A Technical Details of LSRE

A.1 Latent Instance Space Instruction

To construct the latent instance space, we first need to pre-define the target problem distribution $\mathcal{D}$. Specifically, we must obtain a sufficiently large number of problem instances to build $\mathcal{D}$. In this paper, we select the 24 basic functions from CoCo-BBO, and follow the method described in Section 3.1 to ultimately obtain 32,400 instances as $\mathcal{D}$. Actually, the construction of $\mathcal{D}$ can be flexibly extended to other user-defined problem ranges: 1) For synthetic problems, one can apply the method from Section 3.1 across multiple dimensions to generate numerous instances, or refer to the composition and transformation approaches used in CEC competitions (Mohamed et al. 2021) to create instances; 2) For realistic problems, considering that the number of existing realistic problem instances is limited, one can follow the BBOB-Noisy (Hansen et al. 2009) approach by adding noise to synthetic problems to simulate realistic scenarios and generate sufficient instances. Using these problem instance generation methods, users can customize the construction of $\mathcal{D}$ according to their specific application needs.
After obtaining the target problem distribution $\mathcal{D}$, we first compute the ELA feature vector E_f described in Section 3.1 and Appendix A.2 for each function instance $f \in \mathcal{D}$. Subsequently, we construct an autoencoder consisting of encoder W_θ and decoder W_ϕ according to the settings in Section 3.1 and Appendix A.3. The W_θ maps 21-dimensional ELA features to a 2D latent space $\mathcal{H}$, while the W_ϕ learns to reconstruct the original features from $\mathcal{H}$. The training process employs the standard reconstruction loss function defined in Section 3.1. After finishing the training, the fully trained W_θ can transform any problem instance into a 2D latent representation $h \in \mathbb{R}^2$.

A.2 ELA Features Selection

As mentioned in Section 3.1, Exploratory Landscape Analysis (ELA) is widely used for instance space analysis in BBO problems. In recent years, research related to ELA has rapidly developed, resulting in a large number of landscape feature sets with varying advantages, disadvantages, or high similarity. In this paper, based on a sensitivity and discriminability study of ELA features (Renau et al. 2020; Muñoz, Kirley, and Smith-Miles 2022), we carefully selected a representative, expressive, and computationally efficient subset of features to construct our 21-dimensional ELA feature vector for LSRE. These features comprehensively and independently describe key landscape characteristics of optimization problems, such as global structure, local structure, modality, separability, etc. We provide the detailed descriptions of selected ELA features in Table 1.

Feature group and name	Description
Meta-model(9 features) :	
ela_meta.lin_simple.adj_r2	Adjusted coefficient of determination of the linear regression model without interactions
ela_meta.lin_simple.intercept	The intercept of the linear regression model without interactions
ela_meta.lin_simple.coef.min	The smallest absolute coefficients of the regression linear model without interactions
ela_meta.lin_simple.coef.max	The biggest absolute coefficients of the regression linear model without interactions
ela_meta.lin_simple.coef.max_by_min	The ratio of the biggest and the smallest absolute coefficients of the regression linear model without interactions
ela_meta.lin_w_interact.adj_r2	Adjusted coefficient of determination of the linear regression model with interactions
ela_meta.quad_simple.adj_r2	Adjusted coefficient of determination of the quadratic regression model without interactions
ela_meta.quad_w_interact.adj_r2	Adjusted coefficient of determination of the quadratic regression model with interactions
ela_meta.quad_simple.cond	The condition of a simple quadratic model without interactions, i.e. the ratio of its (absolute) biggest and smallest coefficients
Convexity(4 features):	
ela_conv.conv_prob	The estimated probability of convexity
ela_conv.lin_prob	The estimated probability of linearity
ela_conv.lin_dev_orig	The average original deviation between the linear combination of the objectives and the objective of the linear combination of the observations
ela_conv.lin_dev_abs	The average absolute deviation between the linear combination of the objectives and the objective of the linear combination of the observations
Information Content(5 features):	
ic.h_max	The maximum information content from the Information Content of Fitness Sequences (ICoFiS) approach (Muñoz, Kirley, and Halgamuge 2014)
ic.eps_s	The settling sensitivity from ICoFiS
ic.eps_max	Similar to ic.eps_s, but with guaranteed non-missing values. This parameter is formally defined as the ϵ-value satisfying the condition H(ic.eps_max) == ic.h_max
ic.eps_ratio	The half partial information sensitivity from ICoFiS
ic.m0	The initial partial information from ICoFiS
Distribution(3 features):	
ela_distr.skewness	The skewness of the distribution of the function values
ela_distr.kurtosis	The kurtosis of the distribution of the function values
ela_distr.number_of_peaks	The estimation of the number of peaks in the distribution of the function values

