Understanding Framework Advantages in LLM-Driven Program Evolution

Anonymous authors
Paper under double-blind review

Abstract

LLM-driven program evolution frameworks are attracting growing attention, yet how their task-specific advantages develop and vary with models and search budgets remains insufficiently understood. We systematically compare four frameworks across five tasks in 1,560 runs under a common 50M-token request-start threshold. Pairwise and within-pool nearest-neighbor distances characterize the dispersion and local structure of task-specific execution profiles, while size-matched random subsets provide a reference for high-quality candidate concentration. We combine these measures with reconstructed budget trajectories and program-level controls to examine search improvements.

Framework advantages are task-specific and broadly consistent across model configurations, although some pairings reverse rankings. Selecting a framework for each configuration does not consistently outperform a task-wide choice on held-out search repeats with the same inputs. Lower response dispersion alone does not consistently identify the best-performing framework. Reconstructed trajectories show that additional budget narrows quality gaps on some tasks and increases attainment of common quality targets.

These findings support framework and model selection based jointly on task fit and available budget, rather than configuration-level rankings or response dispersion alone.

1 Introduction

LLM-driven program evolution combines program generation with executable evaluation to discover algorithms (Romera-Paredes et al., 2024; Liu et al., 2024). By selecting prior programs and reusing evaluation feedback, frameworks organize successive model calls into a search for better solutions (Novikov et al., 2025). In Figure 1, ShinkaEvolve (Lange et al., 2026) forms the circle-packing endpoint-attainment frontier, whereas OpenEvolve (Sharma, 2025) and EoH (Liu et al., 2024) form the bin-packing endpoint-attainment frontier. This view distinguishes attained quality from when each configuration reaches its own endpoint. Earlier attainment can reflect an earlier plateau at lower quality; common-target comparisons appear in Figure 16 in Appendix E. A useful comparison must establish how broadly each advantage holds and which model or budget conditions change it.

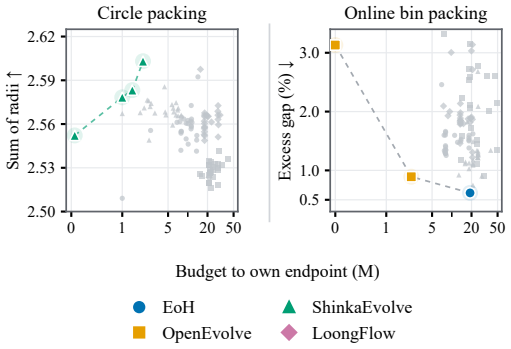

Figure 1: Framework advantages vary by task: ShinkaEvolve forms the circle-packing frontier; OpenEvolve and EoH form the bin-packing frontier. Colored points mark frontier configurations; gray points show the rest.

Comparative studies show task-dependent model and search advantages (Ellis & Castro, 2026) and competitive simple baselines (Gideoni et al., 2026). Trajectory analyses connect optimization to local refinement and conditional novelty (Zhang et al., 2026a); controlled replay examines program edits

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.

(Pelleriti et al., 2026). We ask when a framework advantage persists across model configurations and when it depends on a particular pairing or search budget. Answering requires measuring advantage magnitude and coverage, separating initial quality from subsequent gains, and examining the task operations changed by candidate programs.

To answer these questions, we compare EoH (Liu et al., 2024), OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2026), and LoongFlow (Wan et al., 2025) across five tasks and 26 model and reasoning configurations. Comparisons across tasks, model configurations, and budgets establish where framework advantages persist or change. Decomposing endpoint differences into starting quality and subsequent gains locates those differences at initialization or during search. Program-level diagnostics then characterize code-pattern and response associations, while same-input scoring-function replacements test effects on decisions and quality in selected programs. A leave-one-repeat analysis tests whether configuration-specific framework choices improve subsequent-run performance on the same task inputs.

Prior work measures algorithmic similarity using problem-solving trajectories (Zhang & Lu, 2026). Here, task-specific execution profiles, exploratory source-pattern associations, and same-input program controls characterize response structure, solver decisions, and quality changes. Our contributions are threefold:

1. We compare four frameworks under shared task evaluation, characterizing framework–task fit across model configurations and distinguishing observed rankings from selection gains on held-out search repeats.
2. We decompose endpoint quality into starting quality and search gains, using reconstructed budget trajectories and common-target attainment to examine how framework advantages evolve with budget.
3. We combine execution profiles with program-level controls to examine how program changes affect decisions and outcomes, showing that lower response dispersion alone does not consistently identify the best-performing framework.

