+01)−
$+ / \approx / -$		–	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6

* The Wilcoxon rank-sum tests (with a significance level of 0.05) were conducted between MetaDE and each algorithm individually. The final row displays the number of problems where the corresponding algorithm performs statistically better (+), similar (≈), or worse (−) compared to MetaDE.

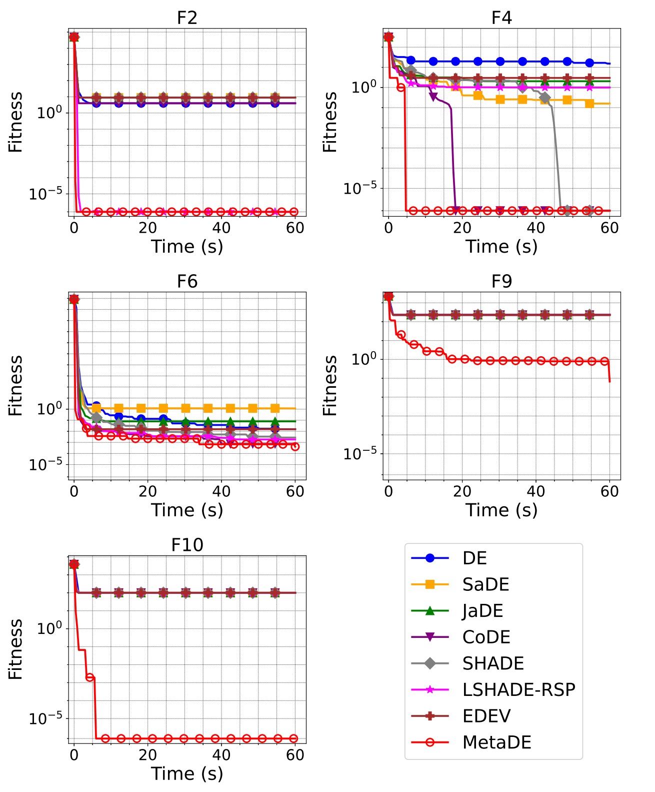


Fig. 6: Convergence curves on 10D problems in CEC2022 benchmark suite. The peer DE variants are set with population size of 1,000.

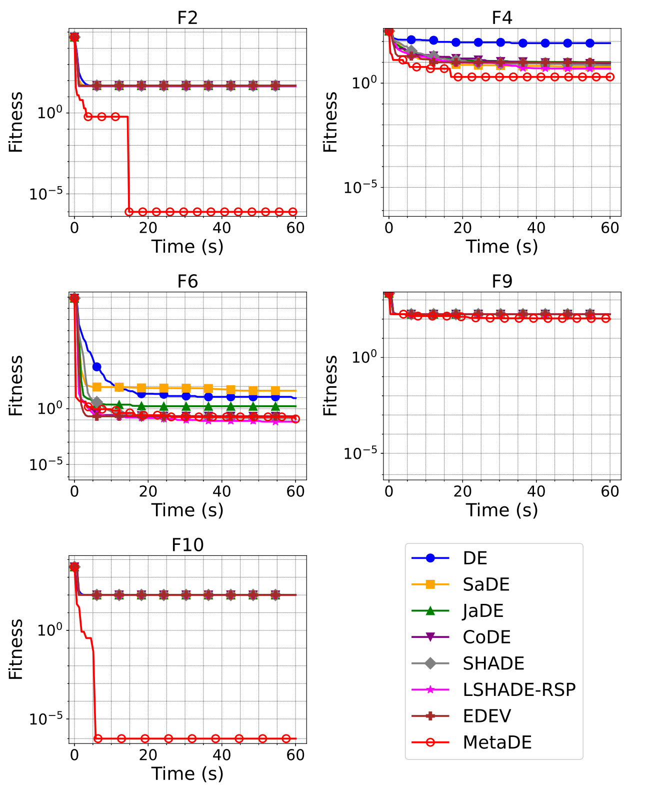


Fig. 7: Convergence curves on 20D problems in CEC2022 benchmark suite. The peer DE variants are set with population size of 1,000.

