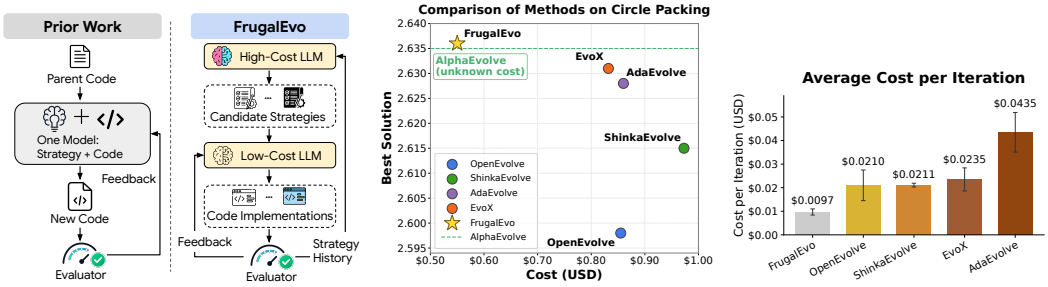
FRUGALEVO: TOWARDS COST-AWARE LLM-GUIDED PROGRAM EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-guided evolutionary methods, such as AlphaEvolve, have emerged as powerful approaches for challenging computational optimization problems, such as circle packing. However, prior work typically optimizes performance gain over a fixed number of iterations. We argue that practical optimization should maximize **gain per unit cost**. To this end, we propose `FrugalEvo`, a cost-aware evolutionary framework where a stronger, higher-cost LLM explores solution strategies, and a cheaper LLM implements them and iteratively refines the resulting code. We also design a cache-efficient evolution process, where our harness and prompts maximize the sharing of prefixes across different evolution steps, to improve cache reuse. To measure solution quality throughout a fixed cost budget, we introduce Budget-Aware Area Under the Curve (BA-AUC), defined as the area under the best-so-far evaluation score curve over cumulative LLM cost, up to the budget. Across 10 mathematical and systems optimization tasks, `FrugalEvo` matches or surpasses state-of-the-art baselines, including OpenEvolve, ShinkaEvolve, AdaEvolve, and EvoX, in final solution quality and achieves higher BA-AUC on 9 tasks. It also achieves higher average performance than these baselines on 10 algorithmic optimization tasks from ALE-Bench-Lite. Notably, on circle packing, `FrugalEvo` achieves new state-of-the-art performance with GPT-5.6 Terra and Luna for only $1.68 and with GLM-5.3 and its Flash variant for only $0.55, matching or surpassing all baselines, including multi-agent methods such as CORAL and SwarmResearch, which cost approximately $50 on average.

1 INTRODUCTION



(a) Comparison of frameworks. (b) Solution quality vs. cost. (c) Average cost per iteration.

Figure 1: High-level overview of `FrugalEvo`. (a) compares the `FrugalEvo` framework with those of prior work. (b) and (c) evaluate methods on the circle packing task: (b) plots solution quality against cumulative API cost, and (c) reports the average cost per iteration. `FrugalEvo` pairs GLM 5.3 with the cheaper GLM 5.3 Flash, whereas all other methods use GLM 5.3 alone. Results are averaged over three runs, each given a budget of $1 (USD).

Rapid advances in large language models (LLMs) have accelerated scientific discovery across diverse domains (Feng et al., 2026b; Liu et al., 2026a; Wang et al., 2026). In particular, LLM-guided evolutionary methods, such as AlphaEvolve (Novikov et al., 2025), OpenEvolve (Sharma, 2025),

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and ShinkaEvolve (Lange et al., 2026), have shown promise in computational optimization problems, including mathematics (Glazer et al., 2024), systems (Cheng et al., 2025), and algorithmic optimization (Imajuku et al., 2026). These methods typically maintain a pool of high-performing programs or solutions; they iteratively select a parent from this pool, prompt an LLM to propose a mutation, and then evaluate the resulting candidate. Progress is often measured as performance after a given number of evolutionary iterations (Liu et al., 2026b; Cemri et al., 2026), with curves plotted against the number of LLM-generated candidates or evaluator calls.

However, focusing on performance after a fixed number of iterations may overlook a key practical constraint in computational optimization: the total cost. This cost may include API fees or GPU compute costs for LLM inference, CPU compute costs for evaluator execution, and other CPU/GPU resource costs associated with task execution. On many challenging tasks, AI agents achieve better performance with more inference-time computation, a phenomenon known as *test-time scaling* (Snell et al., 2024; Brown et al., 2024; Muennighoff et al., 2025). Consequently, higher performance alone does not establish greater efficiency, as it may simply reflect higher expenditure. We therefore argue that methods should be assessed by their performance-cost trade-offs and their ability to maximize solution quality under a fixed cost budget. This shifts the focus from gain per iteration to **gain per unit cost**. To measure gain per unit cost, we introduce a new metric, Budget-Aware Area Under the Curve (BA-AUC), based on the performance achieved at each cost budget.

Next, we propose `FrugalEvo`, a cost-aware LLM-guided evolutionary framework for computational optimization. Since LLM inference often accounts for the largest proportion of search costs in this domain, compared with those of evaluator execution and other CPU/GPU operations, we focus on improving the cost efficiency of LLM inference. As shown in Figure 1a, `FrugalEvo` splits candidate program generation into two steps and uses two different models to carry them out: a strong, high-cost LLM explores diverse, promising design strategies, while a weaker, low-cost LLM implements them as executable programs and iteratively refines the resulting code.

Beyond model collaboration, we design a cache-efficient evolution process for `FrugalEvo`. Inference systems and APIs that support prompt caching can reuse cached key-value (KV) states for identical prompt prefixes, reducing latency and cost. To facilitate this reuse, we design our harness and prompts to maximize the sharing of prefixes across different evolution steps, improving cache reuse.

We evaluate `FrugalEvo` on 20 tasks spanning mathematical, systems, and algorithmic optimization (Section 3). With two model configurations—GPT-5.6 Terra and Luna (OpenAI, 2026), and GLM-5.3 and its Flash variant (Z.ai, 2026a;b)—`FrugalEvo` matches or outperforms OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2026), AdaEvolve (Cemri et al., 2026), and EvoX (Liu et al., 2026b) in final solution quality and BA-AUC on 9 of 10 mathematical and systems optimization tasks. It also achieves the highest average performance across 10 algorithmic optimization tasks and matches or surpasses the reported AlphaEvolve (Novikov et al., 2025) results on all four mathematical tasks with available comparisons. Notably, as shown in Figure 1b, `FrugalEvo` achieves a sum of radii of 2.63599 on *Circle Packing* using GLM-5.3 and its Flash variant for only \$0.55, surpassing all baselines, including multi-agent methods which cost approximately \$50 on average. Moreover, Figure 1c shows that `FrugalEvo` substantially reduces the average cost per iteration.

In summary, our contributions include:

- We argue that practical LLM-guided evolutionary search in computational optimization should focus on improving performance gain per unit cost. In this work, we introduce Budget-Aware Area Under the Curve (BA-AUC) to assess solution quality throughout a fixed LLM cost budget, complementing evaluation based on final scores.
- We propose `FrugalEvo`, a cost-aware evolutionary framework that combines strategy exploration by a strong, high-cost model with code implementation and refinement by a low-cost model, and design a cache-efficient evolution process.
- We evaluate `FrugalEvo` on 20 real-world computational optimization tasks. `FrugalEvo` matches or surpasses all baselines, including OpenEvolve, ShinkaEvolve, AdaEvolve, and EvoX, in solution quality and achieves higher BA-AUC on 9 of 10 mathematical and systems optimization tasks. It also achieves the highest average performance on 10 algorithmic optimization tasks.