Table 1: The list of ELA features

Operands : $[X, X_a, X_b, C]$ Operators :[sum,mean,add,sub,mul,div,neg,pow, sin,cos,abs,Tanh,exp,log,sqrt]	
X	Full vector slice $x[0:n]$, representing all elements of the input vector (dimension n).
Xa	Forward slice $x[0:n-1]$, excluding the last element, used to model predecessor relationships in sequences.
Xb	Backward slice $x[1:n]$, excluding the first element, used to model successor relationships in sequences.
C	Constant Vector, and each dimension's value is generated via scientific notation $mantissa * 10^{exponent}$, where the mantissa (base number) is selected from the integer range $[1, 11]$ (with 10 representing π and 11 representing the natural constant e), and the exponent is selected from the integer range $[-1, 3]$.
sum	Aggregation mode : Returns the summation of all vector elements (scalar) Inter-vector mode : Returns element-wise addition (vector).
$mean$	Aggregation mode : Returns the mean of all vector elements (scalar). Inter-vector mode : Returns element-wise average (vector).
add	Computes element-wise addition between two child nodes' outputs : $X + Y$
sub	Computes element-wise subtraction of child nodes' values: $X - Y$
mul	Computes element-wise multiplication of child nodes' computations: $X \odot Y$
div	Computes element-wise division of child nodes' values: If $\|Y\| < \epsilon$, outputs 1 to avoid division by zero; otherwise returns X/Y
neg	Negates the child node's value or all vector elements: $-X$
pow	Computes power function using left child as base and right child as exponent(element-wise for vectors). If $\|X\| < \epsilon \wedge Y = -1$, outputs 1 for that dimension; otherwise returns X^Y
sin	Applies sine function to the child node's value: $\sin(X)$
cos	Applies cosine function to the child node's value: $\cos(X)$
abs	Computes the element-wise absolute value: $\|X\|$
$Tanh$	Hyperbolic tangent activation: $Tanh(X)$
exp	Natural exponential function: e^X
log	If $\|X\| < \epsilon$, outputs 0; otherwise $\log_{10}(\|X\|)$ (implicit absolute value ensures $X \in \mathbb{R}^+$)
$sqrt$	Computes the square root of the absolute value $\sqrt{\|X\|}$ to prevent invalid inputs (negative values).

Table 2: Symbol Set

A.3 Setting Details of Autoencoder

Network Architecture. The proposed AE architecture employs a fully-connected neural network with symmetrical encoder W_θ and decoder W_ϕ components. W_θ consists of six consecutive linear layers that progressively compress the 21-dimensional feature vector E_f mentioned in Appendix A.2 through 128, 64, 32, 16, and 8-dimensional hidden representations, ultimately projecting the data into a 2-dimensional latent space $\mathcal{H}$. Each layer is followed by a PReLU activation function, with the final layer of W_θ utilizing Tanh activation to constrain outputs to the $[-1, 1]$ range. The decoder mirrors this network structure(excluding Tanh activation function) symmetrically, reconstructing the original E_f from the $\mathcal{H}$ through the same intermediate dimensionalities and PReLU activations.

