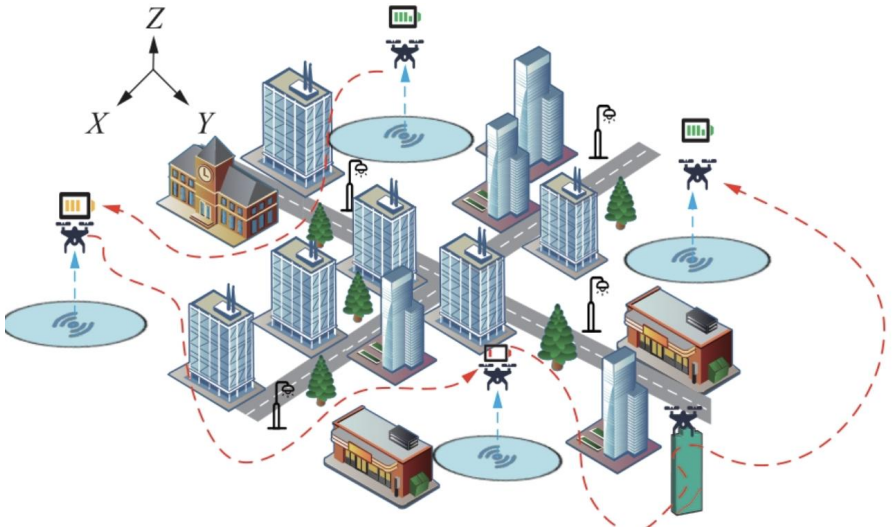
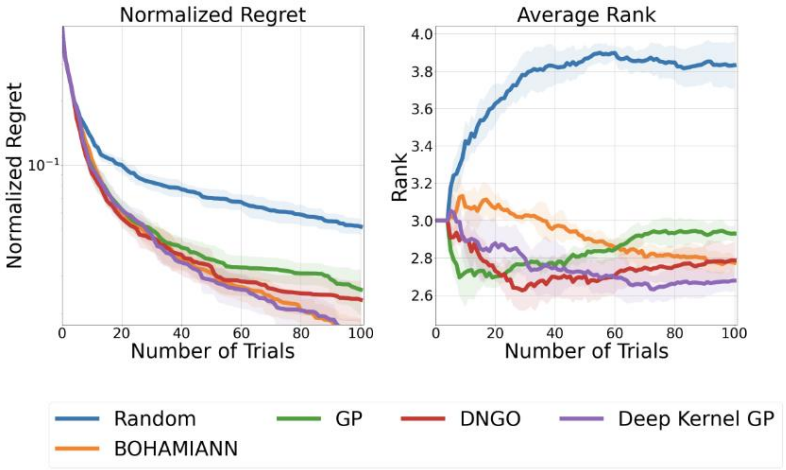
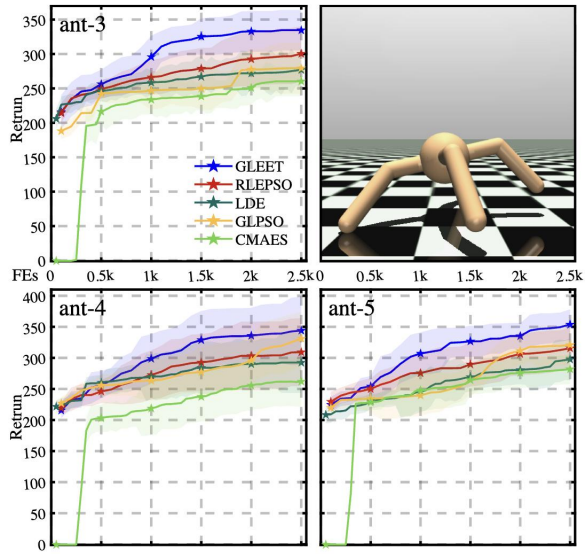
Evolution of Benchmark: Black-Box Optimization Benchmark Design through Large Language Model

Chen Wang*, Sijie Ma*, Zeyuan Ma*, Yue-Jiao Gong

The Crucial Role of Benchmarks in BBO







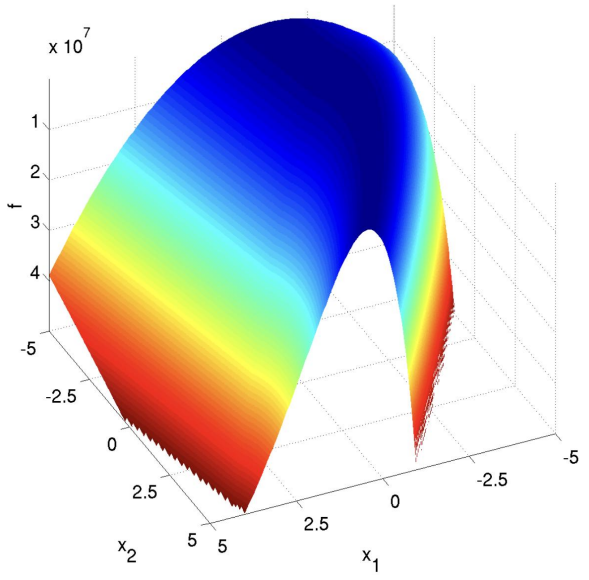
Widespread Applications of BBO:

Inherently hard problems encountered across diverse real-world scenarios.

Algorithm Evaluation: Fair comparison and performance coherence.

Training Learnable Optimizers: The need of a large number of similar instances.