B. Comparisons with Top Algorithms in CEC2022 Competition

To further assess the performance of MetaDE, we compare it with the top 4 algorithms from the CEC2022 Competition on Single Objective Bound Constrained Numerical Optimization[2]. For each algorithm, we set equal FEs as achieved by MetaDE within 60 seconds (refer to Table II for details).

The top 4 algorithms from the CEC2022 Competition are EA4eig [57], NL-SHADE-LBC [58], NL-SHADE-RSP-MID [59], and S-LSHADE-DP [60]:

- EA4eig combines the strengths of four evolutionary algorithms (CMA-ES, CoBiDE, an adaptive variant of jSO, and IDE) using Eigen crossover.
- NL-SHADE-LBC is a dynamic DE variant that integrates linear bias changes for parameter adaptation, repeated point generation to handle boundary constraints, non-linear population size reduction, and a selective pressure mechanism.
- NL-SHADE-RSP-MID is an advanced version of NL-SHADE-RSP, which estimates the optimum using the population midpoint, incorporates a restart mechanism, and improves boundary constraint handling.
- S-LSHADE-DP focuses on maintaining population diversity through dynamic perturbation, adjusting noise intensity to enhance exploration.

[2]https://github.com/P-N-Suganthan/2022-SO-BO

The experimental results are summarized in Tables IV and V. On 10D problems, MetaDE outperforms EA4eig, NL-SHADE-LBC, and NL-SHADE-RSP, while achieving comparable performance to S-LSHADE-DP. On 20D problems,

TABLE IV
COMPARISONS BETWEEN METADE AND THE TOP 4 ALGORITHMS FROM CEC2022 COMPETITION (10D). THE MEAN AND STANDARD DEVIATION (IN PARENTHESES) OF THE RESULTS OVER MULTIPLE RUNS ARE DISPLAYED IN PAIRS. RESULTS WITH THE BEST MEAN VALUES ARE HIGHLIGHTED.

Func	MetaDE	EA4eig	NL-SHADE-LBC	NL-SHADE-RSP	S-LSHADE-DP
F_1	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_2	**0.00E+00 (0.00E+00)**	7.97E-01 (1.78E+00)−	7.97E-01 (1.78E+00)−	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_3	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_4	**0.00E+00 (0.00E+00)**	9.95E-01 (1.22E+00)−	1.99E-01 (4.45E-01)−	2.98E+00 (1.15E+00)−	**0.00E+00 (0.00E+00)**≈
F_5	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_6	5.50E-04 (3.96E-04)	7.53E-04 (5.52E-04)≈	8.93E-02 (1.18E-01)−	4.37E-02 (5.41E-02)−	**5.84E-05 (4.73E-05)**+
F_7	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_8	5.52E-03 (4.41E-03)	1.01E-04 (1.66E-04)+	3.96E-04 (4.23E-04)+	3.13E-01 (3.60E-01)−	**1.26E-05 (1.56E-05)**+
F_9	**3.36E+00 (1.77E+01)**	1.86E+02 (0.00E+00)−	2.29E+02 (3.18E-14)−	8.03E+01 (1.08E+02)−	2.23E+02 (1.31E+01)−
F_{10}	**0.00E+00 (0.00E+00)**	1.00E+02 (0.00E+00)−	1.00E+02 (0.00E+00)−	1.56E-02 (3.12E-02)−	**0.00E+00 (0.00E+00)**≈
F_{11}	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_{12}	**1.39E+02 (4.63E+01)**	1.48E+02 (5.98E+00)−	1.65E+02 (0.00E+00)−	1.62E+02 (2.15E+00)−	1.59E+02 (0.00E+00)−
+ / ≈ / −	–	1/6/5	1/5/6	0/6/6	2/8/2

* The Wilcoxon rank-sum tests (with a significance level of 0.05) were conducted between MetaDE and each algorithm individually. The final row displays the number of problems where the corresponding algorithm performs statistically better (+), similar (≈), or worse (−) compared to MetaDE.