Training Protocol. In the initialization phase of our experiments, we implement the following preprocessing and parameter configurations for AE training: The ELA feature vectors computed from the large-scale problem instances generated in Section 3.1 are partitioned into training-validation sets with an 80:20 ratio, followed by Min-Max normalization exclusively fitted on the train set. The Min-Max normalization process for ELA feature vectors $E_f \in \mathbb{R}^d$ (where $d = 21$) is performed dimension-wise. For the k-th dimension feature $E_f^{(k)}$ of instance f in train set $\mathcal{D}_{train}$, the normalized value $\hat{E}_f^{(k)}$ is computed as:

$$\hat{E}_f^{(k)} = \frac{E_f^{(k)} - \min_{\mathcal{D}_{\text{train}}}(E^{(k)})}{\max_{\mathcal{D}_{\text{train}}}(E^{(k)}) - \min_{\mathcal{D}_{\text{train}}}(E^{(k)})} \tag{1}$$

where $\min_{\mathcal{D}_{\text{train}}}(E^{(k)})$ and $\max_{\mathcal{D}_{\text{train}}}(E^{(k)})$ represent the minimum and maximum values of the k-th feature observed across the train set $\mathcal{D}_{\text{train}}$. The training lasts for 300 epochs for a batch of batch size = 32 $\hat{E}_f^{(k)}$. For AE optimization, we employ the Adam optimizer with an initial learning rate of $1e-3$, coupled with an exponential learning rate scheduler (ExponentialLR) where the decay factor γ is set to 0.9862327.

A.4 Symbol Set Design

In Section 3.2, we propose a comprehensive symbol set comprising flexible operators and operands, constructing a neighboring symbolic space specifically for single-objective continuous optimization problems. The precise definition of this symbol set and its operational processing will be elaborated in Table 2.

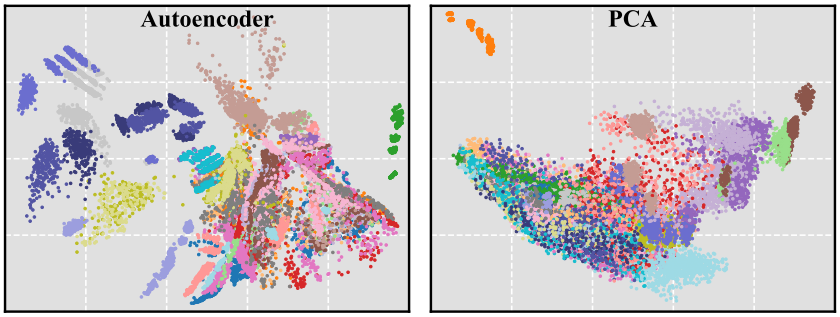

Figure 1: Visualization Result Comparison

A.5 Hyper-Parameters Settings

In LSRE, the hyper-parameter configuration of the GP process critically determines both the search accuracy and efficiency when searching target instance points. Through extensive experimental validation, we have established an optimized parameter set that delivers superior performance: Each GP process employs a population size $pop\_size = 1000$ to maintain population diversity, with both $generations = 50$ and tournament size $tournament\_size = 50$ to ensure evolutionary search efficiency. The stopping criterion $stopping\_criteria =$ 1e-2 for early termination to conserve computational resources, while parallel evaluation utilizes 10 cores($n\_jobs = 10$). During generational search operations, we implement the following genetic operation probabilities: $p\_crossover = 0.6$, $p\_subtree\_mutation = 0.25$, $p\_point\_mutation = 0.1$, $p\_hoist\_mutation = 0.04$ and 0.01 for no operation. For tree initialization, the depth range $init\_depth = [5, 8]$, with offspring trees after genetic operations constrained to $mutate\_depth = [5, 15]$ and a maximum allowable tree depth of 15. This carefully tuned parameter set has been empirically demonstrated to achieve optimal balance between exploration capability and computational efficiency in our LSRE framework.

A.6 Design Philosophy behind LSRE

We will focus on the novel components of LSRE to elaborate their design philosophy and implementation details.