Limitations & Prior Explorations in Benchmark Design



3.12 Bent Cigar Function

$$f_{12}(\mathbf{x}) = z_1^2 + 10^6 \sum_{i=2}^{D} z_i^2 + f_{\text{opt}} \tag{12}$$

- $\mathbf{z} = \mathbf{R}\, T_{\text{asy}}^{0.5}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$

Properties A ridge defined as $\sum_{i=2}^{D} z_i^2 = 0$ needs to be followed. The ridge is smooth but very narrow. Due to $T_{\text{asy}}^{1/2}$ the overall shape deviates remarkably from being quadratic.

- conditioning is about 10^6, rotated, unimodal

Bottlenecks of Traditional Benchmarks (e.g., CoCo-BBOB):

- Costly Labor: Deep expertise and massive labor input required.
- Expert Bias: Human-crafted functions introduce bias and constrain diversity.
- Limited Instances: Cannot keep pace with the swift emergence of novel BBO tasks.

Limitations & Prior Explorations in Benchmark Design

Function ID	1	2	3	4	5	6	7	8	9	10	11	12
Scale Factor	11.0	17.5	12.3	12.6	11.5	15.3	12.1	15.3	15.2	17.4	13.4	20.4
Function ID	13	14	15	16	17	18	19	20	21	22	23	24
Scale Factor	12.9	10.4	12.3	10.3	9.8	10.6	10.0	14.7	10.7	10.8	9.0	12.1

Table 1: Final scale factors used to generate MA-BBOB problems.

:

$$\text{MA-BBOB}(\vec{W},\vec{I},\vec{X}_{\text{opt}})(x) = R^{-1}\left(\sum_{i=1}^{24} W_i \cdot R_i\left(F_{i,I_i}(x-\vec{X}_{\text{opt}}+O_{i,I_i}) - F_{i,I_i}(O_{i,I_i})\right)\right).$$

$$R_i(x) = \frac{\max(\log_{10}(x),-8)+8}{S_i}$$

$$R^{-1}(x) = 10^{(10\cdot x)-8}.$$

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 $\lvert Y\rvert < \epsilon$, 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 $\lvert X\rvert < \epsilon \wedge Y = -1$, 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: $\lvert X\rvert$
$Tanh$	Hyperbolic tangent activation: $Tanh(X)$
exp	Natural exponential function: e^X
log	If $\lvert X\rvert < \epsilon$, outputs 0; otherwise $\log_{10}(\lvert X\rvert)$ (implicit absolute value ensures $X \in \mathbb{R}^+$)
$sqrt$	Computes the square root of the absolute value $\sqrt{\lvert X\rvert}$ to prevent invalid inputs (negative values).

Table 2: Symbol Set

Prior Automation Attempts:

- Hand-crafted extensions
- Affine Recombination
- Genetic Programming

Previous methods are either resource-consuming or heavily dependent on deep domain expertise.

Paradigm Shift: LLM-Driven Automation

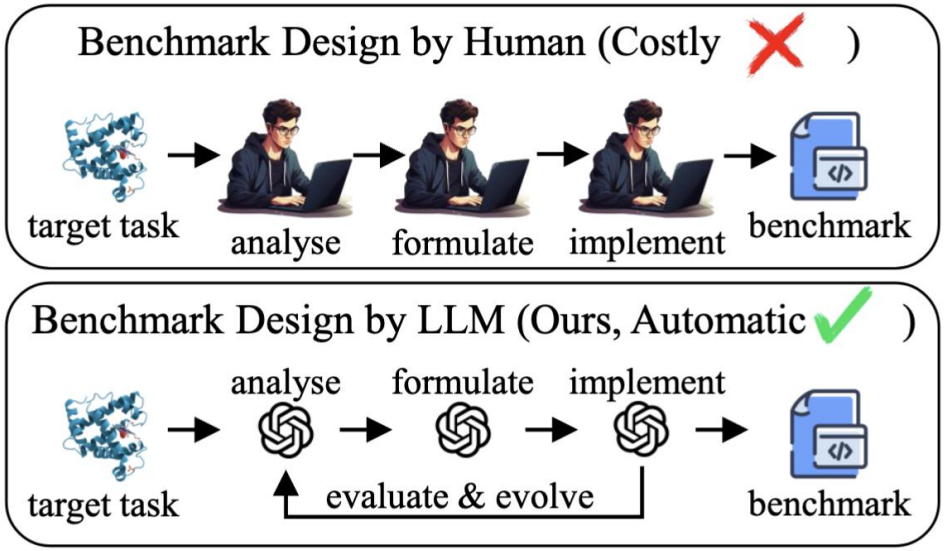


Figure 1. Paradigm shift from human-crafted benchmark design toward LLM-based automation.

The Rise of LLM Program Evolution:
Iterative in-context learning empowers open discovery (e.g., FunSearch, EoH, LLaMEA)

Evolution of Benchmark (EoB):
A compact, self-contained, and fully automated BBO benchmark designer.

The Core Philosophy of EoB

Rethinking Benchmark Generation:
We formulate benchmark design as a rigorous **Multi-Objective Optimization Problem.**

$$\min_{f \in \mathbb{F}} (\mathcal{O}_1(f), ..., \mathcal{O}_M(f)), \qquad (1)$$

The Two Conceptual Goals:
The generated benchmark must "look and feel" like the target real-world problems.
The benchmark must be capable of exposing the performance gaps between different search algorithms.

