EvoX: Meta-Evolution for Automated Discovery

Shu Liu[1*], Shubham Agarwal[1*], Monishwaran Maheswaran[1], Mert Cemri[1], Zhifei Li[1], Qiuyang Mang[1], Ashwin Naren[1], Ethan Boneh[2], Audrey Cheng[1], Melissa Z. Pan[1], Alexander Du[1], Kurt Keutzer[1], Alexandros G. Dimakis[1,3], Koushik Sen[1], Matei Zaharia[1], Ion Stoica[1]

Affiliations

[1]UC Berkeley [2]Stanford University [3]Bespoke Labs

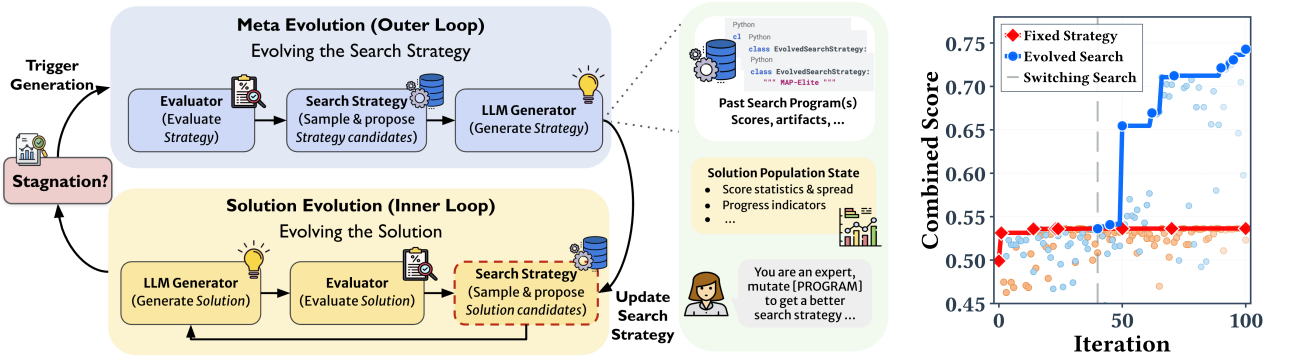
Abstract: Recent work such as AlphaEvolve has shown that combining LLM-driven optimization with evolutionary search can effectively improve programs, prompts, and algorithms across domains. In this paradigm, previously evaluated solutions are reused to guide the model toward new candidate solutions. Crucially, the effectiveness of this evolution process depends on the search strategy: how prior solutions are selected and varied to generate new candidates. However, most existing methods rely on fixed search strategies with predefined knobs (e.g., explore–exploit ratios) that remain static throughout execution. While effective in some settings, these approaches often fail to adapt across tasks, or even within the same task as the search space changes over time.

We introduce EvoX, an adaptive evolution method that optimizes its own evolution process. EvoX jointly evolves candidate solutions and the search strategies used to generate them, continuously updating how prior solutions are selected and varied based on progress. This enables the system to dynamically shift between different search strategies during the optimization process. Across nearly 200 real-world optimization tasks, EvoX outperforms existing AI-driven evolutionary methods including AlphaEvolve, OpenEvolve, GEPA, and ShinkaEvolve on the majority of tasks.

Table 1: **EvoX across optimization tasks.** We compare EvoX against strong human-designed methods and prior AI discovery systems, including AlphaEvolve, GEPA, ShinkaEvolve, and OpenEvolve, across mathematical optimization, systems performance optimization (e.g., GPU-to-model placement), and algorithm engineering tasks (e.g., Frontier-CS). For Frontier-CS, we report median scores over 172 competitive programming problems. Arrows indicate whether higher (↑) or lower (↓) scores are better. All *open* AI frameworks are evaluated under a fixed budget of 100 iterations, and the best result among them is reported as "AI Best." AlphaEvolve results are taken directly from its original publication.

Task	Human Best	AlphaEvolve	EvoX
MATHEMATICS			
Circle-Packing (↑)	2.634	2.635	**2.636**
MinMaxMinDist (↓)	4.16584	4.16579	**4.16578**
Task	**Human Best**	**AI Best**	**EvoX**
SYSTEMS OPTIMIZATION			
Cloud Transfer (↓)	626.24	645.72	**623.69**
GPU Placement (↑)	21.89	26.26	**30.52**
Txn Scheduling (↑)	2724.8	4329	**4347.8**
ALGORITHMS			
Frontier-CS (↑)	–	56.2	**75.5**

* denotes equal contribution. Code is available at https://github.com/skydiscover-ai/skydiscover.



(a) Overall system architecture (b) Evolved search (blue) breakthroughs

Figure 1: **Evolving the search strategy.** (a) *System architecture.* EvoX has two coupled loops: an inner loop that evolves solutions, and an outer loop that evolves the *search strategy* that governs generation. (b) *Effect of search evolution.* A strategy with fixed exploration-exploitation ratio (MAP-Elites, red) stagnates, while an evolving search strategy (blue) produces discrete performance breakthroughs.

1. Introduction

LLM-driven optimization combined with evolutionary search has enabled numerous scientific breakthroughs across domains including mathematics [22], systems performance optimization [7], and competitive programming [15]. Typically, LLM-driven evolutionary systems maintain a population of candidate solutions. At each step, a *search strategy* selects a subset of previously evaluated solutions and constructs a prompt, which is passed to a generator model (typically an LLM) to produce new candidates. These candidates are then evaluated and added back to the population.

Critically, the effectiveness of the evolution process relies on the search strategy, which determines both (i) which candidates are selected from the population and (ii) how new solutions are proposed from them (e.g., via refinement, structural variation, or combining multiple ideas). These choices directly influence what the generator attempts next and which regions of the solution space to explore.

Existing LLM-driven evolutionary systems often rely on fixed search strategies with hand-specified parameters. For example, AlphaEvolve [22] employs MAP-Elites with predefined population database structures and selection ratios. OpenEvolve [28] similarly relies on static elite and diversity heuristics. ShinkaEvolve [17] takes a step toward adaptivity by incorporating bandit-based selection on LLM generators. However, key search strategy knobs remain manually configured. For example, the exploitation ratio is fixed, controlling how often top solutions are refined.

In practice, a fixed search strategy often fails to generalize across problems or across different stages of optimization. This can lead to stagnation during the search and requires manual retuning to make progress. Some tasks benefit primarily from repeated refinement of a strong candidate, where small local modifications incrementally improve an existing solution. For example, in packing problems [22, 11], adjusting a few placements in a near-feasible configuration can yield further gains. In contrast, other tasks require qualitatively different solution structures. For instance, in a GPU-model placement problem, refining a least-loaded heuristic quickly plateaus, and further progress requires reformulating the problem as a bin-packing assignment.

Even within a single optimization run, the effectiveness of a search strategy can change over time. In a multi-objective signal processing task (Figure **1**(b)), a MAP-Elites style strategy (red line) makes rapid

early progress but later stagnates. Switching to a strategy that explicitly samples candidates along different trade-off objectives (blue line) enables continued improvement.

Motivated by this observation, EvoX frames LLM-driven optimization as a meta-learning problem in which *the search strategy itself is treated as an evolvable object*. Rather than fixing a hand-tuned search strategy, EvoX operates as a two-level evolution process comprising a *solution-evolution* loop and a *meta-evolution* loop (Figure **1**(a)).

The solution-evolution loop generates candidate solutions under the standard LLM-driven evolutionary search paradigm. At each step, a search strategy selects prior candidates and determines how to generate new ones, for example by sampling high-performing solutions and applying a variation operator (e.g., refinement or structural variation). The resulting prompt is passed to the LLM, which produces a new candidate that is evaluated and added back to the population.

The meta-evolution loop periodically updates the search strategy. Each strategy is deployed for a window of solution-evolution iterations and evaluated by the progress it induces on the downstream task (e.g., improvement rate). When progress stagnates, the meta-evolution loop generates a new search strategy using the LLM, conditioned on prior strategies, their observed performance, and the current state of the solution population. This loop enables EvoX to evolve the search strategy itself: selecting, mutating, and replacing strategies based on their effectiveness. Because the performance of a strategy depends on the evolving population, EvoX explicitly conditions strategy generation on population-level signals, allowing it to adapt as the search space changes.

Contributions. We evaluate EvoX across nearly 200 real-world optimization tasks spanning mathematics, algorithms, and scientific research benchmarks (Table **1** and Section 6). Starting from a simple random-sampling search strategy, EvoX consistently outperforms existing LLM-driven evolutionary frameworks, including OpenEvolve, ShinkaEvolve, and GEPA [28, 17, 1] on the majority of tasks (e.g., 96% of the math and system optimization benchmarks). It also often matches or surpasses the best human-designed solutions. In summary, we make the following contributions:

1. We formalize LLM-driven optimization as a two-level process that separates solution evolution from search strategy evolution.
2. We introduce EvoX, a method that dynamically evolves search strategies based on improvements in the optimization objective
3. We demonstrate consistent improvements over prior methods across nearly 200 problems, and characterize the cost, scaling behavior, and adaptation dynamics of the search strategy evolution.