2 Related Work

2.1 Comparative Evaluation and Framework Selection

Program-evolution systems differ in generation, selection, and reuse. FunSearch maintains function populations (Romera-Paredes et al., 2024), EoH evolves ideas and code (Liu et al., 2024), and ReEvo uses reflection (Ye et al., 2024). Other approaches include LLaMEA (van Stein & Bäck, 2025), OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2026), LoongFlow (Wan et al., 2025), and CodeEvolve (Assumpção et al., 2025). Successes under different tasks, models, and budgets do not establish which search organization suits a target task.

Comparative studies address task and model dependence (Zhang et al., 2024; Ellis & Castro, 2026), competitive simple search (Gideoni et al., 2026), and budget-dependent rankings (Oved et al., 2026). HeuriGym, ALE-Bench, and BLADE broaden evaluation tools (Chen et al., 2026; Imajuku et al., 2025; van Stein et al., 2025b). We examine ranking coverage, budget dependence, and held-out selection gains.

2.2 Search Processes and Effective Improvement

Process research examines structural inheritance (van Stein et al., 2025a), convergence without selection in a restricted program language (Gurkan et al., 2026), search landscapes (Liu et al., 2025), and execution behavior (van Stein et al., 2026; Zhang & Lu, 2026). Syntactically different programs can exhibit the same problem-solving behavior (Zhang & Lu, 2026). Higher measured diversity does not necessarily coincide with better objective values (Dat et al., 2025). Good Optimizer studies local refinement, conditional novelty, and model mixing (Zhang et al., 2026a); EvoTrace/EvoReplay studies edits through replay and interventions (Pelleriti et al., 2026).

Frontier-Eng studies improvement frequency and magnitude (Chi et al., 2026), and CostAda incorporates cost and remaining budget into search control (Zhang et al., 2026b). We complement these approaches with source-pattern associations, instance-response profiles, and whole-function controls.

3 Evaluation Methodology

We compare framework performance across tasks, models, and budgets, then examine candidate behavior and same-input program controls to connect edits, solver decisions, and final quality (Figure 2).

3.1 Experimental Setup

We compare four frameworks on five tasks across 26 model/reasoning configurations (including MiniMax default mode), with three repeats: 1,560 runs. Requests start below 50M tokens; requests already started may finish and contribute candidates beyond that threshold. All frameworks share task inputs, evaluators, verifiers, scoring rules, and feasibility criteria across configurations and repeats, while retaining native search loops. Leave-one-repeat selection uses the same inputs (Appendix A.2).

The configurations span five model families, model sizes, and reasoning modes; family sensitivity appears in Table 6 in Appendix A.2. Tasks cover complete constructions, embedded solver operators, and online allocation rules (Table 1).

Endpoint and budget analyses use all configurations; Thinking–Non-thinking contrasts use paired modes. IID (Gideoni et al., 2026) and generic feedback search (GFS) provide simple-search references. Protocols and statistics appear in Appendix F; implementations and runtime settings accompany the code. Model names indicate family and total parameters; T/NT denote Thinking/Non-thinking.

Table 1: Tasks, search components, and measurements. Arrows indicate the preferred native-quality direction; execution profiles describe candidate behavior for process analysis.

Task	Search object	Native quality	Execution profile
Circle packing	Complete construction	Sum of radii ↑	Circle centers, radius ratios, and contact fractions
Erdős overlap	Complete construction	C_5 bound ↓	Function responses on fixed probes
Flowshop	Embedded solver operator	Makespan ↓	Fixed-instance makespan responses
Online bin packing	Online allocation rule	Excess gap (%) ↓	Fixed-instance excess-gap responses
TSP	Embedded solver operator	Nonnegative gap (%) ↓	Fixed-instance gap responses

3.2 Performance and Budget Evaluation

Magnitude and conditions of advantage. We average repeats within configurations and weight configurations equally. The task evaluator determines validity and native scores; task-specific transformations orient quality Q so that larger values are better (Appendix F). For conditions A and B on task T, we standardize the quality difference by the task's fixed endpoint-quality sample SD s_T:

$$\Delta Q_T^{\text{std}} = \frac{\overline{Q}_{A,T} - \overline{Q}_{B,T}}{s_T}. \tag{1}$$

For each run, endpoint quality is $Q = I + G$, where I is trajectory-start quality and G is subsequent gain. Thus $\Delta Q = \Delta I + \Delta G$ separates starting-quality differences from later improvement; averaging and uncertainty estimation are detailed in Appendix B.

Budget trajectories and gain accumulation. Budget curves average per-run best-so-far native scores across all configurations on a reconstructed 0–50M-token grid (Figure 5), distinct from completed-request usage. TSP attainment is the fraction reaching $Q \geq 1 - 10^{-12}$, with all runs retained in the denominator.