2 FRUGALEVO: COST-AWARE PROGRAM EVOLUTION

2.1 PROBLEM FORMULATION

Following prior work (Novikov et al., 2025; Romera-Paredes et al., 2024), we formulate LLM-guided evolutionary search as a program optimization process under a fixed budget.

Candidate Evaluation. Let $\mathcal{X}$ denote the program space and let $x_0 \in \mathcal{X}$ be an initial program, which is an example given by the task. An evaluator evaluates each candidate $x \in \mathcal{X}$ and returns a task-specific score and auxiliary feedback:

$$E(x) = (s(x), a(x)), \qquad E : \mathcal{X} \to \mathbb{R} \times \mathcal{A}, \tag{1}$$

where $s(x) \in \mathbb{R}$ is the task-specific score to be maximized and $a(x) \in \mathcal{A}$ contains auxiliary feedback.

Program Database and Search Context. Evaluated programs and their results are stored in a program database. Evaluating the initial program x_0 initializes the database as $\mathcal{D}_0 = \{(x_0, s_0, a_0)\}$. At iteration t, a search strategy S constructs a search context $C_S(\mathcal{D}_t)$ by combining the task description and analysis with a parent program, sampled reference programs, their evaluation results, and summaries of previous attempts retrieved from $\mathcal{D}_t$. Conditioned on this context, an LLM G_{LLM} generates a new candidate, which is then evaluated:

$$x_{t+1} \sim G_{\mathrm{LLM}}(\cdot \mid C_S(\mathcal{D}_t)), \qquad E(x_{t+1}) = (s_{t+1}, a_{t+1}). \tag{2}$$

The resulting record is added to the database as $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(x_{t+1}, s_{t+1}, a_{t+1})\}$.

Optimization Goal. As cost is a key factor in determining the practical utility of an approach, we focus on maximizing solution quality

$$s^\star = \max_{(x,s,a)\in\mathcal{D}_B} s \tag{3}$$

under a fixed cost budget B.

Budget-Aware AUC. We introduce Budget-Aware Area Under the Curve (BA-AUC) to measure how efficiently a method improves solution quality within a fixed LLM cost budget. Let $q(c)$ denote the best-so-far evaluation score at cumulative LLM cost c. For a fixed cost budget B, we define

$$\text{BA-AUC}(B) = \int_0^B q(c)\,\mathrm{d}c. \tag{4}$$

To compare all methods over the same cost interval $[0, B]$, we hold the best-so-far evaluation score constant up to B if a run ends before reaching the budget, and truncate the curve at B if the run exceeds it.

2.2 OUR APPROACH

Overview. Figure 2 presents an overview of our approach, which proceeds as follows: (1) starting from an initial program, the cold start module establishes an "incumbent"—the best-performing program found so far—as the parent code for strategy exploration and solution generation; (2) the context builder combines the problem context, task analysis, and incumbent program with information from search memory; (3) the strategy explorer uses a high-cost LLM to propose multiple design strategies; (4) the solution generator uses a low-cost LLM to implement and refine the proposed strategies; (5) the search memory updates the search context for the next round of strategy exploration; and (6) the process repeats until the budget is exhausted, returning the best program found. A detailed algorithm description is provided in Appendix A and prompts of strategy explorer and solution generator can be found in Appendix D.

Cold Start. The strategy explorer generates design strategies based on a parent program. Using a weak initial program as the parent may lead to low-quality strategies and inefficient use of the cost budget. We therefore introduce a cold-start phase in which a low-cost LLM iteratively refines the initial program until the evaluation score no longer improves. The best resulting program becomes the initial incumbent and serves as the parent for the first round of strategy exploration.

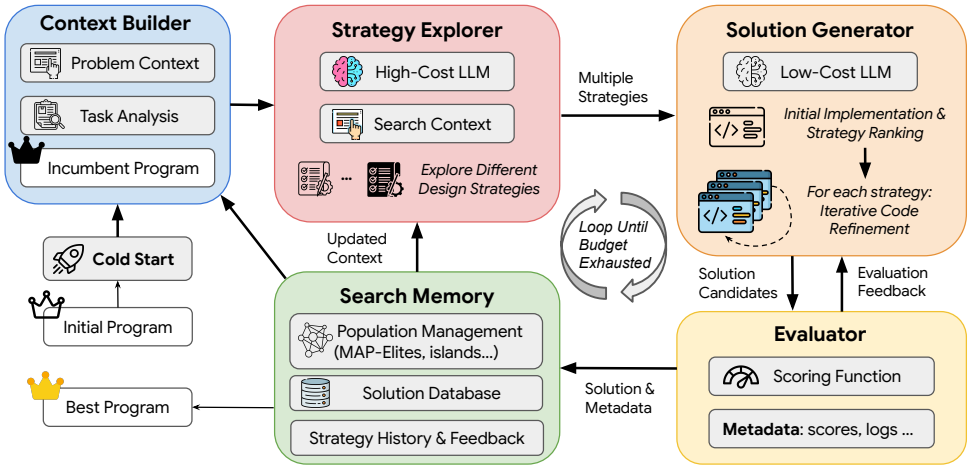


Figure 2: The `FrugalEvo` framework. It comprises six main components: (1) a cold-start module, (2) a context builder, (3) a strategy explorer, (4) a solution generator, (5) an evaluator, and (6) a search memory. Our search memory uses island-based MAP-Elites (Mouret & Clune, 2015; Tanese, 1989) algorithm.

Context Builder. The context builder organizes task information and search history to guide strategy exploration. At the start of evolution, the context builder uses a high-cost LLM to analyze the problem description and evaluator code to identify the optimization objective and evaluation constraints. It combines this information with the incumbent program established by cold start and its evaluation feedback to construct the initial search context. As the search progresses, it updates the context with the latest incumbent and attempted programs, strategies, and evaluation feedback retrieved from the search memory, guiding subsequent strategy exploration.

Cache-Efficient Prompt Construction. Specifically, to reduce input-token costs, we construct cache-efficient prompts by placing shared content at the beginning, enabling prefix reuse through API prompt caching.[1] To maximize prefix reuse between attempts, each prompt begins with fixed content (task instructions, output requirements, etc.). This is followed by information which changes less frequently, e.g., contextual examples, with island-best and elite programs appearing first before randomly sampled programs, lastly followed by frequently changing information, preserving the maximum amount of shared prefix across successive attempts. Additionally, feedback from each attempt is appended to the end of the prompt, allowing cache usage.

Strategy Explorer. Since algorithmic design can substantially affect solution quality, we use a strong, high-cost LLM to explore diverse design strategies. Given the search context, including the parent program, evaluation feedback, and previously explored strategies, it proposes multiple distinct strategies in a single API call. Each strategy provides a concrete plan for modifying the parent program, explaining what to change, why it may improve performance, and what computational limits to follow. The proposed strategies are then passed to the solution generator for implementation and refinement.

Solution Generator. The solution generator uses a low-cost LLM to implement and refine the proposed strategies. It first generates one trial implementation per strategy and ranks the strategies by their evaluation scores. It then follows this ranking to select strategies for further refinement. Each refinement round fixes one strategy and the current incumbent as the parent program, allowing up to M sequential implementation attempts. Each attempt is evaluated, and feedback from unsuccessful attempts informs subsequent prompts while the parent remains unchanged. As soon as an attempt improves the incumbent, the round ends, the improved program becomes the new in-

[1] A couple of LLM API providers support prompt caching, allowing repeated prompt prefixes to be reused at a lower input-token cost; e.g., OpenAI: `https://developers.openai.com/api/docs/guides/prompt-caching`.

cumbent, and another round begins with the same strategy and the updated parent. If a round ends without improvement, the generator moves to the next strategy.