2. Related Work

Context and Memory Management. Many recent LLM systems iteratively improve solutions by incorporating feedback from prior attempts. A key challenge in this setting is how to store, summarize, and present past information to guide future generations. ACE [37] treats context as editable objects, MemGPT [24] introduces hierarchical memory, and Reflexion [30] and Self-Refine [19] incorporate model-generated feedback into future prompts. These approaches improve how prior experience is organized and reused, enabling more effective iterative refinement.

One prominent paradigm that builds on this foundation is evolutionary search, which treats prior solutions as a population and explicitly selects and varies candidates over time.

LLM-Guided Evolutionary Search. Recent work applies evolutionary search to guide LLM-driven optimization, spanning prompt optimization [10, 13, 18, 35, 32, 9] and program or algorithm discovery [22, 28, 17, 1, 3, 14, 31]. These systems differ primarily in how candidates are selected and varied. AlphaEvolve [22] selects candidates using MAP-Elites, GEPA [1] selects along Pareto frontiers, and OpenEvolve [28] and ShinkaEvolve [17] emphasize diversity-driven selection. CodeEvolve [3] integrates LLM-based generation within an island-based genetic algorithm, while DeltaEvolve [16] models semantic deltas between candidates. PACEvolve [34] introduces mechanisms such as hierarchical context pruning and momentum-based backtracking to improve stability. However, these systems still operate under predefined search strategies with manually designed knobs.

Some recent work introduces learning-based adaptation mechanisms. SOAR [25] alternates between search and model fine-tuning, while ThetaEvolve [33], TTT-Discover [36], and FLEX [4] apply reinforcement learning to improve the generator model. These approaches adapt the generator model itself, but do not adapt the search strategy governing candidate selection and variation.

Meta-Learning and Learning to Optimize. Meta-learning and learned optimization frameworks treat the optimization procedure itself as the object of adaptation rather than a fixed component [21, 5]. Prior work explores learned optimizers, symbolic discovery of update rules [6], gradient-based meta-learning [2], and reinforcement learning-based optimizers [26]. These works demonstrate that adapting the optimization process itself can significantly improve search efficiency. Our work extends this principle to LLM-driven evolutionary search.

EvoX: Meta-Evolving the Search Strategy. EvoX builds on these lines of work by treating the search strategy itself as an evolvable object. Rather than relying on fixed candidate selection and variation mechanisms, EvoX evolves the search strategy through evolutionary feedback, dynamically adapting how candidates are selected and varied across different optimization stages and heterogeneous solution landscapes.

3. Problem Formulation

We formalize LLM-driven evolutionary search following standard abstractions from prior work [27, 22, 28, 1, 17]. Candidate solutions are generated by a language model, evaluated by a task-specific evaluator, and iteratively evolved under a search strategy and a fixed evaluation budget.

Candidate solution evaluation. Let $\mathcal{X}$ denote the space of candidate solutions (e.g., programs or prompts). Each candidate $x \in \mathcal{X}$ is evaluated by an evaluator

$$E(x) \to \big(s(x),\, a(x)\big),$$

which returns a scalar score $s(x) \in \mathbb{R}$ together with auxiliary artifacts $a(x)$ (e.g., logs, traces, or any feedback).

Solution population database. The optimization proceeds for T sequential evaluation steps. Let

$$\mathcal{D}_t = \{(x_i, s_i, a_i)\}_{i=1}^{t}, \quad \mathcal{D}_0 = \emptyset,$$

denote the database of all candidate solutions evaluated up to step t. At each step, a new candidate is generated, evaluated, and appended to the database to form $\mathcal{D}_{t+1}$.

Search strategy. A search strategy $S \in \mathcal{S}$ specifies how the next-generation LLM input is constructed from the current database $\mathcal{D}_t$. Concretely, S defines the construction of the generation context:

$$C_S(\mathcal{D}_t) \to (x_{\text{par}},\, \pi,\, \mathcal{I}),$$

which selects (i) one or more *parent* candidate(s) $x_{\text{par}} \in \mathcal{D}_t$ to be modified, (ii) a *variation operator* π expressed in the prompt that specifies how the parent can be modified, and (iii) optionally, an *inspiration set* $\mathcal{I} \subseteq \mathcal{D}_t$ (e.g., diverse candidates) that provides additional exemplars. This abstraction aligns with prior LLM-driven evolutionary systems [17, 28].

Given $(x_{\text{par}}, \pi, \mathcal{I})$, the solution generator (implemented by an LLM) produces a new candidate

$$x' \sim \mathcal{G}_{\text{sol}}(\cdot \mid x_{\text{par}}, \pi, \mathcal{I}),$$

which is evaluated and appended to the database.

Variation operators. The operator π specifies the *kind* of modification requested in a generation step. We use three variation operators: ① **local refinement** for fine-grained edits (exploitation), ② **structural variation** for coarse-grained redesigns (exploration), and ③ **free-form variation**, which imposes no constraints on the edit scope.

The semantics of these operators are task-dependent. For example, in code or algorithmic tasks, refinement may tune parameters, reorder logic, or edit localized code blocks, whereas structural variation may switch algorithm families or restructure the overall design. At the start of each problem, we instantiate these operators using a lightweight model (Section 6) to generate a small set of operator-specific prompts from the problem description. During optimization, the search strategy selects among these operators to control the intended type of variation applied to the parent candidate.

Optimization goal. Search strategies shape optimization behavior, and our goal is to *adaptively select and improve the search strategy* to maximize the final best score

$$\max_{(x,s,a)\in\mathcal{D}_T} s$$

under a fixed evaluation budget of T steps.

4. Co-evolving Solution and Search Strategy

EvoX addresses the problem of iteratively improving solution quality under a fixed evaluation budget defined in Section 3 by *co-evolving* two components: (i) the solution population $\mathcal{D}_t$, and (ii) the search

Algorithm 1 EvoX: Two-level evolution process

Require: Budget T, window W, stagnation threshold τ
Require: Evaluator E; solution generator $\mathcal{G}_{\text{sol}}$; strategy generator $\mathcal{G}_{\text{str}}$
Require: Population descriptor $\phi(\cdot)$; validity test $\textsc{Valid}(\cdot)$
Require: Initial solution database $\mathcal{D}_0$; initial strategy S_0
$\mathcal{D}_t \leftarrow \mathcal{D}_0$; $S_t \leftarrow S_0$; $\mathcal{H} \leftarrow \emptyset$; $t \leftarrow 0$
while $t < T$ **do**
  Phase I: Solution evolution under S_t (one window)
  $\phi_t \leftarrow \phi(\mathcal{D}_t)$
  $s_{\text{start}} \leftarrow \max_{(x,s,a)\in\mathcal{D}_t} s$
  for $i = 1$ to W **do**
    if $t \geq T$ **then break**
    end if
    $(x_{\text{par}}, \pi, \mathcal{I}) \sim C_{S_t}(\mathcal{D}_t)$
    $x' \sim \mathcal{G}_{\text{sol}}(\cdot \mid x_{\text{par}}, \pi, \mathcal{I})$
    $(s', a') \leftarrow E(x')$
    $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \{(x', s', a')\}$; $t \leftarrow t+1$; $\mathcal{D}_t \leftarrow \mathcal{D}_{t+1}$
  end for
  Phase II: Progress monitoring
  $s_{\text{end}} \leftarrow \max_{(x,s,a)\in\mathcal{D}_t} s$
  $\Delta \leftarrow s_{\text{end}} - s_{\text{start}}$
  $J_t \leftarrow \Delta \log(1 + s_{\text{start}}) / \sqrt{W}$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(S_t, \phi_t, J_t)\}$
  Phase III: Strategy evolution (on stagnation)
  if $\Delta < \tau$ **then**
    $S' \sim \mathcal{G}_{\text{str}}(\cdot \mid \mathcal{H}, \phi(\mathcal{D}_t))$
    if $\textsc{Valid}(S')$ **then** $S_t \leftarrow S'$
    end if
  end if
end while
return $\arg\max_{(x,s,a)\in\mathcal{D}_T} s$

strategy S that constructs the next-generation LLM context. Specifically, EvoX proceeds through the three-step process shown in Algorithm 1: (1) evolving the solution database under a fixed search strategy, (2) monitoring population performance, and (3) updating the search strategy when progress stalls, using feedback from previous strategy deployments and the current state of the solution population.

4.1. Solution evolution under the current strategy

Given the current database $\mathcal{D}_t$ and an active strategy S_t, EvoX generates and evaluates new candidates as defined in Section 3: the strategy constructs a generation context $(x_{\text{par}}, \pi, \mathcal{I}) \sim C_{S_t}(\mathcal{D}_t)$, the generator produces x', and the evaluator appends (x', s', a') to the database.

A search strategy controls both *what* the model sees (through parent selection and inspiration construction) and *how* the selected parent is transformed (via the variation operator π).