The Engine:
LLM serves as the evolutionary agent, searching through the vast space of mathematical programs to find the optimal trade-off (Pareto front) between these two goals.

Objective 1 - Landscape Similarity Indicator (LSI)

The Goal: Measure the topological similarity between a generated program f and target tasks F_{target}.

Feature Extraction via **NeurELA**:
•Replaces computationally heavy traditional methods.
•Uses a Dual-Attention Transformer to extract a compact 16-dimensional z_f.

$$\mathcal{O}_1(f|F_{target}) = \frac{1}{1 + \frac{1}{\sigma^2} \min_{p \in F_{target}} ||z_f - z_p||_2}, \quad (3)$$

$$\sigma = \frac{1}{H} \sum_{i=1}^{H} \min_{j \neq i} ||z_{p_i} - z_{p_j}||_2, \quad (4)$$

A higher LSI score means the generated function successfully falls into the "gravitational pull" of the target problem family.

Objective 2 - Algorithm Distinguishing Capability (ADC)

The Goal: Fulfill the fundamental role of a benchmark by effectively evaluating and distinguishing the performance variations across different BBO algorithms.

Evaluation Portfolio:
10 representative BBO algorithms covering diverse search logics

Metric Formulation:

$$\mathcal{O}_2(f|\Lambda) = \text{std}(\frac{1}{B}\sum_{i=1}^{B}\vec{Obj}_{i,:}), \qquad (5)$$

Scalarization: We use Penalty-based Boundary Intersection (PBI) to aggregate LSI and ADC, providing a scalar feedback score to the LLM.

PSO	Single-Objective Optimization	BBO	Particle swarm optimization	1995
DE	Single-Objective Optimization	BBO	Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces	1997
CMAES	Single-Objective Optimization	BBO	Completely Derandomized Self-Adaptation in Evolution Strategies	2001
SHADE	Single-Objective Optimization	BBO	Success-history based parameter adaptation for differential evolution	2013
GLPSO	Single-Objective Optimization	BBO	Genetic Learning Particle Swarm Optimization	2015
SDMSPSO	Single-Objective Optimization	BBO	A Self-adaptive Dynamic Particle Swarm Optimizer	2015
SAHLPSO	Single-Objective Optimization	BBO	Self-Adaptive two roles hybrid learning strategies-based particle swarm optimization	2021
JDE21	Single-Objective Optimization	BBO	Self-adaptive Differential Evolution Algorithm with Population Size Reduction for Single Objective Bound-Constrained Optimization: Algorithm j21	2021
MADDE	Single-Objective Optimization	BBO	Improving Differential Evolution through Bayesian Hyperparameter Optimization	2021
NLSHADELBC	Single-Objective Optimization	BBO	NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 Numerical Optimization	2022

The Evolutionary Pipeline

Algorithm 1 Evolution of Benchmark (EoB)

Input: Large Language Model $\mathcal{LLM}$, Target BBO tasks F_{target}, BBO algorithm pool Λ, #Reference vectors N, #Evolution generations T, Pareto set $PF = \emptyset$.