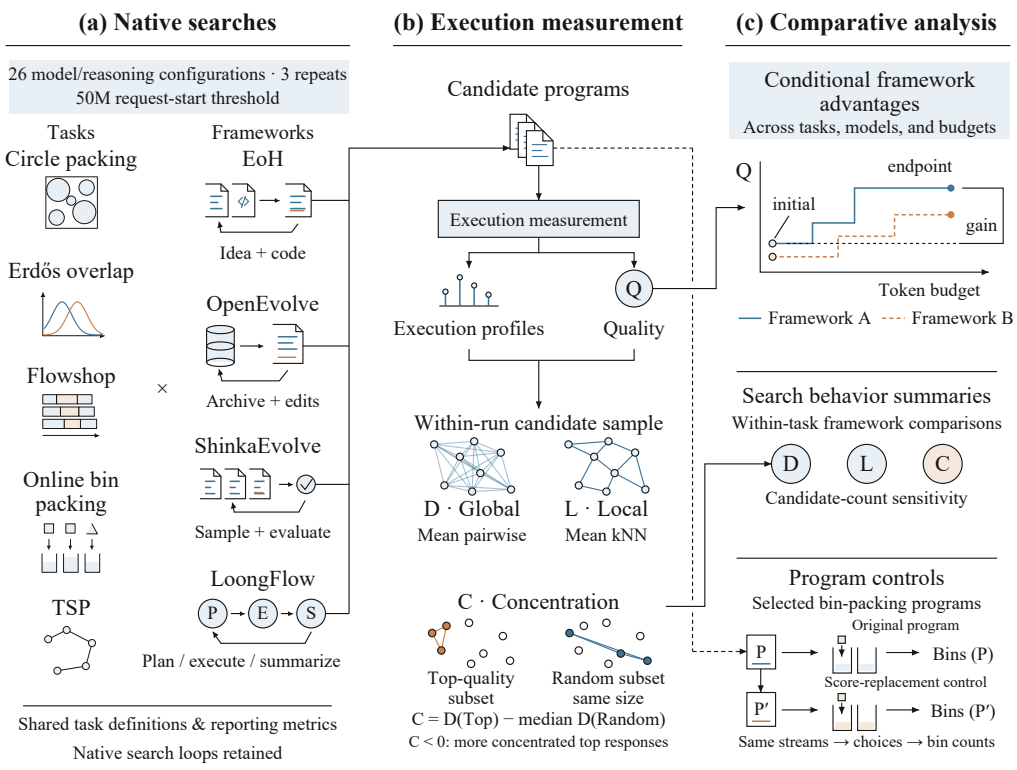

Figure 2: Evaluation overview. (a) Native searches. (b) Quality and execution-profile measurements D, L, C. (c) Model/budget comparisons, search behavior, and same-input program controls. Curves and layouts are schematic; initial quality, gain, and endpoint refer to Framework A.

3.3 Search Process Analysis

Execution profiles and candidate distributions. Profiles encode geometry, function responses, or fixed-instance performance (Table 1), centered by their coordinate mean to describe relative responses. Construction details: Table 15 in Appendix D.1. Let S_r index the $n_r \geq 2$ valid candidate profiles in run r, with $\phi_i \in \mathbb{R}^d$. For task T, let $d_T(i,j) = \|\phi_i - \phi_j\|_2/\sqrt{d}$ be the root-mean-square distance between d-dimensional profiles. Define subset dispersion and mean nearest-neighbor distance by

$$\mathcal{D}_T(S) = \binom{|S|}{2}^{-1} \sum_{i<j\in S} d_T(i,j), \qquad D_r = \mathcal{D}_T(S_r),$$
$$L_r = \frac{1}{n_r k_r} \sum_{i\in S_r} \sum_{j\in \mathrm{kNN}(i;S_r\setminus\{i\})} d_T(i,j), \qquad k_r = \min(5, n_r - 1). \tag{2}$$

Here $S \subseteq S_r$ contains at least two candidates, and kNN selects the k_r nearest candidates within the same pool. Lower D_r and L_r indicate less overall dispersion and denser local behavior, respectively. Comparisons hold task and model configuration fixed.

High-quality response structure. Let H_r contain the best $\max(2, \lceil 0.2 n_r \rceil)$ candidates by quality, including all ties at the cutoff, and let $U_r(m_r)$ be uniform over size-m_r subsets of S_r, where $m_r = |H_r|$. The concentration contrast is

$$C_r = \mathcal{D}_T(H_r) - \mathrm{median}_{S\sim U_r(m_r)}\, \mathcal{D}_T(S). \tag{3}$$

Negative C_r indicates lower dispersion than the size-matched random reference. This auxiliary candidate-pool diagnostic, sampling, and depth sensitivity are detailed in Appendix D.