Symbol Set Design. We formally define and elaborate the symbols in the Proximal Symbolic Space in Section 3.2 and Table 2. Notably, unlike prior work that primarily employs basic symbols `X` and `C`, we introduce the `X[i:j]` notation to enable finer-grained generation. Specifically, we implement two commonly used slice vector symbols `Xa` and `Xb`, whose formal definitions are provided in Table 2. This slice-based symbolic design offers three key advantages: 1) Compared to single-variable symbol designs (e.g., `X1`, `X2`), the slice symbols (`Xa`, `Xb`) significantly reduce tree structure complexity; 2) Slice operations facilitate interactive computations across different dimensions, enhancing the flexibility of tree-based expressions and computations; 3) Such slice symbols frequently appear in existing benchmark functions such as F_8 : Rosenbrock Function(original) in CoCo-BBOB, demonstrating their practical utility in common optimization scenarios.

Latent Space Construction In Section 3.1, we adopt a trainable autoencoder to construct the latent instance space. We note that while PCA (Jolliffe 2011) is widely used for dimension reduction, we prefer autoencoders because: 1) PCA assumes linear feature correlations (Almotiri, Elleithy, and Elleithy 2017; Wang, Yao, and Zhao 2016), whereas ELA features are computed independently; 2) Our tests show PCA's first two components explain <51% variance for many instances in $\mathcal{D}$. To better illustrate the differences between these two techniques, we provide a visualized comparison of autoencoder and PCA in Fig 1. The results demonstrate that: 1) The autoencoder-constructed latent space clearly separates most instances - points from the same base function in CoCo-BBOB (same color) tend to cluster together, with sub-clusters further organized by dimension; 2) In contrast, PCA-based visualization only shows distinguishable separation for a few functions (top-left region). While some instances from the same function may group together, they lack dimensional discrimination, and most functions' instances remain heavily mixed, making PCA ineffective for distinguishing between different function instances.

Cross-Dimensional Strategy. In Section 3.2, we propose a novel cross-dimensional local search strategy to enhance search accuracy. The motivation behind this local search strategy comes from the observation that while we can ultimately find the optimal function formulation, different dimensions of decision variables for the same mathematical expression may lead to variations in the ELA features of the resulting function instances. Therefore, our proposed cross-dimensional local search strategy further identifies the best-performing problem dimension to improve overall search precision. As clearly demonstrated

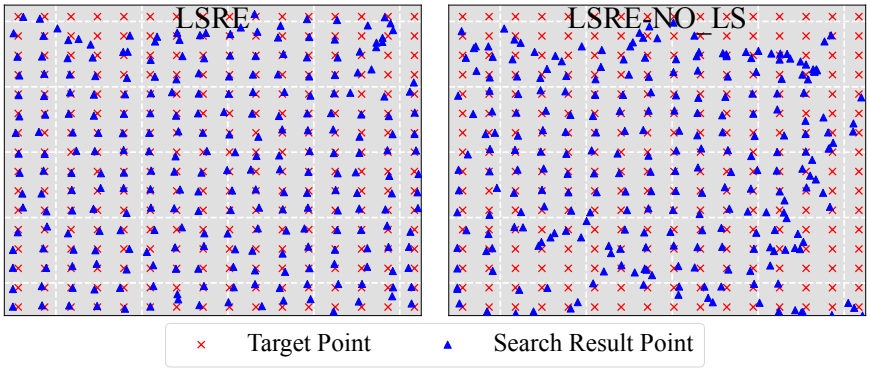

Figure 2: Comparison of Searching Result

in Section 4.3 (Ablation Study), the cross-dimensional local search plays a more crucial role in LSRE. To provide a more intuitive illustration of its impact on search accuracy, we compare two versions of LSRE: one employing the cross-dimensional local search strategy and another using a fixed dimension. We conduct searches for 256 target function points and visualize the results in Fig 2. The comparison clearly shows that LSRE with cross-dimensional local search almost precisely locates the desired function instances, whereas the fixed-dimension version struggles to achieve accurate results in certain regions.