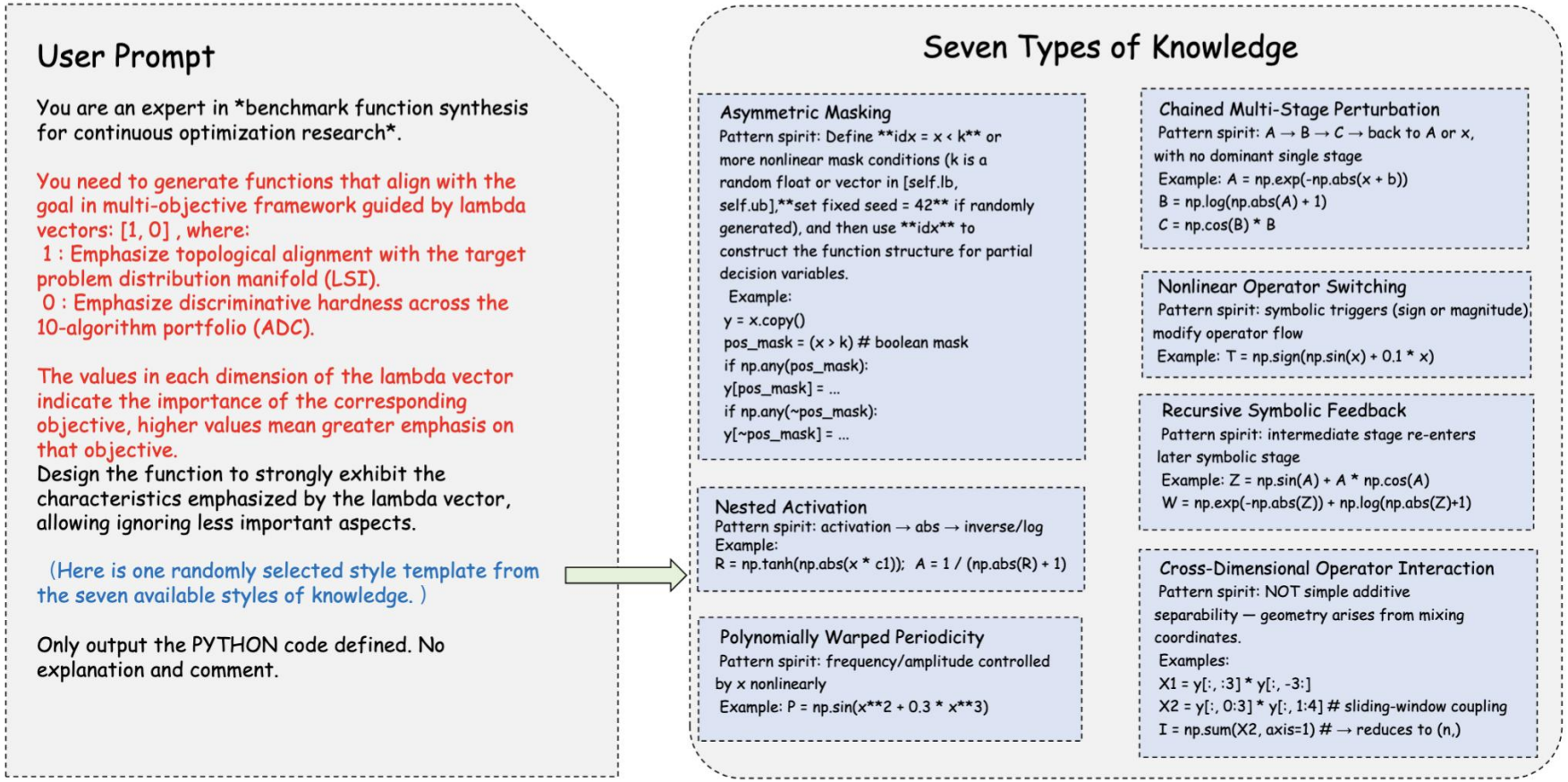
Output: Finally obtained benchmark

1: Let $t = 0$, initialize population $\{f_i^t\}_{i=1}^N$ (Sec. 3.2.1);
2: Evaluate each f_i^t (Sec. 3.2.2), update PF (Sec. 3.2.5);
3: **while** $t < T$ **do**
4: Reproduce offspring $\{f_i^{t+1}\}_{i=1}^N$ (Sec. 3.2.3);
5: Evaluate each f_i^{t+1} (Sec. 3.2.2);
6: Select elites from $\{f_i^t\}_{i=1}^N \cup \{f_i^{t+1}\}_{i=1}^N$ (Sec. 3.2.4);
7: Update PF (Sec. 3.2.5) with elites, $t \leftarrow t + 1$;
8: **end while**
9: **return** PF

the 5-step loop (Initialization -> Evaluation -> Evolution -> Selection -> Pareto Front).

Initialization

The Initialization Challenge:
Without guidance, LLMs tend to generate highly homogeneous functions (poor diversity).
Injecting Landscape Knowledge:
We inject one of **7 specific landscape construction templates** into the initial prompt.