4.2. Progress monitoring and strategy updates

A search strategy does not affect a single candidate in isolation, but shapes a *sequence* of generated candidates. Accordingly, its effectiveness can only be assessed over multiple evaluation steps rather than from a single outcome. For this reason, EvoX monitors progress over a sliding window of the most recent W evaluation steps.

Let t denote the start of the current monitoring window and define

$$s_{\text{start}} = \max_{(x,s,a)\in\mathcal{D}_t} s, \quad s_{\text{end}} = \max_{(x,s,a)\in\mathcal{D}_{t+W}} s, \quad \Delta = s_{\text{end}} - s_{\text{start}}. \tag{1}$$

We treat Δ as the primary signal of strategy efficacy. If Δ falls below a stagnation threshold τ, EvoX triggers a strategy update; otherwise, it continues with the current strategy. This design makes strategy switching *demand-driven* rather than *periodic at a fixed interval*, avoiding unnecessary updates when the current strategy is effective.

Search strategy evaluation. After W evaluation steps have elapsed, EvoX computes a performance score for the strategy employed during that window:

$$J(S_t \mid \mathcal{D}_t) = \frac{(s_{\text{end}} - s_{\text{start}})\ \log(1 + s_{\text{start}})}{\sqrt{W}} \tag{2}$$

The $\log(1 + s_{\text{start}})$ term upweights improvements achieved from higher starting scores, rewarding strategies that drive progress near the frontier where gains are typically harder to obtain. The $\sqrt{W}$ normalization accounts for window length.

4.3. Meta-evolving the search strategy

Adapting search strategies is challenging because their effectiveness is *state-dependent*. For instance, a strategy that drives rapid progress at one stage of search may become ineffective as the solution population evolves. This arises from the inherently non-stationary nature of evolutionary optimization: as the database grows, the distribution of candidate quality, diversity, and variation outcomes shifts, altering which search behaviors are beneficial.

EvoX addresses this challenge by conditioning strategy updates on both *a population of past search strategies* and *the current downstream solution population state*. Conditioning on the population descriptor $\phi(D_t)$ allows the system to interpret stagnation relative to population structure (e.g., loss of diversity, repeated parent selection, ineffective variation). Conditioning on the strategy database H provides empirical evidence about which strategies previously induced progress under similar states.

Together, these signals enable state-conditional strategy adaptation rather than fixed or periodic strategy switching.

Search strategy database. To achieve this, EvoX maintains a *search strategy database*

$$\mathcal{H} = \{(S_j, \phi_j, J_j)\}_{j=1}^{M},$$

which serves as a memory of previously deployed strategies. Each entry records a strategy S_j, a descriptor $\phi_j = \phi(\mathcal{D}_{t_j})$ summarizing the population state before and after deployment, and its observed performance J_j computed using Eq. 2. This database enables EvoX to reason about *which strategies tend to work under which search conditions*.

Population state descriptor. The descriptor $\phi(\mathcal{D}_t)$ summarizes the current state of the solution population. It includes: (i) score statistics (best value, percentiles, spread), (ii) frontier structure (e.g., top-k scores), (iii) progress indicators (e.g., steps since last significant improvement), and (iv) recent window statistics (e.g., parent selection frequency).

Meta-evolution of search strategies. When a strategy update is triggered, EvoX evolves a new search strategy by applying variation to high-performing strategies from the history of the search. The strategy database $\mathcal{H}$ plays the role of a population, where each individual strategy S_j is associated with a score signal J_j evaluated in the population state ϕ_j in which it was deployed.

EvoX performs score-biased selection over $\mathcal{H}$ to choose a high-performing parent strategy to mutate, and selects an inspirational set of strategies prioritizing those that previously induced strong progress and those that performed well under similar population descriptors. The strategy generator (i.e., an LLM), denoted $\mathcal{G}_{\text{str}}$, then applies mutation to the selected parent, conditioned on the current population descriptor $\phi(\mathcal{D}_t)$, to produce a new strategy candidate S':

$$S' \sim \mathcal{G}_{\text{str}}\big(\cdot \mid S_{\text{par}}, \phi(\mathcal{D}_t)\big)\,,$$

Mutations modify the components of a strategy (e.g., parent selection rules, construction of the inspiration set, or preferences over variation operators π).

Strategy deployment. Because strategies directly affect execution, EvoX validates each candidate strategy before deployment. If validation succeeds, EvoX immediately switches to S'; otherwise, it retries generation up to a fixed budget and falls back to the previous strategy if all attempts fail. Importantly, switching strategies never resets the solution population: the database $\mathcal{D}$ is preserved, and evolution continues from the current search state.

5. Case Study: Signal Processing

We illustrate EvoX through a case study on a signal processing task [29, 28]. The goal is to build a filtering program for a noisy, changing time series. To succeed, the program must balance several competing goals: high fidelity to the signal, smoothness to reduce noise, low lag for responsiveness, and a minimal false trend changes. We evaluate candidate programs using a combined score that balances these four objectives.

Figure **2** compares EvoX against a static baseline under the same 100-iteration budget. The static baseline uniformly samples the parent and inspiration set from the population and applies free-form variation to generate new candidates throughout the search. EvoX achieves a 34.1% higher final score by adaptively changing its search strategy based on observed progress, as described in Section 4.

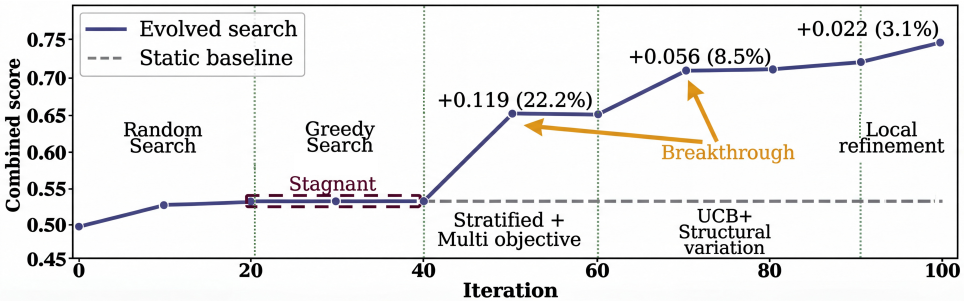


Figure 2: **Evolving search strategy on the signal processing task.** Starting from a uniform random sampling strategy, EvoX detects stagnation and adaptively switches to greedy search, stratified multi-objective sampling, UCB-guided structural variation, and finally local refinement. These strategy changes enable discovery of improved filtering programs and yield major gains at iterations ∼48 (+0.119), ∼70 (+0.056), and ∼96 (+0.022), achieving 34.1% higher final score than the static baseline.

Static Baseline. The dashed curve in Figure **2** illustrates the limitations of a fixed search strategy. This strategy achieves modest early improvement (0.499 → 0.530) but then stagnates. Because the baseline chooses its parent and inspiration programs randomly and uses generic free-form variations, it mostly produces simple filters, such as basic moving averages (MA) or exponential moving averages (EMA) using NumPy. Later generations only make tiny adjustments to these simple settings, which are not enough to handle complex noise patterns. As a result, the search hits a ceiling and stays stuck.

Phase 1: Random Search and Greedy Search. EvoX begins with the same random strategy as the baseline. At iteration 20, the system detects that progress has stopped and tries a greedy strategy that focuses entirely on refining the single best program found so far. However, because the current best program still relies on simple MA/EMA structures, these tiny refinements fail to produce a breakthrough. The search remains stuck because the underlying program structure is too limited.

Phase 2: The Breakthrough (Stratified + Multi-Objective). At iteration 40, *EvoX* learns from its past failure and switches to a more advanced strategy, where instead of only selecting the best overall program, it selects parents and inspiration programs from diverse score tiers and objective-specific rankings.

For example, if a parent filter reaches a high smoothness score but has a high lag at the same time, EvoX selects an inspiration program that excels with lower lag. Conversely, if a parent preserves signal fidelity but leaves too much noise, it picks an inspiration program specialized in smoothing of signal. By merging these different strengths, EvoX discovers novel hybrid designs such as combining singular spectrum analysis (SSA) with Whittaker smoothing. This smart blending of complementary ideas creates the largest jump in performance (+0.119).

Phase 3: Structural Exploration (UCB + Structural Variation). By iteration 60, as the progress slows again, the population state descriptor shows that many recently generated candidates have similar scores. This suggests that simple refinements or combination of ideas is no longer working. To break this pattern, EvoX evolves a policy that increases the use of structural variation to attempt bolder, more complex changes.

At the same time, it uses a UCB selection rule that encourages exploration of programs that have been largely ignored (rarely selected as parents). The strategy continues to use multi-objective sampling for selecting the inspiration programs, as it proved effective in the previous search stage. It also

employs the *structural variation operator* to encourage large, exploratory changes. As a result, the newly generated solutions begin to use advanced SciPy tools to construct filtering pipelines, incorporating higher-order filters, smoothing kernels, and forward–backward filtering operations (e.g., `filtfilt`). This exploration of new solution families yields a significant further improvement (+0.056).