B Experimental Details

B.1 ASRD Test Settings

In the average success rate distribution (ASRD) test in Section 4.1, we construct an algorithm pool comprising six representative optimizers spanning major evolutionary computation paradigms: DE (Storn and Price 1997), NL-SHADE-LBC (Stanovov, Akhmedova, and Semenkin 2022), CMAES (Hansen, Müller, and Koumoutsakos 2003), sep-CMAES (Ros and Hansen 2008), PSO (Kennedy and Eberhart 1995), and GL-PSO (Gong et al. 2015). The parameter configurations and optimization procedures for these optimizers strictly follow their original publications and are implemented based on both PyPop7 [1] and MetaBox-v2 (Ma et al. 2025) codebases. For each test problem instance in the baseline benchmark, every optimizer is independently executed 51 runs, with a computational budget of $maxFES = 10^5 * dim$ allocated per run, where dim denotes the problem instance's dimension. The random seed for each optimizer is set as $100\times(run\_index + 2)$, where $run\_index \in 0, 1, ..., 50$ denotes the sequential number of the current run. When generating test problem instances for the baseline benchmark, we employ distinct random seeds to produce varied test instances. Specifically: 1) For CoCo-BBOB instances, the random seed configuration correlates with the total number of generated instances; 2) For MA-BBOB instances, a fixed seed value of 100 is used to collectively sample all required test instance parameters in one time; 3) For both Diverse-BBOB and GP-BBOB instances, the random seed assignment associates with the search function's identification number during each test instance generation.

B.2 Experiment Protocol of Generalization Test

B.2.1 Benchmarks of Realistic Test

To evaluate MetaBBO's zero-shot performance on unseen real-world problem sets, we select the following three well-known benchmarks in this paper:

- **HPO-B benchmark** (Arango et al. 2021) includes a wide range of hyper-parameter optimization tasks for 16 different model types (e.g., SVM, XGBoost, etc.). These models have various search spaces ranging from $[0, 1]^2$ to $[0, 1]^{16}$. Each model is evaluated on several datasets, resulting in a total of 86 tasks. In this paper, we adopt the continuous version of HPOB, which provides surrogate evaluation functions for time-consuming machine learning tasks to save evaluation time.

[1] https://github.com/Evolutionary-Intelligence/pypop

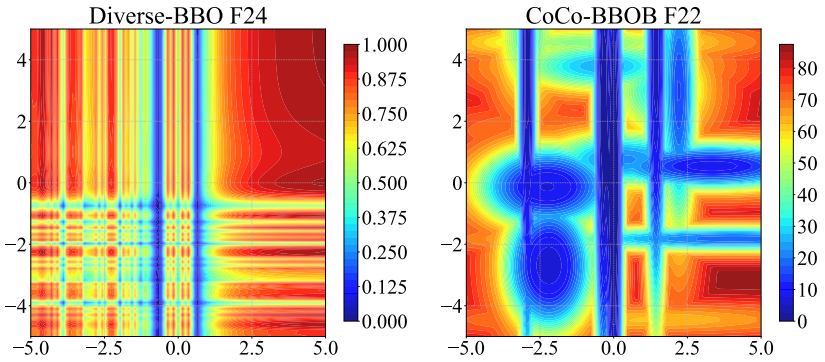

Figure 3: Two Functions' Contour. **Left**: Diverse-BBO F24's contour. **Right**: CoCo-BBOB F22's contour.

- **Protein-docking benchmark** (Hwang et al. 2010), where the objective is to minimize the Gibbs free energy resulting from protein-protein interaction between a given complex and any other conformation. We select 28 protein complexes and randomly initialize 10 starting points for each complex, resulting in 280 problem instances. To simplify the problem structure, we only optimize 12 interaction points in a complex instance (12D problem).

- **UAV benchmark** (Shehadeh and Kudela 2025) provides 56 terrain-based landscapes as realistic Unmanned Aerial Vehicle (UAV) path planning problems, each of which is 30-dimensional. The objective is to select a given number of path nodes (x, y, z coordinates) from the 3D space, so the UAV can fly as shortly as possible in a collision-free manner.