TABLE V
COMPARISONS BETWEEN METADE AND THE TOP 4 ALGORITHMS FROM CEC2022 COMPETITION (20D). THE MEAN AND STANDARD DEVIATION (IN PARENTHESES) OF THE RESULTS OVER MULTIPLE RUNS ARE DISPLAYED IN PAIRS. RESULTS WITH THE BEST MEAN VALUES ARE HIGHLIGHTED.

Func	MetaDE	EA4eig	NL-SHADE-LBC	NL-SHADE-RSP	S-LSHADE-DP
F_1	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_2	3.83E-04 (2.10E-03)	**0.00E+00 (0.00E+00)**+	4.91E+01 (0.00E+00)−	**0.00E+00 (0.00E+00)**+	**0.00E+00 (0.00E+00)**+
F_3	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_4	1.96E+00 (7.76E-01)	7.36E+00 (2.06E+00)−	**1.59E+00 (5.45E-01)**≈	1.07E+02 (1.54E+02)−	3.20E+00 (1.94E+00)−
F_5	**0.00E+00 (0.00E+00)**	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈	2.27E-01 (4.54E-01)−	**0.00E+00 (0.00E+00)**≈
F_6	**1.38E-01 (5.56E-02)**	2.54E-01 (4.28E-01)−	3.06E-01 (2.01E-01)−	2.08E-01 (9.78E-02)−	5.02E-01 (5.34E-01)−
F_7	8.42E-02 (1.01E-01)	1.37E+00 (1.10E+00)−	**6.24E-02 (1.40E-01)**≈	1.28E+00 (1.95E+00)−	9.83E-01 (8.12E-01)−
F_8	2.66E+00 (3.83E+00)	2.02E+01 (1.28E-01)−	**1.01E-01 (1.41E-01)**+	1.99E+01 (4.97E-01)−	2.30E-01 (1.82E-01)+
F_9	**1.32E+02 (3.43E+01)**	1.65E+02 (0.00E+00)−	1.81E+02 (0.00E+00)−	1.81E+02 (0.00E+00)−	1.81E+02 (0.00E+00)−
F_{10}	**0.00E+00 (0.00E+00)**	1.23E+02 (5.12E+01)−	1.00E+02 (9.27E-03)−	**0.00E+00 (0.00E+00)**≈	**0.00E+00 (0.00E+00)**≈
F_{11}	1.74E-03 (7.97E-03)	3.20E+02 (4.47E+01)−	3.00E+02 (0.00E+00)−	**0.00E+00 (0.00E+00)**+	**0.00E+00 (0.00E+00)**+
F_{12}	2.29E+02 (9.70E-01)	**2.00E+02 (2.04E-04)**+	2.37E+02 (3.17E+00)−	2.34E+02 (1.46E+00)−	2.34E+02 (4.51E+00)−
+ / ≈ / −	–	2/3/7	1/5/6	2/3/7	3/4/5

* The Wilcoxon rank-sum tests (with a significance level of 0.05) were conducted between MetaDE and each algorithm individually. The final row displays the number of problems where the corresponding algorithm performs statistically better (+), similar (≈), or worse (−) compared to MetaDE.

MetaDE consistently outperforms the four algorithms. An additional noteworthy observation is that S-LSHADE-DP exhibits promising performance under a large number of FEs.

C. Investigation of Optimal DE Variants

TABLE VI
OPTIMAL DE VARIANTS OBTAINED BY METADE ON EACH PROBLEM OF THE CEC2022 BENCHMARK SUITE. FDC AND RIE ARE TWO FITNESS LANDSCAPE CHARACTERISTICS THAT MEASURE THE DIFFICULTY AND RUGGEDNESS OF THE PROBLEM.