Evaluator and Search Memory. The evaluator assesses each generated candidate program using task-specific scoring functions. Search memory uses island-based MAP-Elites (Mouret & Clune, 2015; Tanese, 1989) to maintain program diversity. It stores candidate programs and their evaluation results, along with proposed strategies, the results of strategy attempts, and failure feedback. The evolution process stops when the remaining budget cannot cover further LLM calls and returns the best program found.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Tasks. We evaluate `FrugalEvo` on 20 real-world optimization tasks, including 5 mathematics optimization problems (Sharma, 2025), 5 systems optimization tasks from the ADRS benchmark (Cheng et al., 2025), and 10 algorithmic optimization problems from ALE-Bench-Lite (Imajuku et al., 2026). Benchmark details are provided in Appendix B.

Baselines. We compare `FrugalEvo` against strong LLM-guided evolutionary methods, including OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2026), AdaEvolve (Cemri et al., 2026), and EvoX (Liu et al., 2026b). For mathematical tasks, we additionally report state-of-the-art results from AlphaEvolve (Novikov et al., 2025).

Model Configurations and Budget. We use two model configurations in our evaluation. In the first configuration, `FrugalEvo` uses GPT-5.6 Terra (OpenAI, 2026) for strategy generation and GPT-5.6 Luna (OpenAI, 2026) for code implementation and refinement, while all four baselines use GPT-5.6 Terra. Each run receives an LLM cost budget of $2 per run for mathematical optimization tasks and $1 for systems and algorithmic optimization tasks. In the second configuration, `FrugalEvo` uses GLM-5.3 (Z.ai, 2026a) for strategy generation and GLM-5.3 Flash (Z.ai, 2026b) for implementation and refinement, while other baselines use GLM-5.3. The corresponding budgets are $1 for mathematical tasks and $0.5 for systems optimization tasks. Model Pricing details are provided in Appendix C.

3.2 MAIN RESULTS

Results on Math Optimization Tasks. We evaluate five tasks spanning geometric packing, extremal geometry, distance-ratio minimization, and multi-objective signal filtering. As shown in Table 1, `FrugalEvo` achieves the highest mean BA-AUC on all five tasks under both model configurations. It also achieves the best or comparable mean performance on all five tasks using GPT-5.6 Terra and Luna and four using GLM-5.3 and the Flash variant. Compared to the strong reported results in the closed-source AlphaEvolve, `FrugalEvo` matches or exceeds AlphaEvolve on four tasks, including *Circle Packing*, *Heilbronn Convex (13)*, *Heilbronn Triangle*, and *Min-Max-3*.

Solution Analysis. Beyond aggregate scores, `FrugalEvo` discovers qualitatively distinct solutions across different tasks. On *Circle Packing*, it finds a packing with a total radius of 2.635996 by evolving five-row layouts with asymmetric displacements, refining them through constrained SLSQP optimization and fixed-center radius linear programming, and applying small radius increases within the evaluator's tolerance.

Figure 3 visualizes the solutions discovered by `FrugalEvo` and compares them with the initial program on *Circle Packing*. It shows that `FrugalEvo` improves the sum of radii on *Circle Packing* from 0.95976 to 2.635996 using GPT-5.6 Terra and Luna for $1.68, and to 2.635990 using GLM-5.3 and its Flash variant for $0.55. These results are better than or comparable to those of CORAL (Qu et al., 2026) and SwarmResearch (Virk et al., 2026), which cost approximately $50 on average to achieve 2.635985 and 2.635996, respectively, demonstrating substantial cost savings.

Moreover, on *Signal Processing*, `FrugalEvo` reaches 0.78912 by adaptively blending Savitzky–Golay, Butterworth, and spectral filters while suppressing minor reversals. On *Heilbronn*

Table 1: **Results on mathematical optimization problems.** We report mean ± sample standard deviation and best scores over three independent runs. "(↑)" means the higher the better, while "(↓)" means the lower the better. **Bold** indicates the best result for each metric within each task and model configuration. Light green highlights the best performance within each task across both model configurations.

	GPT-5.6 Terra/Luna ($2)			GLM-5.3/Flash ($1)		
Method	Avg.	Best	BA-AUC@2	Avg.	Best	BA-AUC@1
Circle Packing (↑) AlphaEvolve: 2.635000						
OpenEvolve	$2.634206_{\pm 0.0015}$	2.635983	$1.9709_{\pm 0.0007}$	$2.555418_{\pm 0.0321}$	2.598002	$0.9438_{\pm 0.0157}$
ShinkaEvolve	$2.630981_{\pm 0.0003}$	2.631285	$1.9653_{\pm 0.0098}$	$2.504844_{\pm 0.0837}$	2.615012	$0.9280_{\pm 0.0259}$
AdaEvolve	$2.632514_{\pm 0.0031}$	2.634305	$1.9653_{\pm 0.0039}$	$2.618077_{\pm 0.0088}$	2.628041	$0.9375_{\pm 0.0162}$
EvoX	$2.635420_{\pm 0.0010}$	2.635983	$1.9549_{\pm 0.0228}$	$2.599142_{\pm 0.0434}$	2.630972	$0.9497_{\pm 0.0337}$
FrugalEvo	$\mathbf{2.635989_{\pm 0.0000}}$	**2.635996**	$\mathbf{1.9925_{\pm 0.0028}}$	$\mathbf{2.635422_{\pm 0.0008}}$	**2.635990**	$\mathbf{0.9939_{\pm 0.0039}}$
Signal Processing (↑)						
OpenEvolve	$0.55003_{\pm 0.0264}$	0.58045	$1.0802_{\pm 0.0308}$	$0.66438_{\pm 0.0746}$	0.74245	$0.6297_{\pm 0.0662}$
ShinkaEvolve	$0.55604_{\pm 0.0301}$	0.59012	$1.0854_{\pm 0.0251}$	$0.66246_{\pm 0.0475}$	0.71555	$0.6305_{\pm 0.0321}$
AdaEvolve	$0.72031_{\pm 0.0224}$	0.73677	$1.2520_{\pm 0.0225}$	$0.67621_{\pm 0.0068}$	0.68281	$0.6315_{\pm 0.0086}$
EvoX	$0.64383_{\pm 0.0377}$	0.66574	$1.1706_{\pm 0.0428}$	$0.67441_{\pm 0.0535}$	0.70925	$0.6401_{\pm 0.0609}$
FrugalEvo	$\mathbf{0.77791_{\pm 0.0145}}$	**0.78912**	$\mathbf{1.4819_{\pm 0.0308}}$	$\mathbf{0.78404_{\pm 0.0034}}$	**0.78641**	$\mathbf{0.7539_{\pm 0.0068}}$
Heilbronn Convex (13) (↑) AlphaEvolve: 0.03094						
OpenEvolve	$0.02314_{\pm 0.0024}$	0.02553	$1.3757_{\pm 0.1146}$	$0.02854_{\pm 0.0021}$	0.03077	$0.8666_{\pm 0.0498}$
ShinkaEvolve	$0.02450_{\pm 0.0028}$	0.02769	$1.4499_{\pm 0.1325}$	$0.02453_{\pm 0.0028}$	0.02758	$0.6531_{\pm 0.1748}$
AdaEvolve	$0.02349_{\pm 0.0011}$	0.02470	$1.4224_{\pm 0.0348}$	$0.02797_{\pm 0.0021}$	0.03024	$0.8371_{\pm 0.0950}$
EvoX	$0.02248_{\pm 0.0002}$	0.02262	$1.3840_{\pm 0.0555}$	$0.02896_{\pm 0.0008}$	0.02985	$0.8279_{\pm 0.0208}$
FrugalEvo	$\mathbf{0.02751_{\pm 0.0005}}$	**0.02781**	$\mathbf{1.7405_{\pm 0.0393}}$	$\mathbf{0.03079_{\pm 0.0002}}$	**0.03094**	$\mathbf{0.9614_{\pm 0.0073}}$
Heilbronn Triangle (↑) AlphaEvolve: 0.03653						
OpenEvolve	$0.03417_{\pm 0.0034}$	**0.03653**	$1.7517_{\pm 0.1833}$	$0.03139_{\pm 0.0042}$	0.03598	$0.7256_{\pm 0.1359}$
ShinkaEvolve	$0.03030_{\pm 0.0004}$	0.03061	$1.4825_{\pm 0.0819}$	$0.03051_{\pm 0.0010}$	0.03168	$0.7561_{\pm 0.0255}$
AdaEvolve	$0.03077_{\pm 0.0019}$	0.03273	$1.5951_{\pm 0.0485}$	$0.03337_{\pm 0.0010}$	0.03456	$0.8192_{\pm 0.0505}$
EvoX	$0.03188_{\pm 0.0041}$	**0.03653**	$1.6444_{\pm 0.2031}$	$\mathbf{0.03486_{\pm 0.0016}}$	0.03617	$0.8316_{\pm 0.0260}$
FrugalEvo	$\mathbf{0.03480_{\pm 0.0008}}$	0.03568	$\mathbf{1.8733_{\pm 0.0132}}$	$0.03463_{\pm 0.0017}$	**0.03653**	$\mathbf{0.8659_{\pm 0.0893}}$
Min-Max-3 (↓) AlphaEvolve: 4.165850						
OpenEvolve	$\mathbf{4.165783_{\pm 0.0000}}$	**4.165783**	$1.9602_{\pm 0.0025}$	$4.168021_{\pm 0.0039}$	**4.165783**	$0.9740_{\pm 0.0180}$
ShinkaEvolve	$4.166404_{\pm 0.0011}$	**4.165783**	$1.9356_{\pm 0.0096}$	$4.167856_{\pm 0.0019}$	**4.165783**	$0.9737_{\pm 0.0043}$
AdaEvolve	$\mathbf{4.165783_{\pm 0.0000}}$	**4.165783**	$1.9464_{\pm 0.0117}$	$4.165784_{\pm 0.0000}$	**4.165783**	$0.9820_{\pm 0.0079}$
EvoX	$\mathbf{4.165783_{\pm 0.0000}}$	**4.165783**	$1.9419_{\pm 0.0020}$	$4.166583_{\pm 0.0014}$	**4.165783**	$0.9798_{\pm 0.0051}$
FrugalEvo	$\mathbf{4.165783_{\pm 0.0000}}$	**4.165783**	$\mathbf{1.9769_{\pm 0.0159}}$	$\mathbf{4.165783_{\pm 0.0000}}$	**4.165783**	$\mathbf{0.9956_{\pm 0.0004}}$