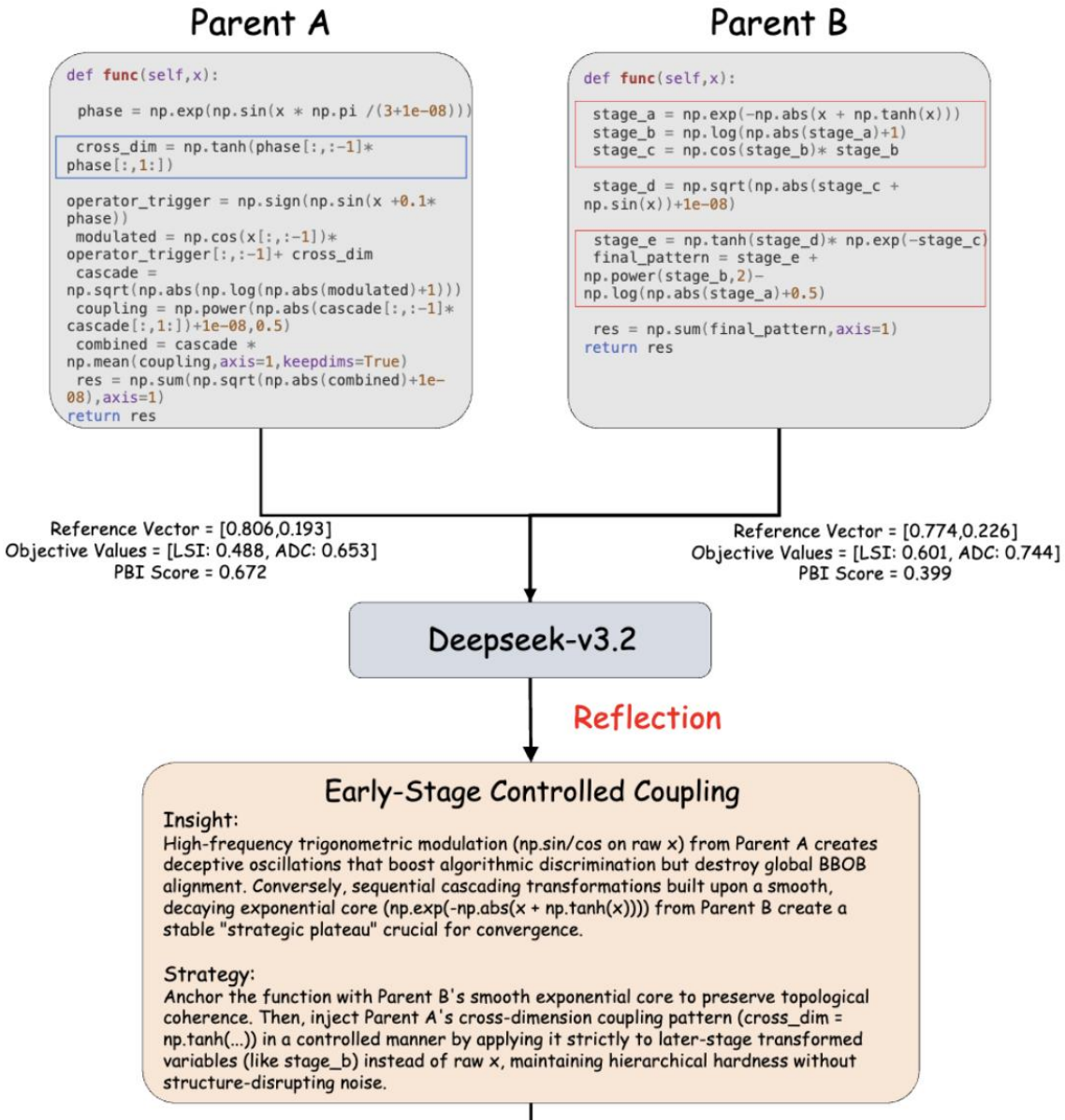


The Evolution Engine - Phase 1: Reflection

The Causal Analysis:
The LLM acts as an optimization expert comparing pairs of generated functions (Winner vs. Loser).

Deep Analysis Rules :
•**Successful Patterns**:
Identify the exact mathematical structure in the Winner that minimized the PBI score.
•**Valuable Genes**:
If the Loser scored highly on a single objective (e.g., high ADC but poor LSI), extract that specific code snippet for preservation.
•**Trade-off Insights**:
Explicitly identify structures that boost one objective but ruin another (e.g., extreme oscillation boosts hardness but destroys convergence).

The Evolution Engine - Phase 1: Reflection



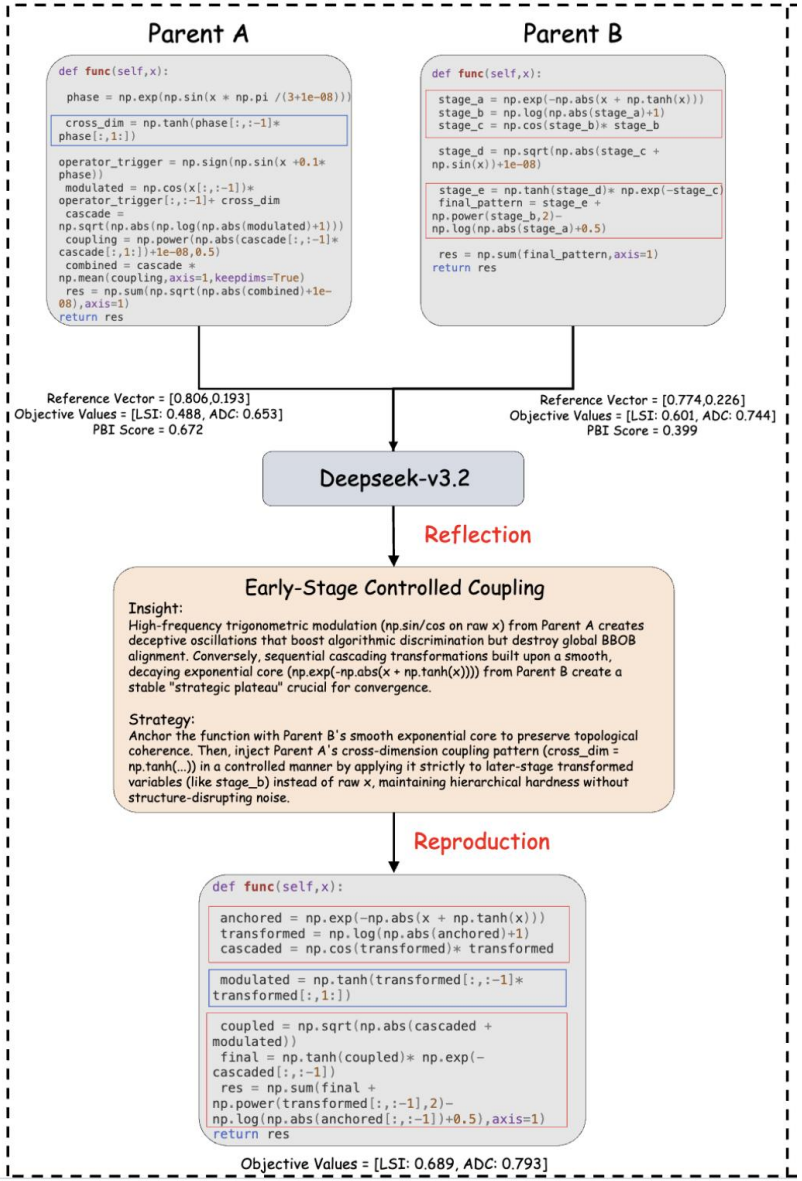
The Evolution Engine - Phase 2: Reproduction