Problem		F	CR	Strategy	FDC	RIE
10D	F_6	0.70	0.99	`rand-to-pbest/1/arith`	0.61	0.81
	F_8	0.51	0.44	`pbest-to-best/1/bin`	0.27	0.62
	F_9	0.02	0.03	`current/2/bin`	0.08	0.82
	F_{12}	0.16	0.00	`current-to-best/4/bin`	-0.15	0.78
20D	F_4	0.13	0.71	`rand-to-best/3/bin`	0.90	0.79
	F_6	0.67	0.99	`pbest-to-rand/1/bin`	0.48	0.80
	F_7	0.27	0.93	`rand/2/bin`	0.26	0.78
	F_8	0.65	0.00	`pbest/1/exp`	0.12	0.40
	F_9	0.06	0.00	`current/2/bin`	-0.17	0.84
	F_{12}	0.33	0.44	`rand-to-best/2/bin`	-0.16	0.85

This part provides an in-depth examination of the optimal DE variants obtained by MetaDE in Section IV-A2, as summarized in Table VI. The optimal parameters correspond to the best individual in the final population of MetaDE. The table only displays the optimal parameters for the ten listed problems, as the remaining problems are relatively simpler, with numerous DE variants capable of locating the optimal solutions of the problems. Furthermore, the optimal parameters presented in the table represent the best results of MetaDE derived from the finest run out of 31 independent trials.

All the problems in the table are characterized by both multimodality and non-separability. Additionally, to further depict the characteristics of the problems' fitness landscapes, we computed both the fitness distance correlation (FDC) [61] and the ruggedness of information entropy (RIE) [62]; the former measures the complexity (difficulty) of the problems, while the latter characterizes the ruggedness of the landscape.

Analyzing the obtained data, it is evident that no single set of parameters or strategies consistently excels across all problems. Parameters such as F and CR exhibit variability across problems without adhering to a specific trend. Similarly, the selection of base vectors (bl and br) does not show a uniform preference either. Regarding the fitness landscape characteristics of each problem, the selection of parameters exhibits distinct patterns. The FDC indicates problem complexity; with simpler problems (higher FDC), such as 10-

dimensional F_6, F_8 and 20-dimensional F_4, F_6, F_7, a larger CR value is favored. Conversely, smaller CR values are chosen for problems with lower FDC. A larger CR tends to facilitate convergence, whereas a CR close to 0 leads to offspring that change incrementally, dimension by dimension. However, the other characteristic, RIE, does not seem to have a clear association with parameter choices. The optimal strategies for identical problems across different dimensions exhibit closeness, with F_8, F_9, and F_{12} demonstrating notably parallel strategies between their 10D and 20D problems. In terms of crossover strategies (cs), it seems to have a preference for binomial crossover. This aligns with the traditional DE configurations.

These observations align with the No Free Lunch (NFL) theorem [10], thus underscoring the importance of distinct optimization strategies tailored for diverse problems. Conventionally, the optimization strategies have oscillated between seeking a generalist set of parameters for broad applicability and a specialist set tailored for specific problems. However, the dynamic nature of optimization problems, where even minute changes like a different random seed can pivot the problem's dynamics, highlights the challenges of a generalist approach. In contrast, MetaDE provides a simple yet effective approach, showing promising generality and adaptability.

D. Application to Robot Control

In this experiment, we demonstrate the extended application of MetaDE to robot control. Specifically, we adopted the evolutionary reinforcement learning paradigm [63] as illustrated in Fig. 8. The experiment was conducted on Brax [64] for robotics simulations with GPU acceleration.

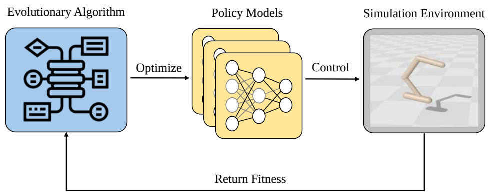


Fig. 8: Illustration of robot control via evolutionary reinforcement learning. The evolutionary algorithm optimizes the parameters of a population of candidate policy models for controlling the robotics behaviors. The simulation environment returns rewards achieved by the candidate policy models to the evolutionary algorithm as fitness values.