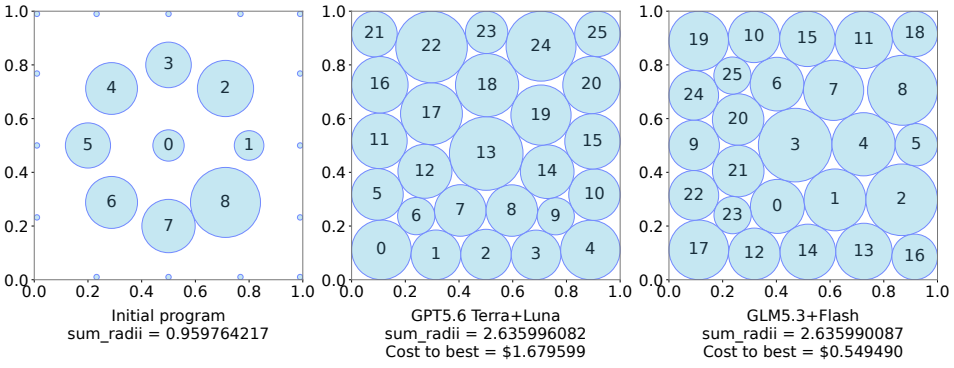


Figure 3: Visualization of `FrugalEvo` solutions on the circle packing task. *Left*: initial program; *Middle*: `FrugalEvo` solution generated by GPT-5.6 Terra and Luna; *Right*: `FrugalEvo` solution generated by GLM 5.3 and its Flash variant.

Convex (13), `FrugalEvo` matches AlphaEvolve through multistart hill climbing, linear programming refinement, and convex-hull area reduction. These results indicate the effectiveness of our approach: using a stronger model to explore strategies and a weaker model to iteratively refine solutions.

Table 2: **Results on systems optimization problems.** We report mean ± sample standard deviation and best scores over three runs. "(↑)" means the higher the better, while "(↓)" means the lower the better. **Bold** indicates the best result for each metric within each task and model configuration. Light green highlights the best performance within each task across both model configurations.