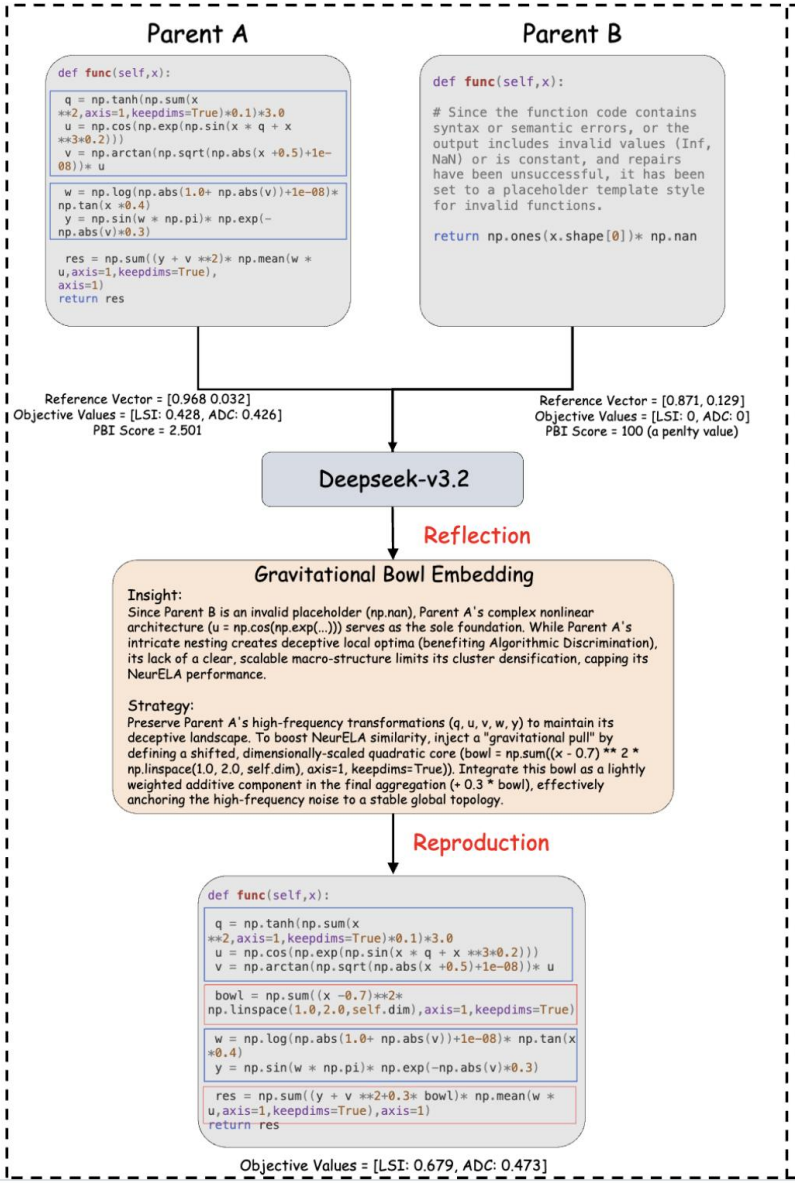
Code Synthesis:
The LLM acts as the "Crossover / Mutation" operator to generate the offspring.

The Recipe:
•Inherit: Retain the successful topology from the Winner.
•Transfer: Assimilate the beneficial "genes" from the Loser.
•Innovate: Apply strategic mathematical perturbations based on the target Reference Vector.

Strict Refactoring:
Enforces functional composition and limits intermediate variables to prevent "spaghetti code."

The Evolution Engine - Phase 2: Reproduction





Experimental Setup

LLM Engine:
DeepSeek-v3.2 (no thinking) serves as the core evolutionary agent.

EoB Hyperparameters:
•Population Size (N): 32
•Evolution Generations (T): 10
•Default Search Dimension (D): 10

Algorithm Portfolio:
10 representative BBO algorithms evaluated over 10 independent runs

Involved Benchmarks / Target Tasks:
•Classic Evaluation: CoCo-BBOB (24 functions).
•Realistic Tasks: UAV path planning, Hyperparameter Optimization

Scenario 1 - Enhancing Traditional Benchmarking

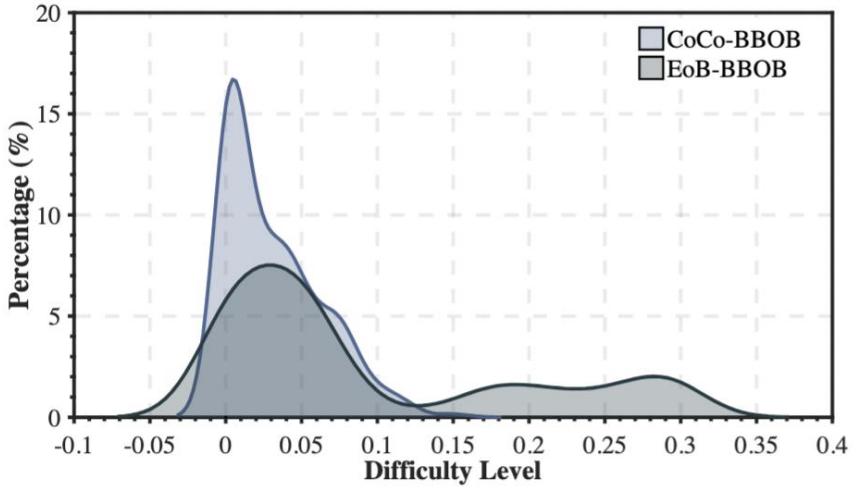


Figure 3. Benchmark diversity of CoCo-BBOB and EoB-BBOB.

The Objective:
Construct a new benchmark (EoB-BBOB) to evaluate classic BBO algorithms more comprehensively.

Key Observations:
•Hand-crafted functions in CoCo-BBOB show low difficulty diversity and concentrated algorithm distinguishing capabilities.