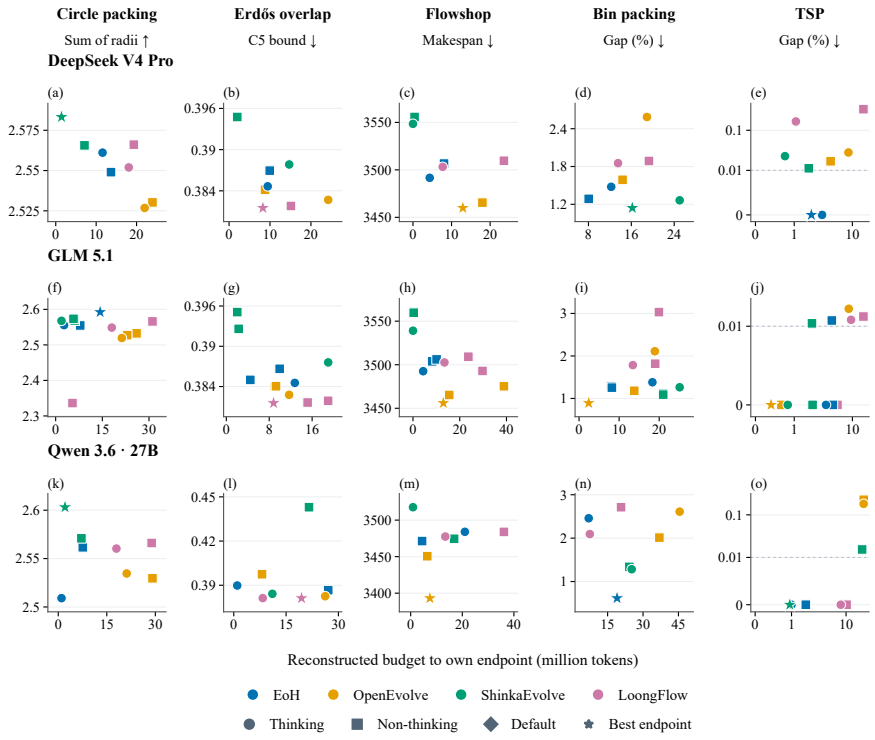

Figure 3: Task outcomes for three models (rows) and five tasks (columns). Colors identify frameworks; circles/squares/diamonds denote Thinking/Non-thinking/default modes. Points pair mean quality with reconstructed budget to each configuration's own endpoint; targets differ. Stars mark best native means. Circle packing is maximized; other scores minimized. Gaps are percentages; axes vary by panel, with symlog for TSP. Remaining models and definitions: Appendix G.

Connecting behavior to outcomes. Same-input bin-packing controls measure whole-function replacement effects on decisions and final bin counts (Appendix C); source tags describe code patterns, and centered profiles describe relative instance responses.

Sensitivity to instance variation. We conducted additional new-instance experiments to examine how framework comparisons depend on assumptions about performance on new inputs. For flowshop and TSP, the experiments vary the persistence of framework effects across additional instances; for bin packing, they also vary stream length and capacity. This analysis makes explicit which assumptions preserve or change the resulting comparisons (Appendix H).

4 Results and Analysis

4.1 Framework Performance Across Tasks

Task-average leaders cover most of the evaluated configurations. Table 2 summarizes endpoint quality and ranking coverage. ShinkaEvolve leads circle packing and online bin packing, LoongFlow leads Erdős, and OpenEvolve leads flowshop. These leaders win outright in most configurations, and all five task-average leaders persist under family-equal weighting and removal of any one model family. For bin packing, ShinkaEvolve leads 20/26 configurations; its mean Q advantage over EoH is 1.889×10^{-3}, with a pointwise 95% bootstrap interval of $[-0.072, 3.610] \times 10^{-3}$ (Table 5 in Appendix A). With TSP ceiling effects, we also examine ties and target attainment: EoH has the highest mean, but its difference from ShinkaEvolve on the Q scale is 2.740×10^{-3} (pointwise 95% bootstrap interval $[-5.274, 9.181] \times 10^{-3}$).

Table 2: Framework performance and winning coverage. Mean task scores across configurations. Bold marks the best mean. Coverage counts unique and tied first places for that task leader.