This experiment involved three robot control tasks: "swimmer", "hopper", and "reacher". As summarized in Table VII, we adopted similar policy models for these three tasks, each consisting of a multilayer perceptron (MLP) with three fully connected layers, but with different input and output dimensions. Consequently, the three policy models comprise 1410, 1539, and 1506 parameters for optimization respectively, where the optimization objective is to achieve maximum reward of each task. MetaDE, vanilla DE [1], SHADE [28], LSHADE-RSP [31], EDEV [37], CSO [65][3], and CMA-ES [66] were applied as the optimizer respectively.

[3]The competitive swarm optimizer (CSO) is a tailored PSO variant for large-scale optimization.

The iteration count for PDE within MetaDE was set to 50, while other algorithms maintained a population size of 100. Each algorithm was run independently 15 times. Considering the time-intensive nature of the robotics simulations, we set 60 minutes as the termination condition for each run.

TABLE VII
NEURAL NETWORK STRUCTURE OF THE POLICY MODEL FOR EACH ROBOT CONTROL TASK

Task	D	Input	Hidden Layers	Output	Overview of objectives
Hopper	1539	11	32×32	3	balance and jump
Swimmer	1410	8	32×32	2	maximizing movement
Reacher	1506	11	32×32	2	precise reaching

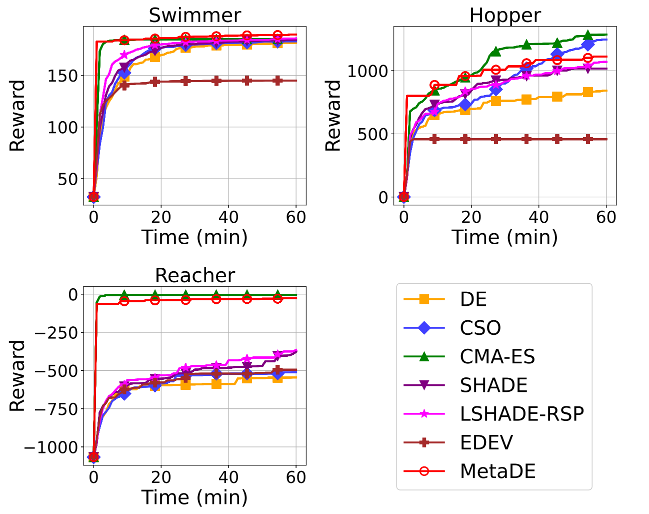


Fig. 9: The reward curves achieved by MetaDE and peer evolutionary algorithms when applied to each robot control task.

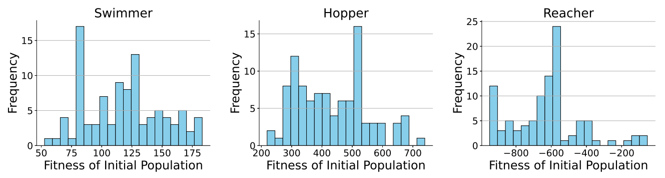


Fig. 10: The fitness distribution of MetaDE's initial population when applied to each robot control task.

As shown in Fig. 9, it is evident that MetaDE achieves the best performance in the Swimmer tasks, while slightly outperformed by CMA-ES and CSO in the Hopper and Reacher task. An interesting observation from the reward curves is that MetaDE almost reaches optimality nearly at the first generation and does not show further significant improvements thereafter. To elucidate this phenomenon, Fig. 10 provides the fitness distribution of MetaDE's initial population, indicating that MetaDE harbored several individuals with considerably high fitness from the initial generation. In other words, MetaDE was able to generate high-performance DE variants for these problems even by random sampling. This can be attributed to the unique nature of neural network optimization. As widely acknowledged, the neural network optimization typically features numerous plateaus in the fitness landscape, thus making it relatively easy to find one of the local optima. MetaDE provides unbiased sampling of parameter settings for generating diverse DE variants. Even without further evolution, some of the randomly sampled DE variants are very likely to reach the plateaus. In contrast, the other algorithms are specially tailored with biases; in such large-scale optimization scenarios, the biases can be further amplified, thus making them ineffective.