B.2.2 Training Protocol of MetaBBO

In Section 4.2 of this paper, we select four representative MetaBBO approaches that cover different methodology categories such as those for operator selection (DEDDQN (Sharma et al. 2019)), for algorithm configuration (LDE (Sun et al. 2021) & GLEET (Ma et al. 2024)) and for algorithm generation (SYMBOL (Chen et al. 2024)). We implement the complete training and testing pipeline on the MetaBox-v2 (Ma et al. 2025) platform, where we configure all MetaBBO methods with 100 training epochs and a batch size of 10, while strictly maintaining the parameter settings as specified in their original papers.

B.3 Insightful and Interesting Finds

Visualization Result Comparison between AE and PCA. We have present the visualized comparison between PCA and autoencoder in instance space analysis in Section A.6 and will not be reiterated here.

Analysis of Function Formula For the function formula generation process of LSRE, genetic operations are performed on the symbolic trees to produce new offspring, evolving the optimal individuals that having specific ELA features. Therefore, analyzing the formulas or symbolic combinations of these evolved optimal individuals and their reflected landscape can help us gain deeper insights into the methodology for constructing diverse benchmark functions with diverse landscape features.

We analyze the mathematical expressions and contour plots of representative instances from Diverse-BBO and CoCo-BBOB. Specifically, we select F24 from Diverse-BBO and F22 (Gallagher's Gaussian 21-hi Peaks Function) from CoCo-BBOB, which occupy overlapping positions in the latent space, indicating that they share similar ELA features. Their contour plots are illustrated in Figure 3. A clear observation reveals that both functions exhibit multiple narrow, long and irregular valley structures. Additionally, their landscapes demonstrate steep variations near the global optimum, accompanied by high condition numbers. The mathematical formulations of Diverse-BBO's F24 and CoCo-BBOB's F22 are presented in Eq. 2 and Eq. 3:

$$F_{24}(x) = |\sum_{i=0}^{n-2}(\cos(-(\sqrt{(X_i)^{-2}}))) \times \frac{1}{n}\sum_{i=0}^{n-1}(\cos(\frac{-(X_i)}{e^{X_i}}))| \tag{2}$$

