A Technical Details

A.1 Technical Details of Λ_θ

Within this paper, the aim of the landscape analysis is to profile the dynamic optimization status of the current optimization process. That is, given a d-dimensional target optimization problem f, at any time step t, the optimization process maintains a population of m candidate solutions $\{X_i^t \in \mathbb{R}^d\}_{i=1}^m$, and their corresponding objective values $\{y_i^t = f(X_i^t)\}_{i=1}^m$. We consider an end-to-end neural network structure that receives the candidates population and their corresponding objective values as input, and then outputs h-dimensional dynamic optimization status s_i^t for each X_i^t. This optimization status feature aggregates the information of the optimization problem and the current candidate population, hence can be used for dynamic landscape analysis in MetaBBO algorithms. We have to note that the core challenges in designing such a neural landscape analyser locate at: 1) **generalizability**: it should be able to handle optimization problems with different searching ranges and objective value scales; 2) **scalability**: it should be capable of computing the dynamic optimization status efficiently as the amount of the sampled candidates or the dimensions of the problem scales. We address the above two challenges by designing a two-stage attention based neural network structure as the landscape analyser (Λ_θ) in NeurELA. We now introduce the architecture of the Λ_θ and establish its overall computation graph step by step. For the convenience of writing, we omit superfix for time step t.

Pre-processing Module. To make NeurELA generalizable across different problems with various searching ranges and objective value scales, we apply min-max normalization over the searching space and the objective value space. Concretely, for a specific d-dimensional optimization problem f (suppose a minimization problem), we acquire its searching range $\{[lb^j, ub^j]\}_{j=1}^d$, where lb^j and ub^j represent the lower bound and the upper bound at j-th dimension. Then we normalize each X_i in the candidate population by $X_i^j = \frac{X_i^j - lb^j}{ub^j - lb^j}$, where X_i^j denotes the j-th dimension of X_i. After the min-max normalization over the searching space, we min-max normalize the objective values within this time step, $y_i = \frac{y_i - y_{min}}{y_{max} - y_{min}}$, where y_{min} and y_{max} denotes the lowest and highest achieved objective values in this time step. We have to note that by normalizing the X_i and y_i within the range of $[0, 1]$, we attain universal representation for different optimization problems, ensuring the generalizability of the subsequent neural network modules. The normalized $\{X_i\}_{i=1}^m$ and $\{y_i\}_{i=1}^m$ are then re-organized as a collection of meta data $\{\{(X_i^j, y_i)\}_{i=1}^m\}_{j=1}^d$ with the shape of $d \times m \times 2$. We then embed the meta data with a linear mapping $W_{emb} \in \mathbb{R}^{2\times h}$ as the final input encoding s, of which the shape is $d \times m \times h$. h denotes the hidden dimension of the subsequent two-stage attention module.

Two-stage Attention Block. We construct a two-stage attention block (Ts-Attn) to aggregate optimization status information across candidate solutions and across each dimension of the decision variables. The overall computation graph of the Ts-Attn is illustrated in the right of Figure 2 in the main body, of which a basic component is the attention block ($Attn$). As illustrated in the left of Figure 2 in the main body, the $Attn$ block mainly follows designs of the original Transformer Vaswani et al. (2017), except that the layer normalization Ba et al. (2016) is used instead of batch normalization Ioffe & Szegedy (2015). Given a group of L input encoding vectors $X_{in} \in \mathbb{R}^{L\times h}$, Eq. (1) details the computation of the $Attn$ block.