Phase 4: Final Polishing (UCB + Local Refinement). By iteration 90, the search enters its final stage. Large structural changes now tend to destabilize performance rather than improve it. Consequently, EvoX shifts its strategy toward local refinement. This involves making small, precise adjustments to the top discovered solutions while keeping the UCB rule to prevent the search from narrowing too quickly. These fine-tuning steps provide the final gains (+0.022) and lock in the high score.

This example demonstrates the core power of *EvoX*: it changes its search strategy based on what happened in previous steps and the current variety of programs it has found. By choosing the right strategy for the right moment, EvoX avoids the plateaus that stall traditional methods and achieves substantially higher final performance.

6. Evaluation

Benchmarks. We evaluate EvoX on 196 real-world optimization tasks spanning mathematics (8), systems (6), and algorithmic and research problems (10 from ALE-Bench-Lite [15] and 172 from Frontier-CS [20]) (Sections 6.1–6.3). We also report results on the ARC-AGI-2 benchmark, a widely used benchmark for evaluating reasoning and generalization capabilities (Appendix B.1). Detailed descriptions on the benchmarks are in Appendix A. Additionally, we conduct ablation studies to analyze the cost and scaling behavior of EvoX in Section 6.4.

Baselines and Setups. We compare EvoX against strong LLM-driven evolutionary frameworks, including OpenEvolve [28], ShinkaEvolve [17], and GEPA [1]. For mathematical tasks, we additionally report human-best results and prior state-of-the-art AlphaEvolve [22] numbers. For ADRS systems benchmarks, we report human-best results. We include further comparisons to CodeEvolve [3] and ThetaEvolve [33] in Appendix B.

For EvoX, we use GPT-5 for search strategy generation with a window size of 10% of the total iteration budget to detect stagnation and trigger strategy evolution (Section 4). All EvoX runs start from a simple *random search strategy* that samples both the parent and inspirational candidates uniformly at random. The effect of different initial search strategies (e.g., Best-of-N, MAP-Elites) is shown in Section 6.4. We report full results of this random search strategy in Appendix B.2.

All *open* frameworks, including OpenEvolve, ShinkaEvolve, GEPA, and EvoX, are evaluated under a fixed budget of 100 iterations per task unless specified otherwise. For math tasks, human-best and AlphaEvolve [22] results are taken from the original paper, as AlphaEvolve is not open-sourced and its iteration budget is not publicly available.

We report mean and best performance over three independent runs using two backbone models, GPT-5 [23] and Gemini-3.0-Pro [12]. Mean scores reflect robustness across runs, while best scores capture peak solution quality. Higher (↑) or lower (↓) scores indicate better performance depending on the task.

Table 2: **Main results for math optimization problems.** We report mean and best over three runs. "↑" / "↓" indicate maximization / minimization. All open methods use 100 iterations under the same backbone model. **Bold** denotes the best LLM-based open framework result per model. Green cells indicate matches or improvements over human SOTA or closed-source AlphaEvolve results. In Circle Packing Rect, EvoX achieves 2.36583237, exceeding AlphaEvolve's 2.36583213, although both appear equal when rounded.

				GPT-5				Gemini-3.0-Pro			
Task	**Stat**	**Human**	**AlphaEvolve**	**OpenEvolve**	**GEPA**	**Shinka**	**EvoX**	**OpenEvolve**	**GEPA**	**Shinka**	**EvoX**
Circle Packing (↑)	Mean	–	–	2.5308	2.6129	2.4642	**2.6324**	2.5414	2.6198	2.6216	**2.6328**
	Best	2.6340	2.6350	2.5414	2.6285	2.5161	**2.6359**	2.5414	2.6208	2.6358	**2.6359**
Circle Packing Rect (↑)	Mean	–	–	2.2673	2.3265	2.3347	**2.3525**	2.3651	2.2160	**2.3658**	**2.3658**
	Best	2.3640	2.3658	2.2756	2.3537	2.3577	**2.3600**	2.3658	2.2507	**2.3658**	**2.3658**
heilbronn_convex (↑)	Mean	–	–	0.0230	0.0259	0.0228	**0.0263**	0.0279	0.0228	0.0279	**0.0287**
	Best	0.0306	0.0309	0.0267	0.0269	0.0256	**0.0272**	0.0280	0.0274	0.0287	**0.0296**
heilbronn_triangle (↑)	Mean	–	–	0.0250	0.0317	**0.0319**	0.0316	0.0331	0.0311	0.0351	**0.0359**
	Best	0.0360	0.0365	0.0283	0.0332	0.0338	**0.0339**	0.0350	0.0329	0.0356	**0.0365**
min_max_min_dist (n=16, d=2) (↓)	Mean	–	–	13.00	12.98	12.99	**12.95**	12.96	12.89	12.89	**12.89**
	Best	12.89	12.89	13.00	12.95	12.98	**12.89**	**12.89**	**12.89**	**12.89**	**12.89**
min_max_min_dist (n=14, d=3) (↓)	Mean	–	–	4.51	4.30	**4.19**	4.21	4.17	4.71	**4.16**	4.17
	Best	4.17	4.17	4.46	4.18	**4.17**	4.18	**4.16**	4.59	**4.16**	**4.16**
third_autocorr_ineq (↓)	Mean	–	–	1.4671	1.4768	1.4785	**1.4714**	1.4609	1.4678	1.4595	**1.4589**
	Best	1.4581	1.4557	1.4610	1.4758	1.4614	**1.4609**	1.4600	1.4598	1.4578	**1.4558**
signal processing (↑)	Mean	–	–	0.5686	0.6891	0.4855	**0.7106**	0.5525	0.6169	0.4799	**0.7351**
	Best	–	–	0.6219	0.7057	0.5328	**0.7214**	0.5649	0.6798	0.5049	**0.7429**

6.1. Main Results: Math Optimization Problems

We evaluate eight mathematical optimization tasks (continuous and discrete) spanning geometric packing, extremal geometry, distance maximization, and sequence design (Table **2**). Across these tasks, EvoX achieves the strongest overall performance among open evolutionary frameworks, including OpenEvolve, ShinkaEvolve, and GEPA.

Under GPT-5, EvoX attains the *best* or tied-best result on 7 of 8 tasks, and under Gemini-3.0-Pro, it achieves the best result on all 8 tasks. In terms of *mean* performance, EvoX achieves the best score on 6 of 8 tasks under GPT-5 and 7 of 8 tasks under Gemini-3.0-Pro, demonstrating robust performance across runs. In the two cases where mean performance lags best performance (Heilbronn triangle and MinMaxMinDist $d = 3$), early random initializations occasionally converge to strong local optima, limiting further improvement within the fixed 100 iteration budget we set.

Compared to the strongest previously reported results, including the closed-source AlphaEvolve, EvoX matches or exceeds AlphaEvolve on 5 out of 7 tasks, such as Circle Packing, Circle Packing Rect, and MinMaxMinDist ($d = 3$), within just 100 iterations. Since AlphaEvolve's iteration budget is not publicly specified, direct cost comparison is not possible.

Beyond aggregate scores, EvoX discovers qualitatively distinct solutions across domains. In circle packing, EvoX achieves state-of-the-art performance by constructing a hexagonal-lattice core packing

Table 3: **Main results for system problems.** We report mean and best scores over three runs. "↑" denotes maximization and "↓" minimization. All methods (except human / prior AI) are run for 100 iterations using GPT-5 and Gemini-3.0-Pro. **Bold** denotes the best LLM-based open framework result per model. Green cells indicate matches or improvements over human SOTA results.

			GPT-5				Gemini-3.0-Pro			
Task	**Stat**	**Human**	**OpenEvolve**	**GEPA**	**Shinka**	**EvoX**	**OpenEvolve**	**GEPA**	**Shinka**	**EvoX**
EPLB (↑)	Mean	–	0.1300	0.1338	0.1185	**0.1358**	0.1272	0.1268	0.1206	**0.1392**
	Best	0.1265	0.1272	0.1445	0.1272	**0.1453**	0.1272	0.1272	0.1272	**0.1453**
PRISM (↑)	Mean	–	25.15	26.19	26.26	**27.67**	26.24	26.16	26.25	**26.26**
	Best	21.89	26.23	26.23	26.26	**30.52**	26.24	26.19	26.26	**26.26**
LLM-SQL (↑)	Mean	–	0.7053	0.7127	0.7123	**0.7231**	0.7251	0.7129	0.7210	**0.7278**
	Best	0.6920	0.7155	0.7129	0.7125	**0.7298**	0.7258	0.7134	0.7212	**0.7300**
Cloudcast (↓)	Mean	–	851.72	689.89	954.79	**662.26**	707.82	720.38	949.79	**623.69**
	Best	626.24	729.80	645.72	812.74	**637.14**	707.82	667.06	1032.42	**623.69**
Transaction (↑)	Mean	–	3860.1	3752.6	4090	**4292.32**	4109.19	3615.57	3931.67	**4267.75**
	Best	2724.8	4237.3	3984.1	4329	**4347.83**	4273.50	3615.57	4255.32	**4310.34**
Telemetry Repair (↑)	Mean	–	0.9031	0.9158	0.9229	**0.9464**	**0.9541**	0.8505	0.9176	0.9381
	Best	0.8222	0.9515	0.9477	0.9515	**0.9520**	**0.9541**	0.8553	0.9331	**0.9467**