	Mean task score				Leader coverage (26)	
Task	EoH	OpenEvolve	ShinkaEvolve	LoongFlow	Unique	Tied
Circle packing	2.5565	2.5192	**2.5709**	2.5525	19	0
Erdős overlap	0.3875	0.3850	0.4065	**0.3820**	25	0
Flowshop	3498.80	**3457.62**	3547.92	3505.33	25	0
Online bin packing	1.4855	2.2821	**1.2928**	2.0000	20	0
TSP	**5.182**	30.571	7.839	147.877	7	13

Metrics and preferred directions follow Table 1.
Units: bin packing, %; TSP, 10^{-3}%.

Applying each task-average leader to other tasks gives descriptive mean shortfalls of 0.482–0.892 task-SD, with a maximum of 2.225. Appendix A reports this summary and 30 within-task framework comparisons, each averaged over 26 configurations, with intervals.

Configuration-specific rankings do not consistently yield selection gains. Figure 3 shows three models (others: Appendix G). Across these models, Erdős/flowshop favor LoongFlowOpen EvolveḂin-packing leaders are ShinkaEvolve (DeepSeek V4 Pro), OpenEvolve (GLM5.1), and EoH (Qwen3.6-27B). For Qwen3.6-27B NT, EoH improves on ShinkaEvolve from 1.338% to 0.617% excess gap in all three repeats. Selecting on two repeats and testing on the third with the same inputs does not consistently favor configuration-specific over task-wide selection: ranking reversals do not automatically improve subsequent-run performance (Table 8 in Appendix A.2).

The interface alone does not determine the preferred framework: circle packing/Erdős share whole-program interfaces, and flowshop/TSP share embedded operators, yet each pair favors different frameworks. Task fit must therefore be examined through decisions inside these interfaces (Section 4.4).

Comparison with simple search. IID/GFS use each task's best framework–model configuration and budget; framework results are reused (Table 3). GFS approaches ShinkaEvolve on circle packing; IID ties the best TSP score. The leading framework outperforms both on the other three tasks. Configurations are framework-selected, not independently tuned per method.

Table 3: Auxiliary comparison with simple search under selected configurations. Means (sample SD in parentheses). Arrows indicate the preferred direction; bold marks the best results, including ties. Configurations appear in Appendix A.3.

Method	Circle packing Radii ↑	Erdős C_5 ↓	Flowshop Makespan ↓	Bin packing Gap (%) ↓	TSP Gap (10^{-3}%) ↓
IID	2.5821 (0.0256)	0.3839 (0.0009)	3476.11 (3.40)	2.3678 (1.2522)	**0.000** (0.000)
GFS	2.5980 (0.0659)	0.3821 (0.0008)	3477.33 (4.67)	3.9374 (0.0647)	10.654 (15.454)
EoH	2.5091 (0.0683)	0.3838 (0.0009)	3465.98 (6.39)	**0.6170** (0.0807)	**0.000** (0.000)
OpenEvolve	2.5346 (0.0234)	0.3816 (0.0003)	**3332.33** (0.00)	2.0123 (1.5905)	181.347 (250.128)
ShinkaEvolve	**2.6031** (0.0398)	0.3984 (0.0246)	3474.78 (2.46)	1.3382 (0.2602)	**0.000** (0.000)
LoongFlow	2.5603 (0.0127)	**0.3812** (0.0001)	3501.22 (10.15)	2.7132 (1.0514)	**0.000** (0.000)

4.2 Model Effects Across Frameworks

Average performance changes vary across frameworks and tasks. Multiplicity-adjusted support for framework-specific model effects is limited; these patterns are descriptive. Qwen3.5 122B