	GPT-5.6 Terra/Luna ($1)			GLM-5.3/Flash ($0.5)		
Method	Avg.	Best	BA-AUC@1	Avg.	Best	BA-AUC@0.5
EPLB (↑)						
OpenEvolve	$0.1346_{\pm 0.0097}$	0.1457	$0.1291_{\pm 0.0006}$	$0.1352_{\pm 0.0101}$	0.1468	$0.0672_{\pm 0.0046}$
ShinkaEvolve	$0.1348_{\pm 0.0097}$	0.1460	$0.1314_{\pm 0.0044}$	$0.1404_{\pm 0.0094}$	0.1469	$0.0678_{\pm 0.0027}$
AdaEvolve	$0.1295_{\pm 0.0004}$	0.1297	$0.1293_{\pm 0.0003}$	$0.1291_{\pm 0.0012}$	0.1298	$0.0643_{\pm 0.0005}$
EvoX	$0.1355_{\pm 0.0099}$	0.1470	$0.1320_{\pm 0.0044}$	$0.1294_{\pm 0.0006}$	0.1298	$0.0641_{\pm 0.0002}$
FrugalEvo	$\mathbf{0.1435_{\pm 0.0058}}$	**0.1471**	$\mathbf{0.1408_{\pm 0.0047}}$	$\mathbf{0.1471_{\pm 0.0003}}$	**0.1473**	$\mathbf{0.0724_{\pm 0.0008}}$
PRISM (↑)						
OpenEvolve	$26.2307_{\pm 0.0400}$	**26.2560**	$25.9249_{\pm 0.0995}$	$\mathbf{26.2560_{\pm 0.0000}}$	**26.2560**	$12.9947_{\pm 0.0447}$
ShinkaEvolve	$26.2136_{\pm 0.0276}$	26.2332	$25.9399_{\pm 0.1104}$	$\mathbf{26.2560_{\pm 0.0000}}$	**26.2560**	$12.8018_{\pm 0.1138}$
AdaEvolve	$25.9493_{\pm 0.1707}$	26.0818	$25.1931_{\pm 0.4972}$	$26.2539_{\pm 0.0035}$	**26.2560**	$12.8213_{\pm 0.1775}$
EvoX	$25.8464_{\pm 0.3368}$	26.2200	$24.9843_{\pm 0.5623}$	$26.2217_{\pm 0.0594}$	**26.2560**	$12.7215_{\pm 0.2384}$
FrugalEvo	$\mathbf{26.2560_{\pm 0.0000}}$	**26.2560**	$\mathbf{26.0695_{\pm 0.1844}}$	$\mathbf{26.2560_{\pm 0.0000}}$	**26.2560**	$\mathbf{13.1117_{\pm 0.0030}}$
LLM-SQL (↑)						
OpenEvolve	$0.7165_{\pm 0.0010}$	0.7175	$0.6263_{\pm 0.0258}$	$0.7262_{\pm 0.0052}$	0.7296	$0.3349_{\pm 0.0044}$
ShinkaEvolve	$0.7204_{\pm 0.0058}$	0.7271	$0.6818_{\pm 0.0077}$	$0.7236_{\pm 0.0066}$	0.7307	$0.3397_{\pm 0.0121}$
AdaEvolve	$0.7190_{\pm 0.0004}$	0.7193	$0.6502_{\pm 0.0047}$	$0.7259_{\pm 0.0076}$	0.7321	$0.3374_{\pm 0.0064}$
EvoX	$0.7183_{\pm 0.0009}$	0.7192	$0.6426_{\pm 0.0319}$	$0.7239_{\pm 0.0084}$	0.7302	$0.3370_{\pm 0.0080}$
FrugalEvo	$\mathbf{0.7248_{\pm 0.0085}}$	**0.7313**	$\mathbf{0.7161_{\pm 0.0057}}$	$\mathbf{0.7336_{\pm 0.0009}}$	**0.7346**	$\mathbf{0.3566_{\pm 0.0121}}$
Cloudcast (↓)						
OpenEvolve	$618.103_{\pm 0.000}$	618.103	$0.1572_{\pm 0.0021}$	$637.112_{\pm 17.314}$	627.100	$0.0702_{\pm 0.0063}$
ShinkaEvolve	$618.098_{\pm 0.004}$	618.093	$0.1568_{\pm 0.0027}$	$725.286_{\pm 19.810}$	705.094	$0.0575_{\pm 0.0069}$
AdaEvolve	$618.098_{\pm 0.000}$	618.098	$\mathbf{0.1578_{\pm 0.0008}}$	$785.177_{\pm 216.772}$	627.132	$0.0622_{\pm 0.0126}$
EvoX	$618.103_{\pm 0.000}$	618.103	$0.1572_{\pm 0.0015}$	$630.026_{\pm 9.204}$	622.617	$0.0638_{\pm 0.0087}$
FrugalEvo	$\mathbf{618.093_{\pm 0.016}}$	**618.075**	$0.1565_{\pm 0.0016}$	$\mathbf{622.601_{\pm 4.506}}$	**618.098**	$\mathbf{0.0776_{\pm 0.0013}}$
Transaction Scheduling (↑)						
OpenEvolve	$4091.92_{\pm 169.16}$	4237.29	$3893.72_{\pm 150.82}$	$4052.41_{\pm 193.96}$	4291.85	$1971.43_{\pm 130.54}$
ShinkaEvolve	$4023.16_{\pm 203.94}$	4149.38	$3734.62_{\pm 111.63}$	$3952.25_{\pm 240.14}$	4115.23	$1840.84_{\pm 71.76}$
AdaEvolve	$3858.51_{\pm 119.85}$	3984.06	$3720.02_{\pm 38.42}$	$4047.19_{\pm 237.88}$	4255.32	$1955.27_{\pm 93.63}$
EvoX	$3913.30_{\pm 271.06}$	4219.41	$3674.87_{\pm 162.27}$	$3975.20_{\pm 265.47}$	4184.10	$1800.25_{\pm 116.93}$
FrugalEvo	$\mathbf{4252.11_{\pm 110.39}}$	**4405.29**	$\mathbf{4133.73_{\pm 124.12}}$	$\mathbf{4288.01_{\pm 98.33}}$	**4366.81**	$\mathbf{2107.07_{\pm 42.86}}$

Results on Systems Optimization Tasks. Table 2 reports results on five real-world systems optimization tasks from the ADRS benchmark (Cheng et al., 2025). It shows that `FrugalEvo` achieves the best or comparable mean and best performance on all five tasks under both model configurations, with the highest mean BA-AUC on four tasks.

Qualitatively, `FrugalEvo` discovers effective approaches to systems optimization. On *EPLB*, it achieves a score of 0.1473 with an expert load-balancing algorithm that combines greedy expert replication, load-aware block assignment, and pairwise swap refinement. On *Transaction Scheduling*, it achieves a combined score of 4405.29 with a scheduler that combines multistart regret-guided insertion, exhaustive local window reordering, and transaction relocation to reduce makespan.

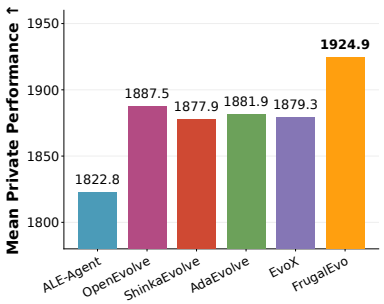


Figure 4: Average private performance across 10 ALE-Bench-Lite tasks. `FrugalEvo` uses GPT-5.6 Terra and Luna and other baselines use GPT-5.6 Terra. The budget is $1.

Results on Algorithmic Optimization Tasks. We evaluate `FrugalEvo` on 10 algorithmic optimization tasks from ALE-Bench-Lite (Imajuku et al., 2026), derived from AtCoder Heuristic Contests, and report the results in Figure 4. Following prior work (Lange et al., 2026), all methods start from solutions produced by ALE-Agent (Imajuku et al., 2026) and are evaluated using private scores.

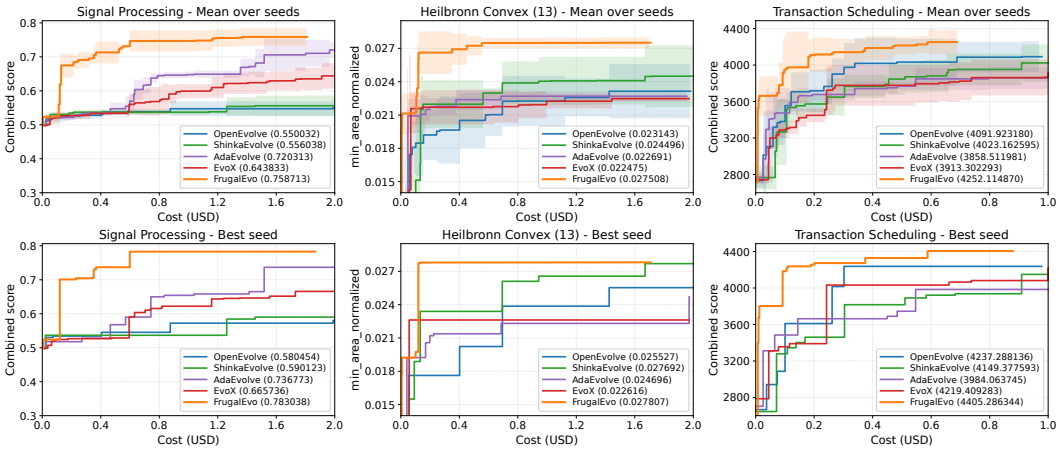


Figure 5: Best-so-far score curves on signal processing, heilbronn convex, and transaction scheduling. `FrugalEvo` uses GPT-5.6 Terra and Luna, while the baselines use GPT-5.6 Terra. The top row shows mean trajectories across three runs, and the bottom row shows the trajectory of the best-performing run for each method. BA-AUC values are printed along with method names.

As shown in Figure 4, `FrugalEvo` achieves the highest score of 1924.9, outperforming strong baselines OpenEvolve (1887.5) and AdaEvolve (1881.9). It discovers diverse algorithmic optimization solutions. On AHC016 (graph classification), `FrugalEvo` constructs graphs from disjoint cliques and their complements, identifying noisy graphs by comparing the sorted numbers of neighbors per vertex with noise-adjusted expectations. On AHC027 (cleaning route), `FrugalEvo` uses dirt accumulation rates, time since the last visit, and travel distances to plan routes that revisit areas requiring more frequent cleaning.