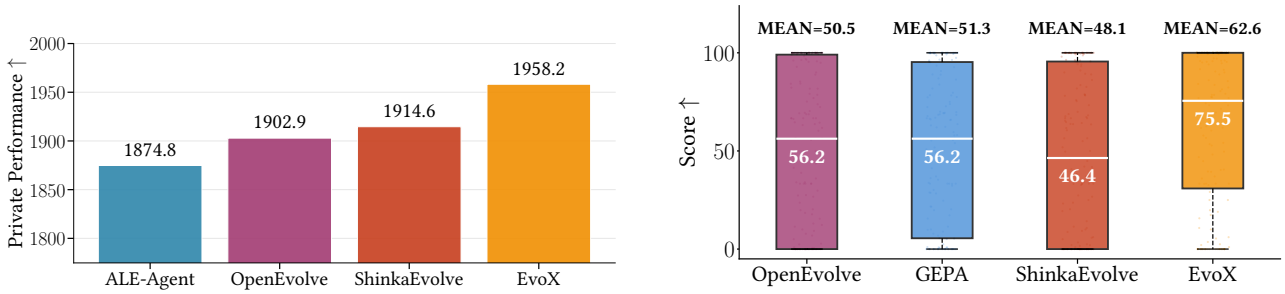
and refining it via constrained SLSQP optimization. In MinMaxMinDist ($d = 3$), EvoX discovers a 14-point configuration in $\mathbb{R}^3$ by solving a constrained extremal-distance problem initialized from structured polyhedral seeds. For the third autocorrelation inequality problem, EvoX constructs a 1024-bin discretized function over the interval $[-0.25, 0.25]$, computes its autoconvolution via FFT, and applies gradient-based optimization to minimize the peak magnitude. These solutions outperform those discovered by existing evolutionary systems with fixed, manually designed search strategies.

6.2. Main Results: System Performance Problems

We also evaluate EvoX on six real-world system optimization tasks spanning expert placement load balancing (EPLB), GPU sharing (PRISM), LLM-driven analytics (LLM-SQL), multi-cloud broadcast optimization (Cloudcast), transaction scheduling, and telemetry repair (Table **3**). EvoX exceeds human-best results on all six benchmarks under Gemini-3.0-Pro.

For mean performance, EvoX achieves the best score on all tasks under GPT-5 and 5 of 6 tasks under Gemini-3.0-Pro. Telemetry repair is the only case where OpenEvolve achieves a slightly higher mean. For best performance, EvoX outperforms or ties all baselines on all tasks under GPT-5 and 5 of 6 tasks under Gemini-3.0-Pro.

Qualitatively, EvoX uncovers new system optimization algorithms. For example, in Cloudcast, EvoX discovers a Steiner-tree–based multicast routing strategy that builds a shortest-path distance graph over regions and jointly optimizes all destinations, reducing transfer cost beyond prior heuristics. In PRISM, EvoX identifies a model-placement strategy that minimizes peak KV-cache pressure by binary-searching a global load threshold and applying Best-Fit-Decreasing packing.



(a) Average performance across 10 ALE-Bench-Lite tasks.

(b) Performance on 172 Frontier-CS tasks.

Figure 3: **Algorithm and Research Challenges.** EvoX achieves the highest average private performance with GPT-5 on 10 different ALE-Bench-Lite problems and highest median score across 172 different Frontier-CS challenges.

6.3. Main Results: Algorithmic and Research Problems

We also evaluate EvoX on algorithmic and research benchmarks (Figure **3**).

ALE-Bench-Lite. As shown in Figure **3**a, we evaluate EvoX on 10 tasks from ALE-Bench-Lite [15], derived from AtCoder Heuristic Contests. Following prior work [17], all methods are initialized from the ALE-Agent baseline and evaluated using private scores.

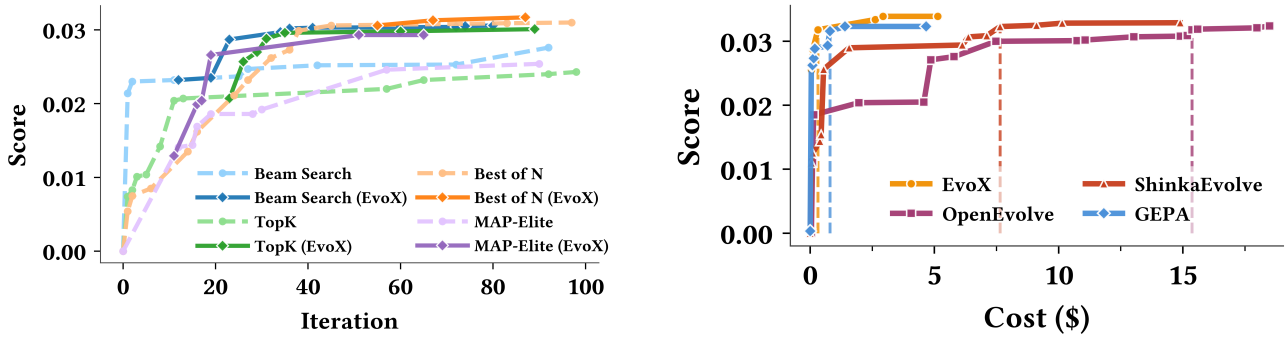
EvoX achieves the strongest overall performance, attaining the highest average private score (1958.2), outperforming ALE-Agent (1874.8), OpenEvolve (1902.9), and our reproduction of ShinkaEvolve result (1914.6) [17]. On AHC016 (graph classification), EvoX finds a solution that replaces graph edit distance with block-pattern signatures, a noise-aware representation enabling efficient matching. On AHC024 (map compression), it improves simulated annealing with boundary-aware tracking, prioritizing boundary updates to improve search progress. Gains from EvoX are smaller on some tasks (e.g., AHC025), where a strong initial solution might bias the search toward a local optimum.

Frontier-CS. We evaluate EvoX on 172 tasks from Frontier-CS [20], a large-scale benchmark of open-ended computer science problems. Each task is scored from 0–100, where 100 corresponds to the best-known human or optimal solution. As shown in Figure **3**b, EvoX achieves the strongest overall performance, reaching a mean score of 62.6 and median of 75.5. This improves over OpenEvolve by 24% in mean and 34% in median score, over GEPA by 22% and 34%, and over ShinkaEvolve by 30% and 63%, respectively. These gains demonstrate that EvoX discovers higher-quality solutions while maintaining stronger consistency across diverse competitive programming tasks.

6.4. Ablations

Starting from Different Search Strategies. Figure **4**a evaluates EvoX initialized from different fixed search strategies (Beam Search, Best-of-N, Top-K, MAP-Elites) on the Heilbronn triangle task. The Heilbronn triangle task presents a highly non-convex objective landscape, where optimization behavior is sensitive to both global configuration discovery and subsequent local improvements.

Beam search and Best-of-N provide the strongest initializations by concentrating compute on repeatedly refining top candidates: beam search via elite expansion and pruning, and Best-of-N through



(a) **Effect of different initial search strategies**

(b) **Cost-quality tradeoff.**

Figure 4: **Search strategy evolution and cost-quality tradeoffs on the Heilbronn triangle task. (a)** Dashed lines show fixed strategies, while solid lines show EvoX initialized from each strategy and allowed to evolve. Regardless of initialization, EvoX continues improving beyond the fixed strategy. **(b)** Cost-quality tradeoff under the GPT-5 model. To exceed a score of 0.031, EvoX and GEPA both require less than $1 in LLM generation cost, compared to ShinkaEvolve ($7.6) and OpenEvolve ($15.4).

large-batch selection. Top-K instead preserves multiple candidates, diffusing optimization pressure and slowing convergence, while MAP-Elites maintains a grid of diverse solutions and lags behind.

Regardless of initialization, fixed strategies exhibit early saturation. In contrast, EvoX consistently improves solution quality by adapting the search strategy during optimization. This behavior indicates robustness to initialization and demonstrates the benefits of strategy evolution.

Cost and Scaling. Figure 4b shows the cost–quality tradeoff on the Heilbronn triangle task under the GPT-5 model. To exceed a score of 0.031, EvoX and GEPA both require less than $1 in LLM generation cost, compared to $7.6 for ShinkaEvolve and $15.4 for OpenEvolve. However, GEPA plateaus at a score of 0.0323 after around 20 iterations and shows no further improvement despite continued search. In contrast, EvoX breaks through this stagnation point and reaches a peak score of 0.0339, which is the highest among all methods, by adapting its search strategy during optimization. This shows that EvoX not only reaches competitive solutions faster, but continues to improve where fixed-strategy baselines stall.