V. Conclusion

In this paper, we introduced MetaDE, a method that leverages the strengths of DE not only to address optimization tasks but also to adapt and refine its own strategies. This meta-evolutionary approach demonstrates how DE can autonomously evolve its parameter configurations and strategies. Our experiments demonstrate that MetaDE has robust performance across various benchmarks, as well as the application in robot control through evolutionary reinforcement learning. Nevertheless, the study also emphasizes the complexity of finding universally optimal parameter configurations. The intricate balance between generalization and specialization remains a challenge, and MetaDE has shed light on further research into self-adapting algorithms. We anticipate that the insights gained from this work will inspire the development of more advanced meta-evolutionary approaches, pushing the boundaries of evolutionary optimization in even more complex and dynamic environments.

References

[1] K. V. Price, "Differential evolution: a fast and simple numerical optimizer," in *Proceedings of North American Fuzzy Information Processing*. IEEE, 1996, pp. 524–527.

[2] R. Storn and K. Price, "Minimizing the real functions of the icec'96 contest by differential evolution," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, 1996, pp. 842–844.

[3] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of North American Fuzzy Information Processing*. IEEE, 1996, pp. 519–523.

[4] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, p. 341, 1997.

[5] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2010.

[6] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution–an updated survey," *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

[7] M. Pant, H. Zaheer, L. Garcia-Hernandez, A. Abraham *et al.*, "Differential evolution: A review of more than two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 90, p. 103479, 2020.

[8] J. Lampinen, I. Zelinka *et al.*, "On stagnation of the differential evolution algorithm," in *Proceedings of MENDEL*, vol. 6. Citeseer, 2000, pp. 76–83.

[9] F. Neri and V. Tirronen, "Recent advances in differential evolution: a survey and experimental analysis," *Artificial Intelligence Review*, vol. 33, pp. 61–106, 2010.

[10] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[11] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999.

[12] A. E. Eiben and S. K. Smit, "Evolutionary algorithm parameters and methods to tune them," in *Autonomous Search*. Springer, 2012, pp. 15–36.

[13] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 201–216, 2019.

[14] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[15] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2008.

[16] J. Zhang and A. C. Sanderson, "Jade: adaptive differential evolution with optional external archive," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[17] Y. Wang, Z. Cai, and Q. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 55–66, 2011.

[18] M. E. H. Pedersen, "Tuning & simplifying heuristical optimization," Ph.D. dissertation, University of Southampton, 2010.

[19] B. Huang, R. Cheng, Z. Li, Y. Jin, and K. C. Tan, "EvoX: A distributed gpu-accelerated framework for scalable evolutionary computation," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2024.

[20] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[21] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied Soft Computing*, vol. 11, no. 2, pp. 1679–1696, 2011.

[22] K. V. Price, "An introduction to differential evolution," in *New Ideas in Optimization*, 1999, pp. 79–108.

[23] H. A. Abbass, "The self-adaptive pareto differential evolution algorithm," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 1. IEEE, 2002, pp. 831–836.

[24] M. G. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution," in *International Conference on Computational and Information Science*. Springer, 2005, pp. 192–199.

[25] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, 2005, pp. 991–998.

[26] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing*, vol. 9, pp. 448–462, 2005.

[27] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Computing*, vol. 10, pp. 673–686, 2006.

[28] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 71–78.

[29] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 1658–1665.