$$\begin{aligned} g &= \text{LN}(X_{in} + \text{MHSA}(X_{in})) \\ v &= \text{FF}^{(2)}(\text{ReLU}(\text{FF}^{(1)}(g))) \\ o &= \text{LN}(g + v) \end{aligned} \tag{1}$$

where MHSA, LN and FF denote the multi-head self-attention Vaswani et al. (2017) (with the hidden dimension of h), layer normalization Ba et al. (2016) and linear feed forward layer respectively. The output o holds identical shape with the input X_{in}. In our Ts-Attn block, we employ an $Attn$ block $Attn_{inter}$ for the first cross-solution information sharing stage, and the other $Attn$ block $Attn_{intra}$ for the second cross-dimension information sharing stage (illustrated in the right of Figure 2 in the main body. The Ts-Attn receives the input encoding s of the current candidate population, and then advances the information sharing in both cross-solution and cross-dimension level. The computation is detailed in Eq. (2).

$$\begin{aligned}
H &= \text{Attn}_{inter}(S) \\
H &= \text{Transpose}(H, d \times m \times h \rightarrow m \times d \times h) + \text{PE} \\
H_{out} &= \text{Attn}_{intra}(H) \\
F_{indiv} &= \text{MeanPooling}(H_{out}, m \times d \times h \rightarrow m \times h) \\
F_{pop} &= \text{MeanPooling}(F_{indiv}), m \times h \rightarrow h)
\end{aligned} \tag{2}$$

At the first stage, we let the input encoding s (attained from the pre-processing module) pass through $Attn_{inter}$. Since we group the encodings of the same dimension of all candidates in s, the $Attn_{inter}$ promotes the optimization information sharing across candidates in current population. From the first stage, we obtain a group of hidden features H with the shape of $d \times m \times h$. At the second stage, we first transpose H into the shape of $m \times d \times h$ to regroup all dimensions of a candidate together. We then add cos/sin positional encoding (PE) over the transposed H to inform the order of different dimensions in a candidate. We then let H pass through $Attn_{intra}$ to advance the information sharing among the different dimensions within the same candidate. The output of $Attn_{intra}$ holds the shape of $m \times d \times h$. At last, we apply MeanPooling on H_{out} to get the landscape feature for each candidate F_{indiv} in the population, and apply a second MeanPooling on F_{indiv} to get the landscape feature for the whole candidate population F_{pop}. We have to note that we calculate both F_{indiv} and F_{pop} to make our NeurELA compatible with diverse MetaBBO algorithms, which either require the landscape feature of the whole population (e.g., Wu & Wang (2022)) or require a separate landscape feature for each candidate (e.g., Sun et al. (2021)). The highly parallelizable attention-based neural-network architecture ensure the scalability of our method as the amount of the sampled candidates or the dimensions of the problem increases.

Now we summarize the end-to-end workflow of the neural landscape analyser (Λ_θ) in our NeurELA. At any time-step t within the optimization process, the pre-processing module transforms the information of the candidate population (i.e., $\{X_i^t\}_{i=1}^m$ and $\{y_i^t\}_{i=1}^m$) into the input encoding s. Then the Ts-Attn module transforms s into the dynamic landscape features F_{inidiv}^t and F_{pop}^t.

A.2 Train-test split of BBOB testsuites

BBOB contains 24 synthetic problems owning various landscape properties Mersmann et al. (2011) (e.g. multi-modality, global structure, separability and etc.). Table 1 list all of the 24 problems according to their category. Due to the diversity of properties, how to split these problems into train-test set becomes a key issue to ensure the training performance and its generalization ability. Our fundamental principle to split is to maximize the inclusion of representative landscape properties as possible. Specifically we visualize these 24 problems under 2D setting, and then select 12 representative problems into train set. We also provide contour map of problems in train set in Figure 1 and test set in Figure 2. Moreover, to avoid possible issue Kudela (2022) coming from fixed optima which is often located in $[0, ..., 0]$ in current benchmark problems (this might facilitate model to overfit to this fixed point), we thus add random offset O into each problems, that is to convert $y = f(x)$ into $y = f(x - O)$. This operation is inserted into both train set $\mathbb{D}_{\text{train}}$ and test set $\mathbb{D}_{\text{test}}$.

A.3 Train-test split of BBOB-Noisy and Protein-Docking testsuites

We summarize some key characteristic of this two testsuits as follows.