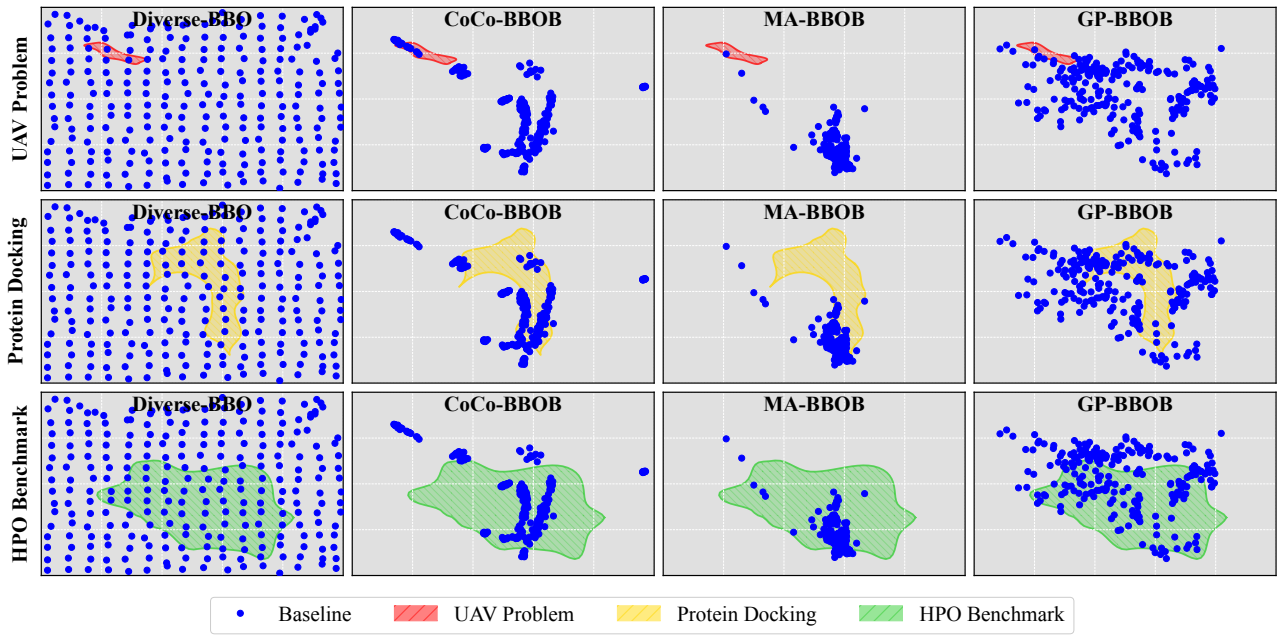

Figure 4: Baselines' Coverage of Realistic Problem Sets.

where n = 2,

$$F_{22}(x) = (10 - \max_{i=1}^{21} w_i \exp(-\frac{1}{2D}(x - y_i)^T R^T C_i R(x - y_i))) \times T_{osz} + f_{pen}(x) + f_{opt} \tag{3}$$

where y_i are the local optima uniformly drawn from the domain $[-4.9, 4.9]^D$ for $i = 2, \ldots, 21$, $y_1 \in [-3.92, 3.92]^D$ is the global optima, and all other symbols follow the definitions provided in CoCo-BBOB.

We can have an intuitive observation when comparing two function formulations: Diverse-BBO's F24 achieves significantly reduced formula complexity relative to the carefully human engineered F22 from CoCo-BBOB, showing that simple symbolic expressions can generate complicated and diverse landscape features. It is worth noting that using LSRE to generate functions eliminates dependency on expert knowledge through evolutionary symbolic composition. Concretely, CoCo-BBOB's F22 constructs its landscape via random sampling the local optima positions, adjusting global optimum weight and using non-linear transformations (max/exp operations); Diverse-BBO's F24 achieves comparable landscape through three components: $\sum_{i=0}^{n-2}(\cos(-(\sqrt{(X_i)^{-2}})))$ for narrow and long global valley structures; $\frac{1}{n}\sum_{i=0}^{n-1}(\cos(\frac{-(X_i)}{e^{X_i}}))$ for narrow and long local optima valleys and multi-modal landscape; Multiplication and outer abs operation combine the two primary landscape features.

Generalization Interpretation of Baselines. In Section 4.2, we evaluated the generalization performance of MetaBBO approaches (trained on four baselines) across three realistic problem sets and conducted corresponding performance analyses. Additionally, we established preliminary connections between benchmark diversity and MetaBBO generalization potential through intuitive function instance visualizations. Here, we further investigate this relationship between benchmark diversity and MetaBBO generalization potential. To this end, we visualize all instances of each realistic problem set in the latent space separately, as shown in Fig 4. Here, the red shaded area represents instance coverage of the UAV problem set, yellow indicates the Protein-Docking problem set, and green denotes the HPO-B problem set.

By analyzing the coverage degree of different benchmarks across these realistic problem sets and correlating them with their corresponding average performance gains, we derive the following insightful conclusions: 1) The Diverse-BBO function instances demonstrate uniform coverage across all three realistic problem sets, which explains why MetaBBO approaches trained on Diverse-BBO achieve the best average performance gains across all benchmarks; 2) For the UAV problem set, CoCo-BBOB function instances show better (though still partial) coverage of the red-shaded UAV distribution compared to the other two baselines, resulting in its MetaBBO approach achieving second-best performance gains; 3) Regarding the Protein-Docking problem, none of the three baseline function instances can effectively cover the yellow-shaded distribution, leading to relatively poor average performance gains across all approaches; 4) For the HPO-B problem, both GP-BBOB and CoCo-BBOB function instances

exhibit partial coverage of the green-shaded distribution, outperforming MA-BBOB in this regard. Notably, GP-BBOB demonstrates more uniform coverage of the green-shaded area compared to CoCo-BBOB, which explains why MetaBBO approaches trained on GP-BBOB achieve the second-best performance gains, followed by those trained on CoCo-BBOB in third place. Therefore, our LSRE framework enables controllable, uniform, and fine-grained instance generation within the instance space. The resulting diverse instances (Diverse-BBO) prove beneficial for MetaBBO training, enhancing its generalization potential.