[30] W. J. Yu, M. Shen, W. N. Chen, Z. H. Zhan, Y. J. Gong, Y. Lin, O. Liu, and J. Zhang, "Differential evolution with two-level parameter adaptation," *IEEE Transactions on Cybernetics*, vol. 44, no. 7, pp. 1080–1099, 2013.

[31] V. Stanovov, S. Akhmedova, and E. Semenkin, "Lshade algorithm with rank-based selective pressure strategy for solving cec 2017 benchmark problems," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8.

[32] K. M. Sallam, S. M. Elsayed, R. K. Chakrabortty, and M. J. Ryan, "Improved multi-operator differential evolution algorithm for solving unconstrained problems," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[33] X. Liu, J. Sun, Q. Zhang, Z. Wang, and Z. Xu, "Learning to learn evolutionary algorithm: A learnable differential evolution," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2023.

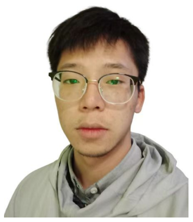
[34] M. Weber, V. Tirronen, and F. Neri, "Scale factor inheritance mechanism in distributed differential evolution," *Soft Computing*, vol. 14, pp. 1187–1207, 2010.

[35] M. Weber, F. Neri, and V. Tirronen, "A study on scale factor in distributed differential evolution," *Information Sciences*, vol. 181, no. 12, pp. 2488–2511, 2011.

[36] ——, "A study on scale factor/crossover interaction in distributed differential evolution," *Artificial Intelligence Review*, vol. 39, pp. 195–224, 2013.

[37] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, and P. N. Suganthan, "Ensemble of differential evolution variants," *Information Sciences*, vol. 423, pp.

172–186, 2018.
[38] G. Wu, R. Mallipeddi, P. N. Suganthan, R. Wang, and H. Chen, "Differential evolution with multi-population based ensemble of mutation strategies," *Information Sciences*, vol. 329, pp. 329–345, 2016.
[39] L. Tong, M. Dong, and C. Jing, "An improved multi-population ensemble differential evolution," *Neurocomputing*, vol. 290, pp. 130–147, 2018.
[40] Y. Ge, W. Yu, Y. Lin, Y. Gong, Z. Zhan, W. Chen, and J. Zhang, "Distributed differential evolution based on adaptive mergence and split for large-scale optimization," *IEEE Transactions on Cybernetics*, vol. 48, no. 7, pp. 2166–2180, 2018.
[41] J. Li, K. Du, Z. Zhan, H. Wang, and J. Zhang, "Distributed differential evolution with adaptive resource allocation," *IEEE Transactions on Cybernetics*, 2022.
[42] R. E. Mercer and J. Sampson, "Adaptive search using a reproductive meta-plan," *Kybernetes*, vol. 7, no. 3, pp. 215–228, 1978.
[43] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 2013.
[44] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, vol. 285, no. 2, pp. 405–428, 2020.
[45] F. Caraffini, F. Neri, and M. Epitropakis, "Hyperspam: A study on hyper-heuristic coordination strategies in the continuous domain," *Information Sciences*, vol. 477, pp. 186–202, 2019.
[46] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.
[47] D. B. Fogel, L. J. Fogel, and J. W. Atmar, "Meta-evolutionary programming," in *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems & Computers*. IEEE Computer Society, 1991, pp. 540–541.
[48] V. Nannen and A. E. Eiben, "Efficient relevance estimation and value calibration of evolutionary algorithm parameters," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 103–110.
[49] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2009, pp. 142–157.
[50] Z. Yuan, M. A. Montes de Oca, M. Birattari, and T. Stützle, "Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms," *Swarm Intelligence*, vol. 6, pp. 49–75, 2012.
[51] S. Luke and A. K. A. Talukder, "Is the meta-ea a viable optimization method?" in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, 2013, pp. 1533–1540.
[52] M. Indu *et al.*, "A meta-evolutionary selection of constituents in ensemble differential evolution algorithm," *Expert Systems with Applications*, vol. 205, p. 117667, 2022.
[53] Z. Yuan, M. A. Montes de Oca, M. Birattari, and T. Stützle, "Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms," *Swarm Intelligence*, vol. 6, pp. 49–75, 2012.
[54] M. Birattari and J. Kacprzyk, *Tuning metaheuristics: a machine learning perspective*. Springer, 2009, vol. 197.
[55] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.
[56] A. Kumar, K. V. Price, A. W. Mohamed, A. A. Hadi, and P. N. Suganthan, "Problem definitions and evaluation criteria for the cec 2022 special session and competition on single objective bound constrained numerical optimization," Tech. Rep., 2021.
[57] P. Bujok and P. Kolenovsky, "Eigen crossover in cooperative model of evolutionary algorithms applied to cec 2022 single objective numerical optimisation," in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 1–8.
[58] V. Stanovov, S. Akhmedova, and E. Semenkin, "Nl-shade-lbc algorithm with linear parameter adaptation bias change for cec 2022 numerical optimization," in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 01–08.
[59] R. Biedrzycki, J. Arabas, and E. Warchulski, "A version of nl-shade-rsp algorithm with midpoint for cec 2022 single objective bound constrained problems," in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2022, pp. 1–8.
[60] L. Van Cuong, N. N. Bao, N. K. Phuong, and H. T. T. Binh, "Dynamic perturbation for population diversity management in differential evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 391–394.
[61] T. Jones, S. Forrest *et al.*, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms." in *Proceedings of the 6th International Conference on Genetic Algorithms*, vol. 95, 1995, pp. 184–192.
[62] K. M. Malan and A. P. Engelbrecht, "Quantifying ruggedness of continuous landscapes using entropy," in *2009 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2009, pp. 1440–1447.
[63] H. Bai, R. Cheng, and Y. Jin, "Evolutionary reinforcement learning: A survey," *Intelligent Computing*, vol. 2, p. 0025, 2023. [Online]. Available: https://spj.science.org/doi/abs/10.34133/icomputing.0025
[64] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax - a differentiable physics engine for large scale rigid body simulation," 2021. [Online]. Available: http://github.com/google/brax
[65] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 191–204, 2014.
[66] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.