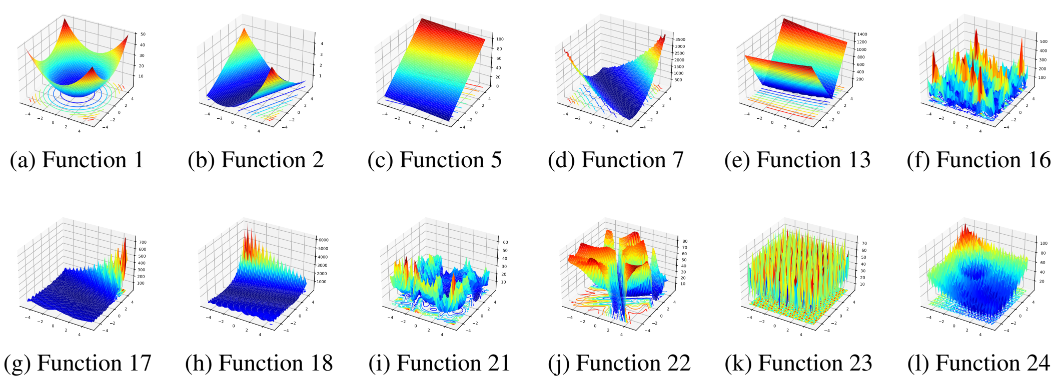
- **BBOB-Noisy**: this testsuits contains 30 noisy problems from COCO Hansen et al. (2021). They are obtained by further inserting noise with different models and levels into problems in BBOB testsuits. BBOB-Noisy is characterized by its noisy nature and often used to examine robustness of certain optimizers.
- **Protein-Docking**: this testsuits contains 280 instances of different protein-protein complexes Hwang et al. (2010). These problems are characterized by rugged objective landscapes and are computationally expensive to evaluate.

We follow train-test split for these two testsuites defined in MetaBox Ma et al. (2023). Under easy mode in MetaBox, 75% of instances are allocated into training and the remaining 25% are used in

Table 1: Overview of the BBOB testsuits.

	No.	Functions
Separable functions	1	Sphere Function
	2	Ellipsoidal Function
	3	Rastrigin Function
	4	Buche-Rastrigin Function
	5	Linear Slope
Functions with low or moderate conditioning	6	Attractive Sector Function
	7	Step Ellipsoidal Function
	8	Rosenbrock Function, original
	9	Rosenbrock Function, rotated
Functions with high conditioning and unimodal	10	Ellipsoidal Function
	11	Discus Function
	12	Bent Cigar Function
	13	Sharp Ridge Function
	14	Different Powers Function
Multi-modal functions with adequate global structure	15	Rastrigin Function (non-separable counterpart of F3)
	16	Weierstrass Function
	17	Schaffers F7 Function
	18	Schaffers F7 Function, moderately ill-conditioned
	19	Composite Griewank-Rosenbrock Function F8F2
Multi-modal functions with weak global structure	20	Schwefel Function
	21	Gallagher's Gaussian 101-me Peaks Function
	22	Gallagher's Gaussian 21-hi Peaks Function
	23	Katsuura Function
	24	Lunacek bi-Rastrigin Function
Default search range: $[-5, 5]^D$		