EoB-BBOB Advantage:
•Automatically designs a more uniform, diversified, and challenging benchmark space through program evolution.

Scenario 2 - Empowering Learnable Optimizers

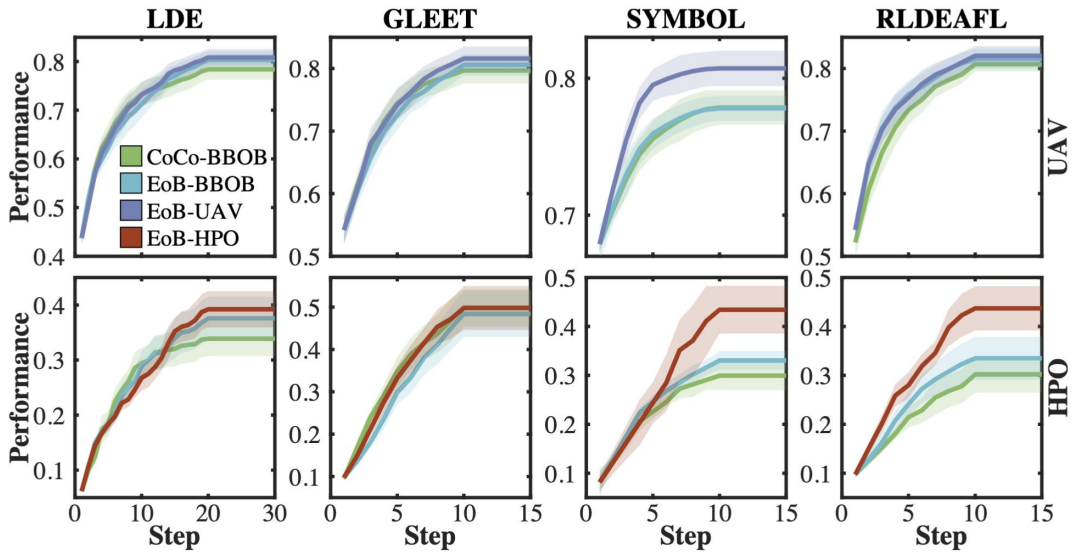


Table 2. The final test-performance of GLHF trained by different baseline benchmarks.

Scenario	CoCo-BBOB	EoB-BBOB	EoB-Scenario
UAV	8.380E-01 ±5.193E-02	7.508E-01 ±1.149E-01	**8.484E-01** ↑ **±3.854E-02**
HPO	5.144E-01 ±2.113E-01	5.214E-01 ±1.769E-01	**6.580E-01** ↑ **±1.933E-01**

Table 1. The wall-clock evaluation time using EoB or target realistic tasks for MetaBBO approaches, less is better.

Algorithm \ Scenario		UAV		HPO	
		Real	EoB	Real	EoB
LDE	Normal	30.38 min	**3.124 s**	23.72 min	**2.923 s**
	Large	≈ 7 h	**28.32 s**	≈ 4 h	**27.43 s**
GLEET	Normal	26.23 s	**0.256 s**	14.68 s	**0.231 s**
	Large	284.2 s	**0.287 s**	146.6 s	**0.274 s**
SYMBOL	Normal	43.68 s	**0.423 s**	190.8 s	**0.412 s**
	Large	78.95 s	**0.528 s**	312.1 s	**0.547 s**
RLDEAFL	Normal	98.24 s	**0.921 s**	15.98 s	**0.852 s**
	Large	22.91 min	**1.442 s**	205.7 s	**1.352 s**

Setup:
Train MetaBBOs (LDE, GLEET, SYMBOL, RLDEAFL and GLHF) on different benchmarks (CoCo-BBOB vs. EoB-UAV/EoB-HPO) Test on zero-shot realistic UAV/HPO tasks

The Result:
Optimizers trained on EoB-generated suites vastly outperform those trained on CoCo-BBOB.
EoB provides a high-quality, task-aligned training source.

Deep Dive - Why Does EoB Work?

Table 3. Test-performance consistency between baseline benchmarks and realistic tasks.

Scenario	CoCo-BBOB	EoB-BBOB	EoB-Scenario
UAV	0.003	0.396	**0.453** ↑
HPO	0.694	0.702	**0.761** ↑

Performance Predictivity:
We measure the performance consistency using:

$$\mathbb{C}(B_{source}, B_{target}) = 1 - \mathcal{W}(\vec{S}_{source}, \vec{S}_{target}),$$

$$\vec{S}_{source} : \{\frac{1}{rank_i}\}_{i=1}^{10}$$

The Mechanism:
•BBO algorithms exhibit highly consistent performance rankings on EoB-generated benchmarks as they do on the actual target tasks.
•This faithful topological alignment ensures valid knowledge transfer for MetaBBO policies.