Performance-For-Cost Curves. To compare cost efficiency, we plot performance-for-cost curves of different methods on *Signal Processing*, *Heilbronn Convex (13)*, and *Transaction Scheduling*. As shown in Figure 5, `FrugalEvo` establishes a clear performance advantage within the first $0.2 on all three tasks and maintains it throughout the remaining search. Its higher mean BA-AUC values on all three tasks further quantify this advantage over the cost interval. These results indicate that `FrugalEvo` achieves higher search efficiency within a given cost budget.

3.3 Ablations

In this section, we present ablation studies on key components of `FrugalEvo` (model collaboration, cold start, and sequential feedback in code refinement) and the results are shown in Table 3.

Impact of Model Collaboration. Our framework assigns strategy exploration and solution generation to different models to improve search efficiency. As shown in Table 3, using either the high-cost or low-cost model for both stages yields lower mean and best performance than `FrugalEvo` on both tasks under the same cost budget. In particular, using only the high-cost model leads to a larger drop in mean performance, possibly because implementation calls consume a substantial portion of the budget, leaving less for further exploration. Moreover, using only the low-cost model reduces the best final score, particularly on *Signal Processing*. This may be because the low-cost model is less effective at identifying promising strategies. These results suggest that combining strategy exploration by a strong, expensive model with low-cost code implementation improves search efficiency under a fixed cost budget.

Table 3: Ablation results with GPT-5.6 Terra and Luna.

Method	Performance Avg.	Best
Circle Packing (↑)		
FrugalEvo	$\mathbf{2.635989}_{\pm 0.0000}$	**2.635996**
- Terra + Terra	$2.632278_{\pm 0.0063}$	2.635983
- Luna + Luna	$2.634565_{\pm 0.0025}$	2.635983
w/o cold start	$2.632417_{\pm 0.0031}$	2.635983
w/o seq feedback	$2.633659_{\pm 0.0021}$	**2.635996**
Signal Processing (↑)		
FrugalEvo	$\mathbf{0.77791}_{\pm 0.0145}$	**0.78912**
- Terra + Terra	$0.68534_{\pm 0.0352}$	0.70706
- Luna + Luna	$0.74865_{\pm 0.0097}$	0.75973
w/o cold start	$0.75576_{\pm 0.0240}$	0.77973
w/o seq feedback	$0.76072_{\pm 0.0209}$	0.77438

Impact of Cold Start and Sequential Feedback. We introduce a cold-start phase to provide a stronger initial incumbent for strategy exploration. As shown in Table 3, removing this phase and using the initial program directly as the parent reduces both mean and best performance on both tasks. This suggests that refining the initial program with a low-cost model is a cost-effective way to support high-quality strategy exploration. Moreover, we also incorporate sequential feedback into code refinement in the solution generator. Removing this feedback lowers mean performance and increases the standard deviation across runs, suggesting that feedback from earlier attempts helps the low-cost model refine programs more effectively and improves search stability.

4 Related Work

LLM-Guided Evolutionary Methods. Recent work combining LLMs with evolutionary search has produced notable results (Agrawal et al., 2026; Novikov et al., 2025) for mathematical (Feng et al., 2026c;d;b), algorithmic (Mang et al., 2025), and systems (Cheng et al., 2025) optimization. For example, FunSearch (Romera-Paredes et al., 2024) combines LLM generation with automated evaluation to evolve mathematical solutions. AlphaEvolve (Novikov et al., 2025) extends this approach from evolving individual functions to larger programs. Subsequently, OpenEvolve (Sharma, 2025) and CodeEvolve (Assumpção et al., 2025) provide open-source frameworks, and ShinkaEvolve (Lange et al., 2026) introduces a sample-efficient evolution method. ThetaEvolve (Wang et al., 2025) and TTT-Discover (Yuksekgonul et al., 2026) apply Reinforcement Learning (RL) to improve the LLM generator at test time. More recently, AdaEvolve (Cemri et al., 2026) and EvoX (Liu et al., 2026b) adapt the search process, while SwarmResearch (Virk et al., 2026) and CORAL (Qu et al., 2026) coordinate multiple coding agents to improve search performance. These methods focus on performance gains over a fixed number of iterations, while our work emphasizes gain per unit cost.

Cost-Efficient LLM Inference. Prior work explores LLM cascades (Chen et al., 2023), routing (Ong et al., 2025; Feng et al., 2026a), reflection (Agrawal et al., 2026; Zhang et al., 2026a), and caching (Zhang et al., 2026b) to improve cost efficiency. For instance, FrugalGPT (Chen et al., 2023) learns model cascades to balance cost and accuracy, while RouteLLM (Ong et al., 2025) learns to route queries between strong and weak models using preference data. For scientific discovery, SimpleTES (Ye et al., 2026) post-trains an open-source model for long-horizon AI discovery by assigning each attempt the final outcome of the trajectory it helped produce. Meanwhile, Squeeze Evolve (Maheswaran et al., 2026) coordinates multiple models for verifier-free evolution using models' own confidence signals, such as token log-probabilities. Our approach allocates model capability based on the characteristics of computational optimization tasks, without additional model training or access to token-level probabilities.

5 Conclusion

We introduce `FrugalEvo`, a cost-aware LLM-guided evolutionary framework for computational optimization, that combines strategy exploration by a strong, high-cost model with code implementation and refinement by a low-cost model, and design a cache-efficient evolution process. To assess solution quality throughout a fixed cost budget, we propose Budget-Aware Area Under the Curve (BA-AUC). Across diverse real-world computational optimization tasks, `FrugalEvo` matches or surpasses all baselines in solution quality and achieves higher BA-AUC on the majority of tasks.

Limitations and Future Work. `FrugalEvo` splits candidate program generation in computational optimization tasks into two steps and assigns them to two different models. This design is tailored to settings where both solution generation and evaluation take place entirely within a code environment, so that candidates can be executed and scored automatically. As a result, our method does not readily extend to scientific domains that require interaction with the physical world, such as biology, where evaluating a solution may involve wet-lab experiments that cannot be automated through code execution. Moreover, guiding the model to explore effective strategies relies on carefully crafted instructions. Our current instructions are manually designed for computational optimization tasks rather than automatically derived from each task. Finally, future work includes extending this approach to other domains that require interaction with the physical world, and develop methods that automatically generate strategy exploration instructions.

AI USE STATEMENT

In preparing this submission, we used LLMs as general-purpose writing assistants and coding agents to implement the algorithm and run experiments. The human authors were responsible for the entire research process, including initial ideation, experimental design, and data analysis. For the experiments, the authors first developed an experimental plan and prepared the codebase and datasets, then instructed coding agents to implement the proposed method and execute the experiments. During writing, the authors drafted each section, including all core ideas and technical details, and used LLMs to improve clarity, coherence, and grammar. All scientific claims and findings presented in this paper originate from the authors, who take full responsibility for the final content.

ETHICS STATEMENT

Our work adheres to the ICLR Code of Ethics. The primary ethical considerations for this research involve the use of solutions generated by our algorithm across 20 optimization tasks. Although assessed by task evaluators, these solutions have not been validated by human experts and are intended for research reference only.

REPRODUCIBILITY STATEMENT

We are committed to ensuring the reproducibility of our research. To this end, we will publicly release all code, data, and supplementary materials upon publication.

REFERENCES

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. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: an open source evolutionary coding agent for algorithmic discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, et al. Adaevolve: Adaptive llm driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

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.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, et al. LLMRouter: Unified infrastructure for developing, evaluating, and deploying llm routers. *arXiv preprint arXiv:2608.06867*, 2026a.

Tony Feng, Junehyuk Jung, Sang-hyun Kim, Carlo Pagano, Sergei Gukov, Chiang-Chiang Tsai, David Woodruff, Adel Javanmard, Aryan Mokhtari, Dawsen Hwang, et al. Aletheia tackles first-proof autonomously. *arXiv preprint arXiv:2602.21201*, 2026b.