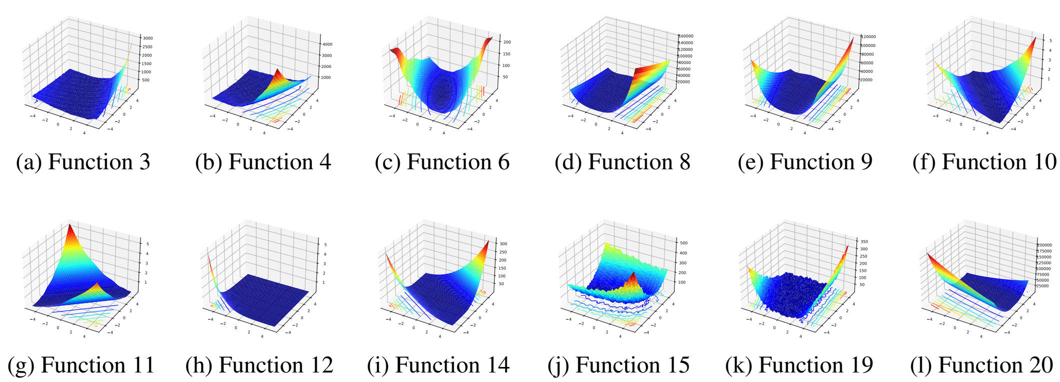
testing. Training or further fine-tuning on these two testsuites in our experiments are executed in the train set, and then validate performance of corresponding MetaBBO algorithms in the test set $\mathbb{D}_{\text{test}}$.



(a) Function 1 (b) Function 2 (c) Function 5 (d) Function 7 (e) Function 13 (f) Function 16

(g) Function 17 (h) Function 18 (i) Function 21 (j) Function 22 (k) Function 23 (l) Function 24

Figure 1: Fitness landscapes of functions in BBOB **train** set when dimension is set to 2.



(a) Function 3 (b) Function 4 (c) Function 6 (d) Function 8 (e) Function 9 (f) Function 10

(g) Function 11 (h) Function 12 (i) Function 14 (j) Function 15 (k) Function 19 (l) Function 20

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

A.4 LICENSE OF USED OPEN-SOURCED ASSETS

Our codebase can be accessed at `https://anonymous.4open.science/r/Neur-ELA-303C`. In Table 2 we listed several open-sourced assets used in our work and their corresponding licenses.

Table 2: Used open-sourced tools and their licenses.

Used scenario	**Asset**	**License**
Top-level optimizer	PyPop7 Duan et al. (2022)	GPL-3.0 license
MetaBBO algorithms implementation Low-level train-test workflow	MetaBox Ma et al. (2023)	BSD-3-Clause license
Parallel processing	Ray Moritz et al. (2018)	Apache-2.0 license
ELA feature calculation	pflacco Kerschke & Trautmann (2019)	MIT license

A.5 CONTROL-PARAMETERS OF ES

Fast CMAES We grid-search three key hyper-parameters in Fast CMAES, including the mean value μ and sigma value σ of the initial Gaussian distribution used for sampling, learning rate of evolution path update c. We list the grid search options in Table 3 and choose the best setting according to training performance on BBOB. Besides, for other control-parameters of the Fast CMAES, we follow the default settings listed in its original paper Li et al. (2018).

Table 3: Grid-search of control-parameters of Fast CMAES.

Control-parameters	**Grid options**	**Selected setting**
Initial mean value μ	$[0^D, \mathcal{R}^D]$	$\mathcal{R}^D$
Initial sigma value σ	[0.1, 0.3]	0.3
Learning rate of evolution path update c	$[2.0/(D+5.0), 6.0/(D+5.0)]$	$2.0/(D+5.0)$

Note: D represents the searching dimension of Fast CMAES. More specifically, as the top-level optimizer to neural-evolve our neural landscape analyser Λ_θ, D specifies the dimension of Λ_θ which is 3296 under default settings in our main experiment.

Other candidate evolution strategy variants We follows the default settings as implementations in PyPop7 Duan et al. (2022) for other candidate top-level optimizers. We made a comparision study among SEP-CMAES Ros & Hansen (2008), R1ES, RMES Li & Zhang (2017), original CMAES Hansen & Ostermeier (2001) and Fast CMAES under their default settings and finally select Fast CMAES as the default top-level optimizer of this work.