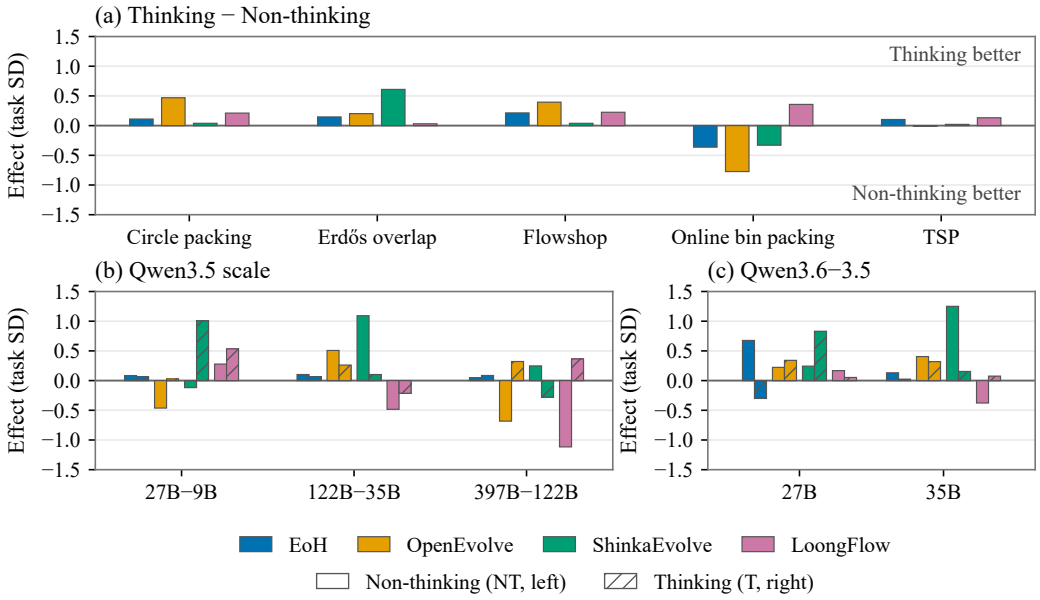

Figure 4: Model-change effects by framework (colors), in task endpoint-SD units. (a) Thinking minus Non-thinking; (b) Qwen3.5 size; (c) Qwen3.6 minus Qwen3.5. Panels (b,c) average five tasks: plain/hatched bars denote NT/T. Positive values favor the first condition. Pooled means and 95% intervals: Figure 9 in Appendix B.

minus 35B (NT) gives five-task mean effects of $+0.102$, -0.484, $+0.507$, and $+1.092$ task-SD for EoH, LoongFlow, OpenEvolve, and ShinkaEvolve. The ShinkaEvolve–LoongFlow difference is 1.576, positive on every task, with 56.3% of its absolute magnitude from Erdős. Tests and normal/skewed, unequal-variance error calibrations appear in Appendix F.3.

Gains from model changes depend on the task and need not yield the best endpoint. Thinking improves flowshop by 0.218 task-SD (95% CI $[0.078, 0.394]$) but reduces bin-packing quality by 0.277 ($[-0.468, -0.087]$); the overall effect is 0.091 ($[-0.036, 0.224]$). Qwen3.5-27B versus 9B gives $+0.408$ ($[0.212, 0.731]$) in T and -0.055 ($[-0.366, 0.198]$) in NT (Figure 4). A larger gain can still leave a framework behind: after the 35B-to-122B change on Erdős, ShinkaEvolve reaches $Q = 0.944751$, compared with LoongFlow at 0.997014. Selection must therefore consider both responsiveness to the model change and the quality ultimately reached.

Changes in initial quality and subsequent gain can offset each other. For Qwen3.5-9B bin packing, Thinking changes OpenEvolve's initial quality by -0.1782 and gain by $+0.1627$ in Q, yielding -0.0155 at the endpoint. ShinkaEvolve instead starts 0.0047 higher and gains 0.0176 less; LoongFlow starts at nearly the same quality and gains 0.0091 more. Greater search gain can therefore recover a poor starting point, while a better start can coexist with less improvement. Model value therefore depends on both the starting solution and continued improvement (Appendix B).

4.3 When Search Gains Accumulate

Framework advantages evolve with the search budget. In Figure 5, LoongFlow catches up after 20M on circle packing while EoH and ShinkaEvolve remain largely stable. On Erdős, Open EvolveLoongFlow reverse ranks from 10M to 20M in the reference reconstruction; the 20M ordering is coordinate-sensitive (Table 18, Appendix E). Budget thus changes when advantages emerge. Budget allocation should follow the quality trajectory on the target task.

More TSP runs reach the common quality threshold as budget increases. At $Q \geq 1 - 10^{-12}$, LoongFlow's attainment count rising from 17/78 at 10M to 32/78 at 20M and 40/78 at 50M. Over 10–50M, OpenEvolve increases from 43/78 to 48/78 and ShinkaEvolve from 59/78 to 60/78, while EoH

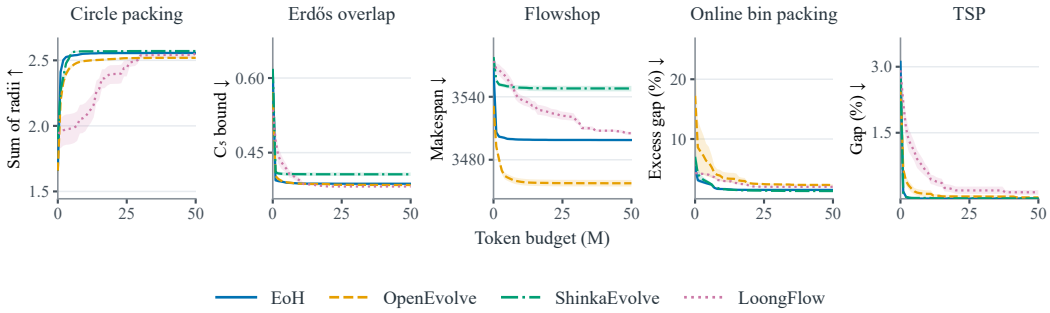

Figure 5: Framework performance on the reconstructed budget grid, averaged over 26 configurations. Colors/styles identify frameworks. Shading: pointwise 95% bootstrap intervals, resampling runs within configurations with trajectories fixed. Circle packing is maximized; other scores minimized.

remains at 70/78. These counts measure evaluator-ceiling attainment, not beyond-ceiling quality or per-instance optimality (Table 19 in Appendix E).