Tony Feng, Trieu Trinh, Garrett Bingham, Jiwon Kang, Shengtong Zhang, Sang-hyun Kim, Kevin Barreto, Carl Schildkraut, Junehyuk Jung, Jaehyeon Seo, et al. Semi-autonomous mathematics discovery with gemini: A case study on the erdős problems. *arXiv preprint arXiv:2601.22401*, 2026c.

Tony Feng, Trieu H Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung, Joonkyung Lee, Carlo Pagano, Sang-hyun Kim, Federico Pasqualotto, et al. Towards autonomous mathematics research. *arXiv preprint arXiv:2602.10177*, 2026d.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. *Advances in Neural Information Processing Systems*, 38, 2026.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026.

Amelia Liu, Andrew Ho, Anne Marie Droste, David Martin, Edmund Wong, Edward Zhou, Isabelle Zhou, Joshua Park, Joy Jiao, Katie-Rose Skelly, et al. LifeSciBench: Evaluating language models on realistic, expert-level tasks in the life sciences. *bioRxiv*, pp. 2026–08, 2026a.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, et al. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026b.

Monishwaran Maheswaran, Leon Lakhani, Zhongzhu Zhou, Shijia Yang, Junxiong Wang, Coleman Hooper, Yuezhou Hu, Rishabh Tiwari, Jue Wang, Harman Singh, et al. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *arXiv preprint arXiv:2604.07725*, 2026.

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.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

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

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenAI. GPT-5.6 system card. Technical report, OpenAI, July 2026. URL `https://deploymentsafety.openai.com/gpt-5-6/gpt-5-6.pdf`.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Reiko Tanese. *Distributed genetic algorithms for function optimization*. University of Michigan, 1989.

Yuvraj Virk, Zack Edds, Chunqiu Steven Xia, and Lingming Zhang. Swarmresearch: Orchestrating coding agents for open-ended discovery. *arXiv preprint arXiv:2607.02807*, 2026.

Miles Wang, Robi Lin, Kat Hu, Joy Jiao, Neil Chowdhury, Ethan Chang, and Tejal Patwardhan. FrontierScience: Evaluating ai's ability to perform expert-level scientific tasks. *arXiv preprint arXiv:2601.21165*, 2026.

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.

Haotian Ye, Haowei Lin, Jingyi Tang, Yizhen Luo, Caiyin Yang, Chang Su, Rahul Thapa, Rui Yang, Ruihua Liu, Zeyu Li, et al. Evaluation-driven scaling for scientific discovery. *arXiv preprint arXiv:2604.19341*, 2026.

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.

Z.ai. GLM-5.3: Frontier coding with emergent cyber capabilities. Technical report, Z.ai, August 2026a. URL `https://z.ai/blog/glm-5.3`.

Z.ai. GLM-5.3-Flash: Frontier intelligence, flash cost. Technical report, Z.ai, August 2026b. URL `https://z.ai/blog/glm-5.3-flash`.

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. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026a.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient llm agents. *Advances in Neural Information Processing Systems*, 38: 103270–103296, 2026b.

Appendix

A Algorithm Description **13**

B Benchmark Details **14**

B.1 Mathematical Optimization 14

B.2 Systems Optimization 14

B.3 Algorithmic Challenges 15

C Model Pricing **15**

D Prompts **15**

A Algorithm Description

Algorithm 1 describes our algorithm, which mainly contains four phases: (1) cold start, (2) strategy exploration, (3) initial code implementation and strategy ranking, and (4) iterative code refinement.

Algorithm 1 `FrugalEvo`: Cost-aware LLM-guided program evolution

Require: Budget B; task d; evaluator E; packages $\mathcal{P}$; evaluated database $\mathcal{D}$
Require: High-cost LLM $\mathcal{G}_h$; low-cost LLM $\mathcal{G}_l$; opening cap N_0; strategy count K; round cap M
Require: Consecutive-invalid limit F; tolerance ϵ; flags I_0, I_{ref}

$\mathcal{H} \leftarrow \varnothing$; $x^\star \leftarrow \arg\max_{x \in \mathcal{D}_{\text{valid}}} s(x)$
Phase I: Cold start
$(\mathcal{D}, x^\star) \leftarrow \textsc{ColdStartAndCheapClimb}(\mathcal{G}_l, \mathcal{D}, x^\star, N_0, M, F, \epsilon, I_0, B)$ ▷ Cheap search until stalled
$A \leftarrow \textsc{AnalyzeOnce}(\mathcal{G}_h, d, E, \mathcal{P}, B)$ ▷ Cache evaluator and task analysis
while BUDGETAVAILABLE(B) **do** ▷ New strategy portfolio
Phase II: Strategy exploration
  $\bar{x} \leftarrow x^\star$; $\mathcal{C} \leftarrow \textsc{Context}(\mathcal{D}, \bar{x})$ ▷ Build search context
  $(\{S_i\}_{i=1}^k, x_{\text{ref}}) \leftarrow \textsc{Explore}(\mathcal{G}_h, \bar{x}, \mathcal{C}, \mathcal{H}, A, K, I_{\text{ref}}, B)$ ▷ Propose K strategies
  $(\mathcal{D}, x^\star, \bar{x}, \mathcal{C}, \mathcal{H}) \leftarrow \textsc{ProcessReference}(x_{\text{ref}}, \mathcal{D}, x^\star, \bar{x}, \mathcal{C}, \mathcal{H}, \epsilon)$ ▷ Evaluate reference; use as shared parent if improved
Phase III: Initial code implementation and strategy ranking
  $(\mathcal{T}, \mathbf{q}, \mathcal{D}, x^\star) \leftarrow \textsc{ParallelTrials}(\mathcal{G}_l, \{S_i\}_{i=1}^k, \bar{x}, \mathcal{C}, \mathcal{D}, x^\star, B)$ ▷ One trial per strategy; shared parent
  $\mathcal{H} \leftarrow \textsc{RecordTrials}(\mathcal{H}, \mathcal{T})$; $\pi \leftarrow \text{StableArgsort}_i(-q_i)$ ▷ Rank strategies by trial scores
Phase IV: Iterative code refinement
  for $j = 1, \ldots, k$ **do**
    while BUDGETAVAILABLE(B) **do**
      $b \leftarrow s(x^\star)$ ▷ Cache the incumbent score
      $(\mathcal{R}, \mathcal{D}, x^\star) \leftarrow \textsc{SequentialImprove}(\mathcal{G}_l, S_{\pi(j)}, \mathcal{D}, x^\star, M, F, \epsilon, B)$ ▷ Up to M attempts; end round on improvement
      $\Delta \leftarrow s(x^\star) - b$; $\mathcal{H} \leftarrow \textsc{Record}(\mathcal{H}, S_{\pi(j)}, \mathcal{R}, \Delta)$ ▷ Record global improvement
      if $\Delta \leq \epsilon$ **then**
        break ▷ Move to the next strategy
      end if
    end while
  end for
end while
return $x^\star$

B BENCHMARK DETAILS

We evaluate `FrugalEvo` on 20 optimization problems spanning mathematical optimization (5 problems), systems optimization (5 problems), and algorithmic challenges (10 problems).

B.1 MATHEMATICAL OPTIMIZATION

Table 4: **Mathematical optimization tasks.** Summary of optimization tasks used to evaluate `FrugalEvo`, including geometric optimization and signal processing problems.