B ADDITIONAL DISCUSSION

B.1 TRAINING CONVERGENCE

In NeurELA, the meta-objective as defined in Eq. (4), is non-differentiable. Hence, we train the neural network in NeurELA through neuroevolution. Such paradigm requires effective evolutionary optimizers which maintain a population of neural networks and reproduce elite offsprings iteratively according to the training objective of the neural networks. In NeurELA, we adopt Evolution Strategy (ES) since it is claimed to be more effective then other optimizers. There are many modern variants of ES method, of which we select five: Fast CMAES (Li et al., 2018), Sep-CMAES (Ros & Hansen, 2008), R1ES (Li & Zhang, 2017), RMES (Li & Zhang, 2017) and CMAES (Hansen & Ostermeier, 2001) as candidates. We present the training curves of all five optional ES baselines under our training settings in Figure 3. The results demonstrate that the Fast CMAES we adopted for training NeurELA converges and achieves superior training effectiveness to other ES baselines.

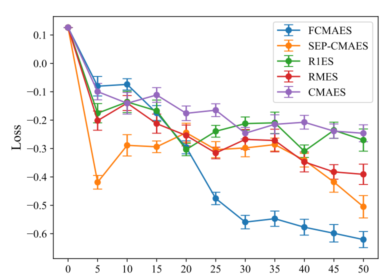


Figure 3: Training curves of different ES baselines when training NeurELA

B.2 Difference between NeurELA and Deep-ELA

Although a previous work Deep-ELA (Seiler et al., 2024) also proposed using attention-based architecture for landscape analysis, there are significant differences between our NeurELA and Deep-ELA, which we listed as below:

1. Target scenario. NeurELA is explicitly designed for MetaBBO tasks, where dynamic optimization status is critical for providing timely and accurate decision-making at the meta level. In contrast, Deep-ELA serves as a static profiling tool for global optimization problem properties and is not tailored for dynamic scenarios. NeurELA supports dynamic algorithm configuration, algorithm selection, and operator selection. In contrast, Deep-ELA's features are restricted to static algorithm selection and configuration, limiting its adaptability in dynamic MetaBBO workflows.

2. Feature extraction workflow. First, NeurELA addresses the limited scalability of Deep-ELA for high dimensional problem. Concretely, the embedding in Deep-ELA is dependent on the problem dimension and hence the authors of Deep-ELA pre-defined a maximum dimension (50 in the original paper). To address this, NeurELA proposes a novel embedding strategy which re-organizes the sample points and their objective values to make the last dimension of the input tensor is 2 (Section 3.2). This embedding format has a significant advantage: the neural network of NeurELA is hence capable of processing any dimensional problem and any number of sample points. NeurELA enhances the information extraction through its two-stage attention-based neural network. Specifically, when processing the embedded data, Deep-ELA leverages its self-attention layers for information sharing across sample points only. In contrast, NeurELA incorporates a two-stage attention mechanism, enabling the neural network to first extract comprehensive and useful features across sample points (cross-solution attention) and then across problem dimensions (cross-dimension attention). This design helps mitigate computational bias and improve feature representation.

3. Training method. The training objective and training methodology in NeurELA and Deep-ELA are fundamentally different. Deep-ELA aims to learn a neural network that could serve as an alternative of traditional ELA. Its training objective is to minimize the contrastive loss (InfoNCE) between the outputs of its two prediction heads (termed as student head and teacher head) by gradient descent, in order to achieve invariance across different landscape augmentation on the same problem instance. In contrast, the training objective of NeurELA is to learn a neural network that could provide dynamic landscape features for various MetaBBO tasks. Specifically, its objective is to maximize the expected relative performance improvement when integrated into different MetaBBO methods. Since such relative performance improvement is not differentiable, NeurELA employs neuroevolution as its training methodology. Neuroevolution is recognized as an effective alternative to gradient descent, offering robust global optimization capabilities.

In summary, NeurELA and Deep-ELA are two totally different works with different target operating scenarios, algorithm design tasks, neural network designs and workflows, and training methodologies.