Search Evolution Examples. We provide additional case studies of search strategy evolution in Appendix D. In some tasks, strong generator models already produce high-quality candidates, allowing even simple strategies to make progress. For example, under Gemini-3.0-Pro, random sampling alone yields substantial improvements on Cloudcast (Appendix Table A6), without specialized selection or variation.

In contrast, other tasks rely heavily on adaptive strategy. For Signal Processing, major improvements arise during multi-objective sampling, followed by refinement-focused phases. For Circle Packing, early free-form variation yields large gains but quickly saturates; continued progress emerges through structural variation, with later refinement contributing incremental improvements. The Heilbronn Triangle task derives its improvements predominantly from strategies emphasizing local refinement, on top of a reasonable initial configuration identified in the early iterations.

Appendix Tables A7, A8, and A9 provide detailed breakdowns of these patterns. Across tasks, both

selection mechanisms and variation operators evolve in task- and run-dependent ways, reflecting differences in optimization dynamics.

7. Conclusion

We introduced EvoX, a meta-evolution method that jointly evolves candidate solutions and the search strategies that generate them. By adapting search strategy to the structure and stage of the optimization process, EvoX delivers consistent improvements in solution quality and cost efficiency across diverse domains. As optimization problems continue to scale in size and complexity, these results point toward a future of general-purpose, self-evolving systems that continuously refine how they search, reducing reliance on fixed, manually designed procedures.

Acknowledgments

This research is supported by NSF (IFML) CCF-2019844 and gifts from Accenture, AMD, Anyscale, Broadcom Inc., Google, IBM, Intel, Intesa Sanpaolo, Lambda, Mibura Inc, Samsung SDS, and SAP.

References

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, pages 3981–3989, 2016.

[3] Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. Codeevolve: An open source evolutionary coding agent for algorithm discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025.

[4] Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jinsong Su, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. Flex: Continuous agent evolution via forward learning from experience. *arXiv preprint arXiv:2511.06449*, 2025.

[5] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[6] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

[7] Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, et al. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

[8] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

[9] Harshita Chopra and Chirag Shah. Feedback-aware monte carlo tree search for efficient information seeking in goal-oriented conversations. *arXiv preprint arXiv:2501.15056*, 2025.

[10] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

[11] Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

[12] Google DeepMind. Gemini 3 Pro model card. Technical report, Google DeepMind, December 2025.

[13] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv e-prints*, pages arXiv–2309, 2023.

[14] Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. Evolving code with a large language model. *Genetic Programming and Evolvable Machines*, 25(2):21, 2024.

[15] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. *arXiv preprint arXiv:2506.09050*, 2025.

[16] Jiachen Jiang, Tianyu Ding, and Zhihui Zhu. Deltaevolve: Accelerating scientific discovery through momentum-driven evolution. *arXiv preprint arXiv:2602.02919*, 2026.

[17] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution, 2025.

[18] Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*, 2025.

[19] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[20] Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, et al. Frontiercs: Evolving challenges for evolving intelligence. *arXiv preprint arXiv:2512.15699*, 2025.

[21] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, volume 97, pages 4556–4565. PMLR, 2019.

[22] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[23] OpenAI. GPT-5 system card. Technical report, OpenAI, August 2025.

[24] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

[25] Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on arc-agi. *arXiv preprint arXiv:2507.14172*, 2025.

[26] Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning, 2025.

[27] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[28] Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent. GitHub, 2025.

[29] Belle A Shenoi. *Introduction to digital signal processing and filter design*, volume 169. John Wiley & Sons, 2005.

[30] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[31] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

[32] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

[33] Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. ThetaEvolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

[34] Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, Chi Wang, et al. Pacevolve: Enabling long-horizon progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026.

[35] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

[36] Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

[37] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Appendix

A. Benchmark Details

We evaluate EvoX on 196 problems: 24 optimization problems spanning three domains (mathematical optimization (8 problems), system performance (6 problems), and algorithmic challenges (10 problems)) and 172 algorithmic problems from FrontierCS.

A.1. Mathematical Optimization

Table A1: **Mathematical optimization benchmarks.** Summary of optimization tasks used to evaluate EvoX, including geometric, combinatorial, and signal processing problems.

Problem	**Objective**	**Description**
Circle Packing (Square)	max r	Place n equal-radius circles inside the unit square without overlap; maximize the common radius subject to non-overlap and boundary constraints.
Circle Packing (Rectangle)	max r	Pack n equal-radius circles into a rectangle with fixed aspect ratio, maximizing the radius under geometric constraints.
Heilbronn Triangle	max min area	Place n points in $[0,1]^2$ to maximize the area of the smallest triangle formed by any three points.
Heilbronn Convex	max min area	Generalization of the Heilbronn triangle problem: maximize the minimum area of convex hulls formed by any subset of $k > 3$ points.
Min–Max–Min Distance (2 variants)	max $\frac{d_{\min}}{d_{\max}}$	Place points to maximize uniformity, measured by the ratio between minimum and maximum pairwise distances.
Third Autocorrelation Inequality	min C_3	Construct witness functions that minimize the third autocorrelation constant in additive combinatorics.
Signal Processing	multi-objective	Design a causal, online filtering program for noisy time series, balancing fidelity, smoothness, lag, and false trend detection.

A.2. System Performance Optimization

Table A2: **System performance optimization benchmarks.** Benchmarks drawn from ADRS-Bench [7], capturing realistic optimization problems from production systems.

Problem	Objective	Description
EPLB (Expert Placement Load Balancing)	max throughput ratio	Place and replicate experts in MoE models across GPUs based on activation frequency, minimizing load imbalance and computational skew.
PRISM (Global Model Scheduling)	max throughput	Schedule multiple deep learning models across distributed GPUs under memory, compute, and latency SLO constraints in a multi-tenant setting.
LLM–SQL	max accuracy	Optimize LLM-driven SQL generation via prompting or post-processing; a query is correct only if it exactly matches ground-truth execution results.
Cloudcast (Multi-Region Data Transfer)	min cost	Design cost-efficient data transfer strategies across cloud regions with heterogeneous bandwidths, latencies, and egress pricing under deadline constraints.
Transaction Scheduling	max commits/sec	Schedule database transactions under contention to maximize throughput while minimizing aborts from conflicts, deadlocks, or timeouts.
Telemetry Repair	max reconstruction accuracy	Reconstruct missing or corrupted telemetry signals using partial observations, exploiting temporal and spatial correlations.

A.3. Algorithmic and Research Problems

Table A3: **Algorithmic and research problem benchmarks.** We evaluate on 182 open-ended problems: 10 NP-hard optimization problems from ALE-Bench-Lite [15] and 172 problems from FrontierCS [20].

Problem / Category	Count	Description
ALE-Bench-Lite: NP-hard optimization from AtCoder Heuristic Contests		
`ahc008` (Territory)	1	Control agents on a grid to place fences isolating pets; max satisfaction.
`ahc011` (Sliding Puzzle)	1	Rearrange tiles so line patterns form a large connected tree; max connectivity.
`ahc015` (Candy Clustering)	1	Cluster candies of the same flavor using global tilt operations; max clustering score.
`ahc016` (Graph Classification)	1	Design reference graphs and classify noisy, permuted query graphs; min classification error.
`ahc024` (Map Compression)	1	Compress a grid map preserving district adjacency; min map size.
`ahc025` (Weight Balancing)	1	Partition items with unknown weights into balanced groups; min imbalance.
`ahc026` (Box Stacking)	1	Rearrange numbered boxes across stacks to a target configuration; min moves.
`ahc027` (Cleaning Route)	1	Design a cyclic robot route to minimize steady-state dirt; min average dirtiness.
`ahc039` (Fishing Net)	1	Construct a rectilinear polygon enclosing targets under a perimeter budget; max net score.
`ahc046` (Skating with Blocks)	1	Visit target squares in order using moves, slides, and blocks; min actions.
FrontierCS: open-ended CS problems		
`Optimization`	38	Maximize or minimize a quantitative objective over a parameterized search space under resource constraints. IDs: 1, 2, 15, 16, 22, 25, 26, 27, 28, 30, 33, 35, 36, 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 52, 53, 54, 57, 58, 59, 61, 64, 68, 69, 79, 81, 86, 93, 229.
`Constructive`	62	Synthesize a valid structured object (e.g., packing, graph, expression) under global constraints. IDs: 0, 3, 4, 5, 6, 7, 8, 9, 13, 17, 23, 24, 60, 62, 63, 70, 72, 73, 75, 77, 80, 82, 83, 85, 87, 89, 142, 174–193, 192, 193, 203, 205, 207, 209–214, 217, 220, 222, 225, 227, 228, 239, 241.
`Interactive`	72	Solve a hidden-instance task via an adaptive query-response protocol, minimizing queries or interaction steps. IDs: 10, 11, 14, 101, 104, 106–113, 117, 119–125, 127, 132–135, 137, 138, 140, 141, 143–145, 147–171, 226, 231, 233, 243, 245, 247–249, 252–258.

B. Additional Results

We report additional evaluations to examine EvoX behavior beyond the primary optimization benchmarks discussed in the main paper.