4.4 From Candidate Behavior to Task Improvement

		(a) Behavior & quality				(b) Improvement efficiency					
						per 100 proposals		per 100 requests		per 1M tokens	
		D	L	C	Q	Events	Gain	Events	Gain	Events	Gain
Circle packing	EoH	1.40	1.10	−0.36	0.97	0.05	5.63×10^{-3}	0.02	1.76×10^{-3}	0.07	8.70×10^{-3}
	OpenEvolve	0.30	0.22	−0.15	0.96	0.57	0.01	0.58	0.01	0.35	5.48×10^{-3}
	ShinkaEvolve	0.22	0.16	−0.11	0.98	2.84	0.19	10.01	0.56	3.43	0.19
	LoongFlow	0.38	0.33	−0.10	0.97	3.72	0.76	0.89	0.15	0.53	0.08
Erdős overlap	EoH	0.40	0.28	−0.24	0.98	0.07	9.24×10^{-4}	0.07	1.76×10^{-3}	1.13	9.08×10^{-3}
	OpenEvolve	0.24	0.11	−0.14	0.99	1.92	0.04	1.92	0.04	3.25	0.07
	ShinkaEvolve	0.11	0.04	−0.07	0.92	1.78	0.13	1.78	0.13	2.42	0.49
	LoongFlow	0.35	0.30	−0.05	1.00	3.98	0.25	3.44	0.18	9.40	0.48
Flowshop	EoH	20.78	9.28	−9.48	2.86×10^{-4}	0.54	5.64×10^{-7}	0.68	1.20×10^{-6}	1.07	2.08×10^{-6}
	OpenEvolve	10.76	5.05	−5.32	2.88×10^{-4}	0.16	1.12×10^{-6}	0.16	1.12×10^{-6}	0.21	1.15×10^{-6}
	ShinkaEvolve	8.57	3.65	−4.99	2.82×10^{-4}	1.01	1.51×10^{-5}	1.01	1.51×10^{-5}	0.92	9.00×10^{-6}
	LoongFlow	18.51	8.90	−6.97	2.85×10^{-4}	2.00	2.65×10^{-6}	0.55	3.45×10^{-7}	0.85	4.75×10^{-7}
Online bin packing	EoH	4.18×10^{-3}	2.23×10^{-3}	-1.87×10^{-3}	0.98	1.37	6.81×10^{-3}	1.32	3.52×10^{-3}	2.32	9.40×10^{-3}
	OpenEvolve	2.04×10^{-3}	8.00×10^{-4}	-1.49×10^{-3}	0.98	0.32	1.24×10^{-3}	0.32	1.24×10^{-3}	0.28	2.55×10^{-3}
	ShinkaEvolve	1.91×10^{-3}	9.61×10^{-4}	-1.14×10^{-3}	0.99	0.76	2.87×10^{-3}	0.76	2.87×10^{-3}	0.40	1.44×10^{-3}
	LoongFlow	4.98×10^{-3}	1.93×10^{-3}	-4.78×10^{-3}	0.98	0.87	4.64×10^{-3}	0.36	2.99×10^{-3}	0.38	3.06×10^{-3}
TSP	EoH	0.90	0.29	−0.67	0.99	0.26	0.05	0.38	0.05	0.63	0.12
	OpenEvolve	0.26	0.11	−0.24	0.97	0.55	0.02	0.55	0.02	0.63	0.03
	ShinkaEvolve	0.40	0.17	−0.38	1.00	0.55	0.05	0.55	0.05	0.18	0.02
	LoongFlow	0.93	0.34	−0.54	0.90	4.67	0.98	0.33	0.06	0.32	0.06

Low — High

Figure 6: Task–framework summaries of (a) search behavior and endpoint quality and (b) improvement efficiency.

Lower response dispersion alone does not identify quality leaders. ShinkaEvolve has the least dispersion on circle packing, Erdős, flowshop, and bin packing, but leads only circle packing and bin packing among these four (Figure 6). For instance-based profiles, concentration of centered

response patterns alone is insufficient to assess final quality and should be interpreted alongside overall performance and the program's decision rules.