B.3 Further Interpretation Analysis

To further interpret what features have been learned by our NeurELA, we have conducted following experimental analysis to further interpret the relationship between NeurELA features and traditional ELA features, where we uses Pearson Correlation analysis to quantify the correlation between each NeurELA feature and each traditional ELA feature.Below, we explain our experimental methodology step by step:

1. We select three MetaBBO methods (LDE, RLEPSO and RL-DAS) from our training task set and employ their pre-trained models to optimize the 24 problem instances in CoCo BBOB-10D suite. Each MetaBBO method performed 10 independent runs per problem instance, with each run consisting of 500 optimization steps. Now we obtain 3*24*10 = 720 optimization trajectories, each with length 500, and the data at each step of a trajectory is the population and the corresponding objective values $\{Xs, Ys\}$.

2. Based on the obtained trajectories, we use the pre-trained NeurELA model (outputs 16 features) and the traditional ELA (we choose 32 ELA features from the traditional ELA including the Meta-model group, Convexity group, Level-Set group, Local landscape group and Distribution group) to calculate landscape features for each optimization step. After the computation, we obtain 720 landscape features time series for NeurELA and traditional ELA respectively.

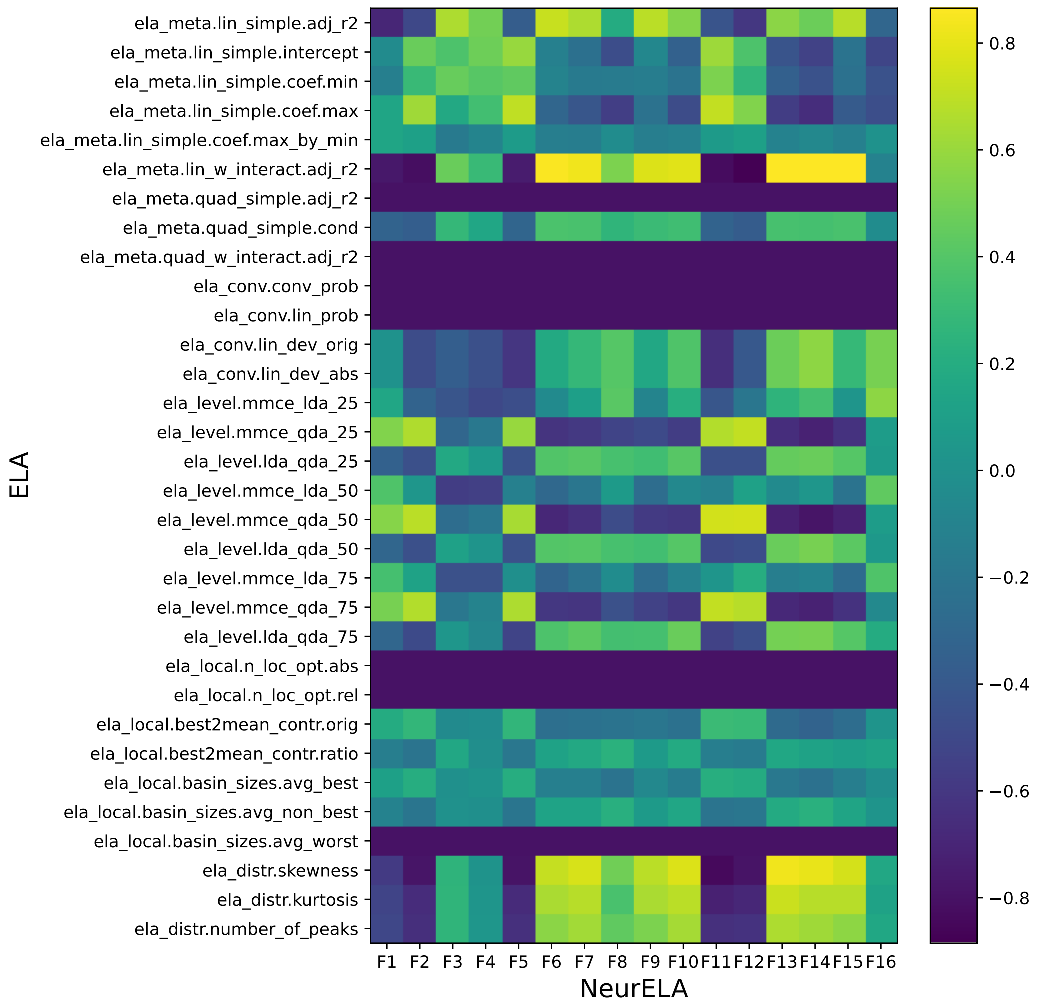


Figure 4: Correlation score of NeurELA features and traditional ELA features

3. For each pair of landscape features time series, we measure the relationship between the i-th feature in NeurELA features and the j-th feature in traditional ELA features by computing the Pearson Correlation Coefficient $r_{i,j}$ of the time series of these two features: $\{F_{i,1}^{NeurELA} \dots F_{i,500}^{NeurELA}\}$ and $\{F_{j,1}^{ELA} \dots F_{j,500}^{ELA}\}$.

4. We obtain the final correlation scores of each feature pair between NeurELA and traditional ELA by averaging $r_{i,j}$ of this feature pair over the 720 time series data, $i \in \{1, 2, \dots, 16\}$, $j \in \{1, 2, \dots 32\}$. Finally we obtain a correlation matrix with 32 rows and 16 columns. We illustrate this correlation matrix by the heatmap in Figure 4. The x-axis denote 16 NeurELA features and y-axis denote 32 ELA features, a larger value denotes the two features are closely related.