B.1. ARC-AGI Evaluation

ARC-AGI-2 Tasks [8] evaluate abstract and compositional reasoning across programmatic problem-solving instances. Although ARC-AGI-2 is not explicitly designed as an optimization benchmark, it provides a useful testbed for analyzing cross-domain robustness.

Experiments follow the evaluation protocol described in Section 4.1. OpenEvolve (OE) and EvoX operate under a matched inference budget (30 LLM iterations) per task.

Table A4: EvoX performance on ARC-AGI benchmarks. Values denote final accuracy.

Backbone	OpenEvolve	EvoX
GPT-5	41%	48%
Gemini-3-Pro	45%	51%

These results indicate that EvoX maintains performance improvements even on reasoning-oriented tasks outside traditional optimization settings. We emphasize that ARC-AGI fundamentally differs from the evolutionary optimization regime, as standard ARC evaluation assumes strict train–test separation, whereas evolutionary frameworks typically adapt during inference.

B.2. Full Results

We additionally compare against CodeEvolve [3] and ThetaEvolve [33], two recent systems representing strong prior approaches for LLM-driven optimization.

Across tasks, EvoX maintains consistent performance advantages over both fixed-policy and adaptive-policy baselines. Detailed per-task breakdowns are provided below.

Table A5: Main results for math optimization problems. We report mean and best over three runs. "↑" / "↓" indicate maximization / minimization.

						GPT-5					Gemini-3.0-Pro				
Task	Stat	Human	AlphaEvolve	ThetaEvolve	CodeEvolve	Random	OpenEvolve	GEPA	Shinka	EvoX	Random	OpenEvolve	GEPA	Shinka	EvoX
Circle Packing (↑)	Mean	–	–	–	–	2.4905	2.5308	2.6129	2.4642	2.6324	2.5564	2.5414	2.6198	2.6216	2.6328
	Best	2.6340	2.6350	2.6360	2.6360	2.5000	2.5414	2.6285	2.5078	2.6360	2.5853	2.5414	2.6208	2.6358	2.6360
Circle Packing Rect (↑)	Mean	–	–	–	–	2.2105	2.2673	2.3265	2.3347	2.3525	2.3432	2.3651	2.2160	2.3658	2.3658
	Best	2.3640	2.3658	–	2.3658	2.2783	2.2756	2.3537	2.3577	2.3600	2.3652	2.3658	2.2507	2.3658	2.3658
heilbronn_convex (↑)	Mean	–	–	–	–	0.0185	0.0230	0.0259	0.0228	0.0263	0.0285	0.0279	0.0228	0.0279	0.0287
	Best	0.0306	0.0309	–	–	0.0198	0.0267	0.0269	0.0256	0.0272	0.0298	0.0280	0.0274	0.0287	0.0296
heilbronn_triangle (↑)	Mean	–	–	–	–	0.0261	0.0250	0.0317	0.0319	0.0316	0.0318	0.0331	0.0311	0.0351	0.0359
	Best	0.0360	0.0365	–	–	0.0285	0.0283	0.0332	0.0338	0.0339	0.0329	0.0350	0.0329	0.0356	0.0365
min_max_min_dist (n=16, d=2) (↓)	Mean	–	–	–	–	13.00	13.00	12.98	12.99	12.95	12.92	12.96	12.89	12.89	12.89
	Best	12.89	12.89	–	12.89	13.00	13.00	12.95	12.98	12.89	12.89	12.89	12.89	12.89	12.89
min_max_min_dist (n=14, d=3) (↓)	Mean	–	–	–	–	4.47	4.51	4.19	4.19	4.21	4.19	4.17	4.71	4.16	4.17
	Best	4.17	4.17	–	4.17	4.46	4.46	4.17	4.17	4.18	4.16	4.16	4.59	4.16	4.16
third_autocorr_ineq (↓)	Mean	–	–	–	–	1.4820	1.4671	1.4768	1.4785	1.4714	1.4594	1.4609	1.4678	1.4595	1.4589
	Best	1.4581	1.4557	1.4930	–	1.4607	1.4610	1.4758	1.4614	1.4609	1.4561	1.4600	1.4598	1.4578	1.4558
signal processing (↑)	Mean	–	–	–	–	0.5484	0.5686	0.6891	0.4855	0.6580	0.5527	0.5525	0.6169	0.4799	0.7351
	Best	–	–	–	–	0.5858	0.6219	0.7057	0.5328	0.6898	0.5647	0.5649	0.6798	0.5049	0.7429

Table A6: Main results for system problems. We report mean and best scores over three runs. "↑" denotes maximization and "↓" minimization.

			GPT-5					Gemini-3.0-Pro				
Task	Stat	Human	Random	OpenEvolve	GEPA	Shinka	EvoX	Random	OpenEvolve	GEPA	Shinka	EvoX
EPLB (↑)	Mean	–	0.1338	0.1300	0.1338	0.1185	0.1358	0.1280	0.1272	0.1268	0.1206	0.1392
	Best	0.1265	0.1445	0.1272	0.1445	0.1272	0.1453	0.1285	0.1272	0.1272	0.1272	0.1453
PRISM (↑)	Mean	–	26.23	25.15	26.19	26.26	27.67	26.23	26.24	26.16	26.25	26.26
	Best	21.89	26.23	26.23	26.23	26.26	30.52	26.23	26.24	26.19	26.26	26.26
LLM-SQL (↑)	Mean	–	0.7201	0.7053	0.7127	0.7123	0.7231	0.7380	0.7251	0.7129	0.7210	0.7278
	Best	0.6920	0.7289	0.7155	0.7129	0.7125	0.7298	0.7440	0.7258	0.7134	0.7212	0.7300
Cloudcast (↓)	Mean	–	659.00	851.72	689.89	954.79	662.26	629.84	707.82	720.38	949.79	623.69
	Best	626.24	644.84	729.80	645.72	812.74	637.14	635.72	707.82	667.06	1032.42	623.69
Transaction (↑)	Mean	–	4123.87	3860.10	3752.6	4090.0	4292.32	4037.14	4109.19	3615.57	3931.67	4267.75
	Best	2724.8	4149.38	4237.3	3984.1	4329.0	4347.83	4237.29	4273.50	3615.57	4255.32	4310.34
Telemetry Repair (↑)	Mean	–	0.8959	0.9031	0.9158	0.9229	0.9464	0.9179	0.9541	0.8505	0.9176	0.9381
	Best	0.8222	0.9498	0.9515	0.9477	0.9515	0.9520	0.9515	0.9541	0.8553	0.9331	0.9467

C. Prompt

Below is the prompt used for search strategy evolution, together with example mutation operator labels.

System Model: LLM-Driven Optimization with Evolving Search

You are an expert developer implementing a search strategy for LLM-driven program optimization. Your task is to implement `EvolvedProgramDatabase`, which realizes the `SearchStrategy` interface used by the optimizer.

The optimizer maintains a population database of evaluated candidate programs. At each step, the search strategy constructs the next-generation LLM input by choosing: (i) a parent program to modify, (ii) a variation operator that specifies how the parent should be modified, and (iii) an optional inspiration set of past programs drawn from the population. The generated candidate is then evaluated and added back to the population.

The quality of a search strategy is determined by how effectively it improves the best solution over the course of the optimization process.

You must implement a class named `EvolvedProgramDatabase`, which defines the behavior of the search strategy through two methods:

- `add`, which controls how evaluated programs are added to the population
- `sample`, which selects the parent program, the variation operator applied to it, and an optional inspiration set of past programs.

The base class provides `self.programs: Dict[str, EvolvedProgram]` (all stored programs), `self.get(program_id)` for retrieval, and predefined label constants `self.DIVERGE_LABEL` and `self.REFINE_LABEL` (must not be modified).

Each candidate program is represented as an `EvolvedProgram`. Its fields include `id`, `code`, `metrics` (with main score `combined_score`), `iteration_found`, `timestamp`, `parent_id`, `parent_info` (a `(label, id)` tuple tracking the label applied to the parent), and `metadata`. These fields may be read but must not be modified.

The `sample` method returns a tuple of two dicts:

- `parent_dict: Dict[str, EvolvedProgram]` — maps a label to exactly one parent program;
- `inspiration_set_dict: List[EvolvedProgram]` — lists of inspirational programs providing complementary context.

Label rules. The parent label should be the empty string `""` by default. `self.REFINE_LABEL` (local refinement) or `self.DIVERGE_LABEL` (structural variation) can also be used as the parent label according to the search states.

Each generation step can only apply exactly one variation operator:

- free-form variation (the default, label `""`),

- local refinement (`REFINE_LABEL`), which requests small, structure-preserving edits,
- structural variation (`DIVERGE_LABEL`), which encourages larger, exploratory changes.

```
=======================
CONSTRAINTS
=======================
```

1. Program evaluation metrics are read-only and must not be modified.
2. All imports and dataclass definitions must be preserved.
3. All variables used by the strategy must be initialized with safe default values to avoid runtime errors.
4. When accessing metrics, always validate types before arithmetic (`isinstance(value, (int, float))`).

```
=======================
CRITICAL INSTRUCTION
=======================
```

Design a search strategy wisely and adaptively to maximize improvement.