References

Almotiri, J.; Elleithy, K.; and Elleithy, A. 2017. Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 1–5. IEEE.

Arango, S. P.; Jomaa, H. S.; Wistuba, M.; and Grabocka, J. 2021. Hpo-b: A large-scale reproducible benchmark for black-box hpo based on openml. *arXiv preprint arXiv:2106.06257*.

Chen, J.; Ma, Z.; Guo, H.; Ma, Y.; Zhang, J.; and Gong, Y.-j. 2024. Symbol: Generating Flexible Black-Box Optimizers through Symbolic Equation Learning. In *ICLR*.

Gong, Y.-J.; Li, J.-J.; Zhou, Y.; Li, Y.; Chung, H. S.-H.; Shi, Y.-H.; and Zhang, J. 2015. Genetic learning particle swarm optimization. *IEEE transactions on cybernetics*, 46(10): 2277–2290.

Hansen, N.; Finck, S.; Ros, R.; and Auger, A. 2009. *Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions*. Ph.D. thesis, INRIA.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1): 1–18.

Hwang, H.; Vreven, T.; Janin, J.; and Weng, Z. 2010. Protein–protein docking benchmark version 4.0. *Proteins: Structure, Function, and Bioinformatics*, 78(15): 3111–3114.

Jolliffe, I. 2011. Principal component analysis. In *International encyclopedia of statistical science*, 1094–1096. Springer.

Kennedy, J.; and Eberhart, R. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, 1942–1948.

Ma, Z.; Chen, J.; Guo, H.; Ma, Y.; and Gong, Y.-J. 2024. Auto-configuring Exploration-Exploitation Tradeoff in Evolutionary Computation via Deep Reinforcement Learning. In *GECCO*, 1497–1505.

Ma, Z.; Gong, Y.-J.; Guo, H.; Qiu, W.; Ma, S.; Lian, H.; Zhan, J.; Chen, K.; Wang, C.; Huang, Z.; et al. 2025. MetaBox-v2: A Unified Benchmark Platform for Meta-Black-Box Optimization. *arXiv preprint arXiv:2505.17745*.

Mohamed, A. W.; Hadi, A. A.; Mohamed, A. K.; Agrawal, P.; Kumar, A.; and Suganthan, P. N. 2021. Problem definitions and evaluation criteria for the CEC 2021 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization. Technical report.

Muñoz, M. A.; Kirley, M.; and Halgamuge, S. K. 2014. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE transactions on evolutionary computation*, 19(1): 74–87.

Muñoz, M. A.; Kirley, M.; and Smith-Miles, K. 2022. Analyzing randomness effects on the reliability of exploratory landscape analysis. *Natural Computing*, 21(2): 131–154.

Renau, Q.; Doerr, C.; Dreo, J.; and Doerr, B. 2020. Exploratory landscape analysis is strongly sensitive to the sampling strategy. In *International Conference on Parallel Problem Solving from Nature*, 139–153. Springer.

Ros, R.; and Hansen, N. 2008. A simple modification in CMA-ES achieving linear time and space complexity. In *International conference on parallel problem solving from nature*, 296–305. Springer.

Sharma, M.; Komninos, A.; López-Ibáñez, M.; and Kazakov, D. 2019. Deep reinforcement learning based parameter control in differential evolution. In *GECCO*, 709–717.

Shehadeh, M. A.; and Kudela, J. 2025. Benchmarking global optimization techniques for unmanned aerial vehicle path planning. arXiv:2501.14503.

Stanovov, V.; Akhmedova, S.; and Semenkin, E. 2022. NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 Numerical Optimization. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, 01–08. IEEE.

Storn, R.; and Price, K. 1997. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4): 341.

Sun, J.; Liu, X.; Bäck, T.; and Xu, Z. 2021. Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. *TEC*, 25(4): 666–680.

Wang, Y.; Yao, H.; and Zhao, S. 2016. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184: 232–242.