From the correlation results in that Figure, we could find some relationship patterns between our NeurELA features and the traditional ELA features: a) four NeurELA features F1, F4, F8 and F16 are novel features learned by NeurELA which show weak correlation (< 0.6) with all ELA features. b) some NeurELA features show strong correlation with one particular feature group in traditional ELA, such as F3 with the Meta-model group. c) some NeurELA features show strong correlation with multiple feature groups in traditional ELA, such as F10 with Distribution group and Meta-model group. d) all NeurELA features show weak correlation with the Convexity group and Local landscape group, which might reveals these two group features are less useful for addressing MetaBBO tasks.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Qiqi Duan, Guochen Zhou, Chang Shao, Zhuowei Wang, Mingyang Feng, Yijun Yang, Qi Zhao, and Yuhui Shi. Pypop7: A pure-python library for population-based black-box optimization. *arXiv preprint arXiv:2212.05652*, 2022.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 2001.

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

Howook Hwang, Thom Vreven, Joël Janin, and Zhiping Weng. Protein–protein docking benchmark version 4.0. *Proteins: Structure, Function, and Bioinformatics*, 2010.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Pascal Kerschke and Heike Trautmann. Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco. *Applications in Statistical Computing*, 2019.

Jakub Kudela. A critical problem in benchmarking and analysis of evolutionary computation methods. *Nature Machine Intelligence*, 2022.

Zhenhua Li and Qingfu Zhang. A simple yet efficient evolution strategy for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 2017.

Zhenhua Li, Qingfu Zhang, Xi Lin, and Hui-Ling Zhen. Fast covariance matrix adaptation for large-scale black-box optimization. *IEEE Transactions on Cybernetics*, 2018.

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 *Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Annual Conference on Genetic and Evolutionary Computation*, 2011.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging ai applications. In *Symposium on Operating Systems Design and Implementation*, 2018.

Raymond Ros and Nikolaus Hansen. A simple modification in cma-es achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature*, 2008.

Moritz Vinzent Seiler, Pascal Kerschke, and Heike Trautmann. Deep-ela: Deep exploratory landscape analysis with self-supervised pretrained transformers for single-and multi-objective continuous optimization problems. *arXiv preprint arXiv:2401.01192*, 2024.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

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