Ablation Studies

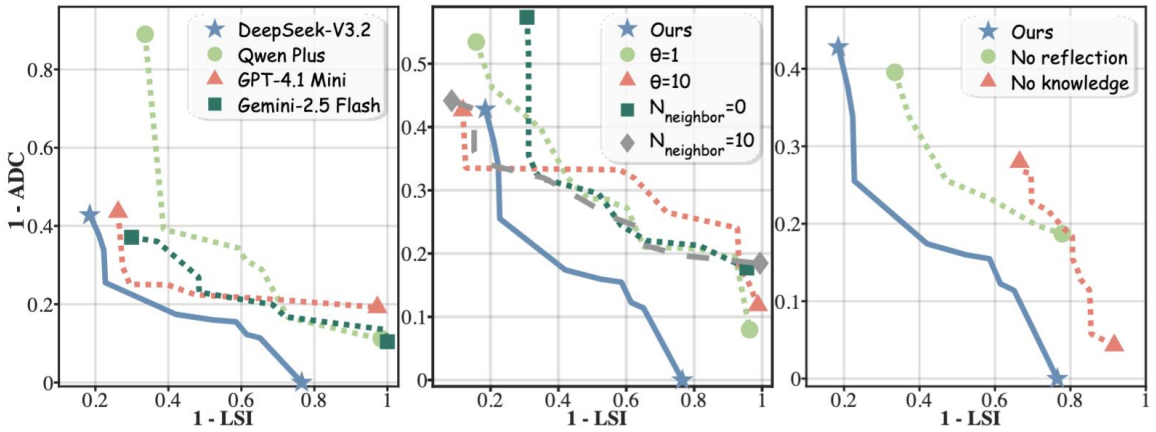


Figure 5. The ablation results shown as the pareto front set PF found by different baselines.

Table 4. Ablation on landscape feature representations in LSI. Values are final MetaBBO test performances, and larger is better.

Feature Representation	LDE	GLEET	SYMBOL	RLDEAFL
NeurELA	**0.7621**	**0.8342**	**0.7428**	**0.8593**
DeepELA	0.7509	0.7923	0.7295	0.7967
ELA: Meta-Model	0.7528	0.7685	0.7324	0.7891
ELA: Distribution	0.7424	0.7858	0.7188	0.7812
ELA: Convexity	0.7495	0.7612	0.7397	0.7954
ELA: Local Search	0.7487	0.7705	0.7242	0.7761
ELA: Levelset	0.7398	0.7902	0.7315	0.7905
ELA: Curvature	0.7462	0.7698	0.7332	0.7747

Reflection & Knowledge:

Removing the Reflection stage severs the feedback loop (degrading efficiency).

Removing Landscape Knowledge causes LSI stagnation.

Feature Extractor:

NeurELA strictly outperforms traditional ELA and DeepELA, accurately capturing target landscape topologies.

Parameter Sensitivity

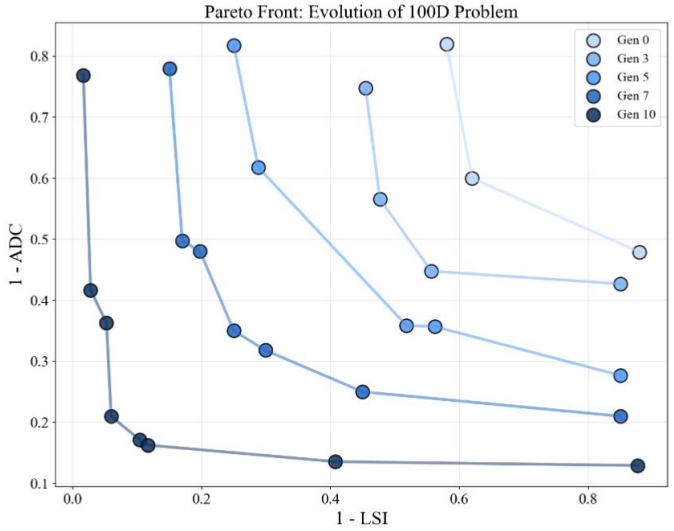


Figure 11. Pareto front evolution of EoB under the 100D setting.

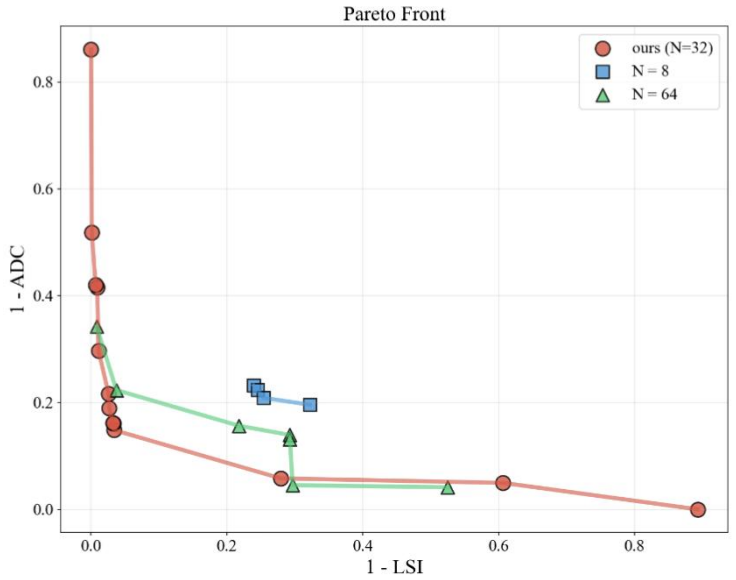


Figure 12. Pareto fronts obtained by EoB with different population sizes.

Dimension:

Scaled the search space to 100D CoCo-BBOB problems.

EoB progressively discovers balanced solutions (high LSI & ADC) even in 100D spaces where manual design intuition completely fails.

Population:

•N=8: Yields a narrow Pareto front due to insufficient search diversity.

•N=64: Diminishing returns with significantly higher computational API costs.

•The Sweet Spot: N=32 delivers the optimal trade-off between evolutionary exploration capability and computational efficiency.

Our Team Page