Search evolution — end-to-end prompt (P_{search})

System message

> You are an expert software developer tasked with iteratively improving a codebase. Your objective is to maximize the **combined score** while exploring diverse solutions across feature dimensions. The system maintains a population of programs; both high performance and diversity are valuable signals.

User message

Downstream problem context. Your search algorithm evolves solutions for the following downstream problem. Use this information to inform your search strategy.

> {problem_template}

Search algorithm information. **Search window.** Defines the iteration range, optimization horizon, and any adaptive replacement rules governing the lifetime of the search algorithm.

> {search_window_context}

Solution population statistics. Summarizes the current state of the solution database, including population size, score distribution, stagnation signals, reuse patterns, and execution traces.

> {population_state}

Current search algorithm. The program to be rewritten.

{current_program}

Relevant prior attempts. Previous search-algorithm variants that may provide useful design signals.

{relevant_programs}

Task. Rewrite the program to improve the search-algorithm score given the population state and downstream objective.
Provide the complete rewritten program and briefly explain one or two high-level principles guiding your redesign.
Keep the implementation simple, and preserve the original program's inputs and outputs while improving its internal behavior.

```
# Your rewritten search algorithm here
```

D. Analysis of Search Evolution

Table A7: **Search strategy evolution on Signal Processing (Gemini, 100 iterations).** Initial score 0.499 $\rightarrow$ final 0.743. Δ: improvement achieved within the phase window; W: window length. Variation operator $\pi \in \{\textit{local refinement}, \textit{structural variation}, \textit{free-form variation}\}$, where free-form variation is the default.

Task	Strategy	Search Mechanism	Variation Operator	Δ	W
Signal Processing	Random sampling	Parents and inspiration programs sampled uniformly from the population	Free-form variation	+0.03	11
	Greedy sampling	Parents sampled from top-performing candidates	Free-form variation	+0.01	10
	Stratified + multi-objective sampling	Parents selected across objective-specific score rankings	Structural variation	+0.12	19
	UCB-based selection	Usage-based parent selection with penalty for overuse	Structural variation	+0.06	16
	Top-tier refinement	Parents restricted to high-score candidates	Local refinement	+0.03	14

Table A8: **Search strategy evolution on Circle Packing (Gemini, 100 iterations).** Initial score 0.364 $\rightarrow$ final 1.0004. Δ: improvement achieved within the phase window; W: window length. Variation operator $\pi \in \{\textit{local refinement}, \textit{structural variation}, \textit{free-form variation}\}$, where free-form variation is the default.

Task	Strategy	Search Mechanism	Operator Bias	Δ	W
Circle Packing	Random sampling	Parents and contexts sampled uniformly from the population	Free-form variation	+0.59	21
	Usage-penalized sampling	Parent selection penalizing recently sampled candidates	Structural variation	+0.04	17
	Tiered sampling	Parents selected from elite, mid, and lower score bands	Mixed (refinement + structural)	+0.004	10
	Stagnation-aware sampling	Parent selection conditioned on recent parent-child scores	Structural variation	+0.003	10
	Elite refinement	Parents restricted to high-score candidates	Local refinement	0	20
	Quantile-biased refinement	Parents sampled from upper quantiles of the score distribution	Local refinement	$\sim$2e$-$5	10

Table A9: **Search strategy evolution on Heilbronn Triangle (Gemini, 100 iterations).** Initial score 0.0 $\rightarrow$ final `min_area` $\approx$ 0.0365 (SOTA 1.0); best at iteration 91. Δ: improvement achieved within the phase window; W: window length. Variation operator $\pi \in \{\textit{local refinement}, \textit{structural variation}, \textit{free-form variation}\}$, where free-form variation is the default.

Task	Strategy	Search Mechanism	Operator Bias	Δ	W
Heilbronn Triangle	Stratified sampling	Parents selected from top and mid score bands	Free-form variation	+0.853	26
	Exploration-biased sampling	Parents sampled uniformly with occasional high-score selection	Structural variation	0	10
	Usage-penalized sampling	Parent selection penalizing recently sampled candidates	Mixed (refinement + structural)	+0.026	13
	Tiered refinement	Parents selected from top and mid score bands	Local refinement	+0.100	16
	Visit-weighted sampling	Parent selection weighted by historical sampling frequency	Free-form variation	0	10
	Refinement-focused sampling	Parents sampled from high-score candidates	Local refinement	+0.021	10

D.1. Case Study: Search Evolution in Circle Packing

We now illustrate EvoX through a case study on the **Circle Packing** task, where the objective is to construct a program that maximizes packing efficiency under geometric constraints. The generated program must balance multiple competing factors, including achieving dense configurations, maintaining numerical stability, preventing overlap violations, and converging reliably within the evaluation budget. Candidate programs are evaluated using a normalized packing score reflecting achieved density under constraint satisfaction.

Starting from an initial score of 0.364, EvoX reaches a near-optimal score of 1.0004. Table A8 summarizes the sequence of evolved strategies.

Static Baseline Behavior. Uniform parent sampling combined with free-form variation produces rapid early gains (+0.59), primarily by discovering coarse geometric heuristics such as greedy placement rules, collision-aware perturbations, and simple local displacement schemes. However, progress quickly saturates. Free-form variation largely generates parameter-level modifications, adjusting perturbation radii, iteration counts, or threshold values without introducing qualitatively new optimization mechanisms. Because uniform sampling increasingly selects structurally similar programs, variation operates over a narrowing region of the program space.

As a result, improvements exhibit diminishing returns. Local heuristic updates frequently induce oscillatory adjustments, repeated near-collisions, and unstable refinements near dense configurations. Constraint satisfaction remains externally enforced through penalties or rejection rules rather than intrinsically modeled by the update mechanism. Without adaptive strategy evolution, the baseline cannot redirect search toward mechanisms capable of coordinated global adjustments.

Evolving Search Strategy in EvoX.

Exploration-Dominated Phase. EvoX initially mirrors baseline behavior, rapidly identifying stable geometric constructions (+0.59). At this stage, improvements arise primarily from discovering viable placement schemes rather than refining precise optimization dynamics.

Diversity-Inducing Strategies. As structural redundancy emerges within the population, EvoX evolves usage-penalized and tiered sampling strategies. These mechanisms alter search dynamics by discouraging repeated selection of recently sampled parents and explicitly sampling candidates across score bands. This transition prevents over-exploitation of early high-performing but structurally limited heuristics while increasing exposure to partially successful yet structurally diverse programs. Although immediate gains are modest (+0.04, +0.004), these phases preserve population diversity, which proves critical for subsequent mechanism discovery.

Mechanism Discovery via Structural Variation. Under stagnation-aware sampling, structural variation produces programs that introduce a fundamentally different optimization paradigm based on constrained numerical optimization using SLSQP. This transition represents a qualitative shift in solution construction. Earlier heuristic programs perform sequential local adjustments, modifying circle positions independently through handcrafted displacement rules. Such updates are inherently myopic and frequently induce constraint violations or oscillatory corrections. In contrast, SLSQP-based programs formulate packing as a constrained optimization problem in which circle positions become jointly optimized variables and overlap constraints are explicitly encoded.

This paradigm-level shift fundamentally alters refinement dynamics. Constraint satisfaction becomes intrinsic to the update rule rather than externally enforced, coordinated gradient-based updates mitigate oscillatory behaviors, and the optimizer implicitly captures higher-order interactions among circles. Rather than relying on independent local perturbations, updates now reflect globally coordinated corrections across multiple variables. This mechanism discovery effectively breaks prior stagnation patterns and unlocks further improvements.

Refinement-Dominated Phase. Once SLSQP-based solutions emerge, large structural edits increasingly destabilize high-quality configurations. EvoX therefore shifts operator bias toward local refinement. Refinement now operates within a significantly stronger optimization framework, primarily adjusting convergence parameters, constraint tolerances, and step-size dynamics. Quantile-biased sampling further stabilizes search by relaxing strict elite selection, preventing overfitting to brittle local optima. Incremental gains observed in this phase reflect convergence stabilization rather than structural discovery.

This example highlights a central property of EvoX: strategy evolution enables optimization-mechanism discovery rather than merely improving sampling efficiency. Early improvements arise from discovering viable geometric heuristics, whereas later improvements require identifying mechanisms capable of coordinated global updates. Static strategies typically fail to induce such transitions because variation operators alone rarely trigger paradigm shifts, uniform sampling suppresses structurally novel candidates, and exploitative refinement reinforces local heuristic biases.

By evolving strategies conditioned on both historical feedback and population-state signals, EvoX identifies when refinement-based search becomes ineffective and increases structural variation pressure until new optimization mechanisms emerge. Overall, Circle Packing illustrates how adaptive strategy evolution governs phase transitions in search dynamics, enabling qualitative improvements that static search policies fail to realize.