Minyang Chen received his B.S. degree in Automation from Guangxi Minzu University, Guangxi, China, in 2018, and his M.S. degree in Control Science and Engineering from East China University of Science and Technology, Shanghai, China, in 2022. He is currently pursuing a Ph.D. degree in Electrical and Electronic Engineering at the University of Glasgow, Glasgow, UK. From 2022 to 2023, he worked as a research assistant with the Department of Computer Science and Engineering at Southern University of Science and Technology, Shenzhen, China. His research interests include evolutionary computation and machine learning, particularly in large-scale optimization, differential evolution, and optimization in integrated circuits and antennas.

Chenchen Feng received the B.S. degree in Computer Science and Technology from the Southern University of Science and Technology, Shenzhen, China, in 2024, where he is currently pursuing the M.S. degree. His research interest is mainly focused on large-scale evolutionary computation.

Ran Cheng (Senior Member, IEEE) received the B.Sc. degree from the Northeastern University, Shenyang, China, in 2010, and the Ph.D. degree from the University of Surrey, Guildford, U.K., in 2016. He is currently an Associate Professor with the Department of Data Science and Artificial Intelligence, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. He is a recipient of the IEEE Transactions on Evolutionary Computation Outstanding Paper Award (2018 and 2021), the IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award (2019), the IEEE Computational Intelligence Magazine Outstanding Paper Award (2020), and the IEEE Computational Intelligence Society Early Career Award (2025). He is the Founding Chair of the IEEE Computational Intelligence Society Shenzhen Chapter. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, IEEE Transactions on Artificial Intelligence, IEEE Transactions on Emerging Topics in Computational Intelligence, and IEEE Transactions on Cognitive and Developmental Systems.