Problem	Objective	Description
Circle Packing (Square)	$\max \sum_{i=1}^{26} r_i$	Pack 26 circles with variable radii into the unit square, maximizing the sum of their radii subject to non-overlap and containment constraints.
Signal Processing	Maximize composite score	Filter noisy, non-stationary time series, balancing directional reversals, recent and overall tracking errors, false reversals, correlation with the clean signal, noise reduction, and run success across five test signals.
Heilbronn Convex ($n = 13$)	$\max A_{\min}/A_{\text{hull}}$	Arrange 13 points in the plane to maximize the smallest triangle area over all point triples, normalized by the area of the convex hull of all 13 points.
Heilbronn Triangle ($n = 11$)	$\max A_{\min}/A_T$	Place 11 points inside or on a fixed equilateral triangle of side length one, maximizing the smallest triangle area over all point triples, normalized by the container area $A_T = \sqrt{3}/4$.
Min-Max Distance Ratio (3D)	$\max(d_{\min}/d_{\max})^2$	Arrange 14 points in three-dimensional Euclidean space to maximize the squared ratio of the minimum to maximum pairwise Euclidean distance, without a prescribed bounding region.

B.2 SYSTEMS OPTIMIZATION

Table 5: **Systems optimization tasks.** Benchmarks drawn from ADRS-Bench (Cheng et al., 2025), capturing realistic optimization problems from production systems.

Problem	Objective	Description
EPLB (Expert Parallelism Load Balancer)	Maximize composite score	Replicate and place MoE experts across GPUs using recorded expert loads, balancing load across expert replicas while reducing rebalancing and simulation runtime.
PRISM (GPU Model Placement)	Minimize max KV-cache pressure	Assign models to GPUs under memory constraints, minimizing the maximum ratio of SLO-weighted request load to memory remaining for the KV cache.
LLM-SQL (Prefix Caching)	Maximize cache reuse and speed	Optimize column orderings when serializing table rows into LLM prompts, maximizing shared-prefix cache hits while minimizing reordering runtime across five datasets.
Cloudcast (Multi-Cloud Broadcast)	Minimize total cost	Construct partitioned broadcast routes from one source region to multiple destinations, exploiting shared relay paths to reduce egress and instance costs under heterogeneous bandwidths and pricing.
Transaction Scheduling	Minimize makespan	Order transactions with read–write and write–write conflicts on shared data items to minimize total simulated makespan across three workloads.

B.3 ALGORITHMIC CHALLENGES

Table 6: **Algorithmic optimization tasks.** Summary of the 10 algorithmic optimization problems from ALE-Bench-Lite (Imajuku et al., 2026), drawn from AtCoder Heuristic Contests, used to evaluate `FrugalEvo`.

Problem / Category	Count	Description
ahc008 (Territory)	1	Build partitions to maximize humans' reachable space while reducing the number of pets sharing it.
ahc011 (Sliding Puzzle)	1	Slide tiles to form the largest tree; minimize moves once all tiles are connected.
ahc015 (Candy Clustering)	1	Tilt a box to cluster candies by flavor, maximizing the sum of squared cluster sizes.
ahc016 (Graph Classification)	1	Design graphs robust to edge noise and vertex relabelling, balancing classification errors against graph size.
ahc024 (Map Compression)	1	Minimize occupied map cells while preserving district connectivity and all adjacencies, including the exterior.
ahc025 (Weight Balancing)	1	Partition items of unknown weight using limited balance queries; minimize variation in group weights.
ahc026 (Box Stacking)	1	Rearrange box stacks to remove boxes in increasing label order with minimum relocation energy.
ahc027 (Cleaning Route)	1	Find a closed route covering every grid cell to minimize average dirt under repeated traversal.
ahc039 (Fishing Net)	1	Enclose mackerels while avoiding sardines using a simple rectilinear polygon under perimeter and vertex limits.
ahc046 (Skating with Blocks)	1	Visit targets in order with minimum actions using steps, slides, and block placement or removal.

C MODEL PRICING

We present the token prices for the two model configurations used in our experiments: GPT-5.6-Terra with GPT-5.6-Luna, and GLM-5.3 with GLM-5.3-Flash in Table 7.

Table 7: **Model pricing.** Prices were retrieved from the official OpenAI website and OpenRouter on September 4, 2026, for the GPT and GLM models, respectively. All prices are in USD per million tokens.

Model	Input	Output	Cache Read	Cache Write
GPT-5.6-Terra	$2.00	$12.00	$0.20	$2.50
GPT-5.6-Luna	$0.20	$1.20	$0.02	$0.25
GLM-5.3	$1.15	$3.50	$0.10	—
GLM-5.3-Flash	$0.075	$0.25	$0.015	—

D PROMPTS

Below are the prompts used for strategy exploration and code implementation and refinement.

Strategy Exploration Prompt

```
You are the strategy designer for a program-evolution system.
Propose exactly {K} distinct strategies to improve the current
program's evaluated score.

Inputs:
- Task specification and validity constraints
- Current program and evaluated metrics
- Evaluator-aware guidance, when available
- Previous strategies, implementation results, and failures

Each strategy must describe:
1. Mechanism family: a short identifier for the approach.
2. Replacement scope: the function, phase, or block to replace.
3. Compute budget: explicit limits on additional computation.
4. Gain mechanism: the bottleneck addressed or new degree of
   freedom introduced.

Propose different mechanisms, not variations in random seeds,
tolerances, or iteration counts. For K >= 3, include local
refinement, subsystem replacement, and structural exploration.
Use previous results to avoid repeating exhausted approaches.

Make every plan self-contained and immediately implementable.
Preserve public interfaces and hard validity constraints.
Do not rely on unavailable tools, fabricated data, or an
unspecified future optimization step.

Explain why the proposed change can improve the actual scoring
objective. Compare quantities on the same scale. End each plan
with a concise gain check: either a justified score estimate
or a testable hypothesis tied to a specific new mechanism.
A fallback or an accept-only-if-better rule is not itself
evidence of improvement.

Return each strategy in this format:
<STRATEGY_TITLE>[mechanism-family] short title</STRATEGY_TITLE>
<STRATEGY>
Mechanism family: ...
Replacement scope: ...
Compute budget: ...
Gain mechanism: ...
Implementation plan and expected-gain check.
</STRATEGY>

When requested, also return one complete, executable portfolio
reference program in <REFERENCE_IMPLEMENTATION> tags.
It is an independent candidate, not assigned to any strategy.
Keep program code outside the strategy blocks.
```

Code Implementation and Refinement Prompt

```
You are the implementation model in a program-evolution system.
Implement the supplied strategy to improve the current program.

Inputs:
- Task specification and validity constraints
- Current parent program and evaluated metrics
- Strategy to implement
- Previous implementation feedback, when available

Implement the strategy faithfully. Replace the specified
mechanism and remove superseded code when safe. Do not
substitute an unrelated idea, make only a cosmetic change,
or append another complete optimization pipeline.

Ensure that the new mechanism executes on the normal return
path. Preserve public interfaces, permitted edit boundaries,
and hard validity constraints. Keep added computation within
the strategy's budget. A valid fallback may be retained, but
returning only the fallback does not implement the strategy.

If the parent already implements the proposed mechanism,
refine that mechanism rather than adding a duplicate.
Check dimensions, data types, mutability, API calls, and
consistency between function definitions and call sites.

For a pilot implementation, build the smallest executable
```

```
version that exercises the strategy's distinctive mechanism.

For subsequent refinement attempts, use evaluated metrics and
errors to repair failures or revise unsuccessful choices.
Do not repeat failed patches or merely change random seeds.
For timeouts, reduce work or replace the expensive stage while
preserving the strategy's gain mechanism.

Return code in the required output format. For SEARCH/REPLACE
edits, copy each SEARCH block exactly from the current parent
and ensure that it matches uniquely. Previous attempts do not
replace the parent as the edit target.

When repairing a supplied failing program, return a complete,
coherently corrected program rather than a diff against the
failed attempt.
```