Code-pattern associations and program-edit controls link program structure to outcomes. Across seven code-pattern categories, flowshop ranking/index-selection tags show a within-run validity advantage of 34.1 percentage points; their valid-quality difference has a marginal 95% interval spanning zero. Four TSP edit controls yield three ties and one small gain. Together, the analyses identify code patterns associated with candidate performance and test the effects of specific program changes (Appendix C).

A bin-packing mechanistic case study connects scoring rules, decisions, and outcomes. With driver, streams, feasibility filtering, and tie rule fixed, 11/18 programs outperform whole-function best-fit replacements; seven tie. Positive rescaling of best-fit scores leaves all 25,000 placements unchanged. Together, these controls distinguish changes in code from changes in the executed allocation rule. Across five streams, OpenEvolveLoongFlow stage programs change 1.252%/47.276% of decisions and save 23/33 bins: far more changes yield only modestly greater savings. Another OpenEvolve program saves 208 bins; its returned vector can select a different bin from its internal best index. Replacement measures score and return logic jointly; tracing explains the displayed decision, not total savings. Two programs move from two and four extra bins to final savings of 33 and 30. These cases trace how returned scores affect allocation decisions and the states encountered by later items; decision-change frequency alone does not account for their final savings.

Source-pattern associations and selected whole-function effects do not quantify contributions to between-framework performance gaps.

5 Conclusion

Framework advantages in LLM-driven program evolution depend on the target task and how quality develops under the available budget. Our comparison shows that task-average leaders retain broad coverage across model configurations, while configuration-specific framework choices do not consistently improve performance on held-out search repeats with the same inputs. Model changes affect frameworks differently in the observed comparisons, and larger search gains can compensate for a poorer starting solution. Reconstructed trajectories further show that additional search narrows some quality gaps and increases common-target attainment on TSP. Together, these findings make task fit and budget-dependent progress central to joint framework and model selection.

The process analysis connects this selection problem to the programs produced during search. Lower response dispersion alone does not consistently identify the quality leader. Source-pattern comparisons characterize validity and quality associations, while whole-function replacements and execution traces in bin packing connect changes in scoring rules to allocation decisions and final outcomes. This combination distinguishes descriptions of candidate structure from measured effects of selected program changes. It motivates studying which task-specific modifications a framework generates, retains, and reuses, alongside the quality those modifications ultimately achieve.

The evaluation covers five tasks, selected model configurations, and fixed task inputs; its held-out-repeat analysis measures selection across searches, while the additional-instance scenarios remain synthetic. Budget comparisons rely on reconstructed coordinates, and their ordering can depend on the reconstruction. The simple-search comparison uses configurations selected from the main experiment and reuses the corresponding framework runs. Source-pattern associations and selected whole-function controls characterize program-level relationships and effects, but do not isolate the contributions of framework components to performance gaps. These boundaries define the scope of the comparisons and the evidence needed to extend them.

Future work should test joint framework and model selection on new task instances, tasks, feedback conditions, and budgets, with independently selected configurations and fresh evaluation runs. Targeted interventions on selection and reuse can test whether these mechanisms favor productive task-specific modifications. A further direction is adaptive search control: using execution signals and observed improvement to decide when to broaden search, restructure a solution, or continue refinement within a fixed total token budget.

AI USE STATEMENT

We used generative AI tools to assist with manuscript writing and refinement, including proofreading, language editing, organization, technical exposition, and LaTeX preparation. We also used AI-assisted retrieval and discovery to identify relevant literature and potential references. AI assistance supported research ideation, analysis-code preparation, checking mathematical explanations, and discussing the interpretation of experimental results. As part of the experiments, LLMs generated and iteratively refined candidate programs within the evaluated frameworks; task-specific evaluators and verifiers assessed their quality and feasibility, as described in the methodology and appendices. The authors retain responsibility for the research design, data analysis, scientific claims, citation accuracy, and final manuscript.

REPRODUCIBILITY STATEMENT

Section 3 defines the comparison units and measurements. The appendices provide full outcomes, model-effect decomposition, program controls with source excerpts, candidate-pool definitions and sensitivity, gain progress and resource efficiency, and complete model panels. The manuscript source package contains the figures and inputs needed for compilation. The experiment code package provides framework adapters, task evaluators, runtime configuration files, IID/GFS implementations, and commands for checking the supplied numerical results.

REFERENCES

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: an open source evolutionary coding agent for algorithmic discovery and optimization, 2025. URL https://arxiv.org/abs/2510.14150v6. arXiv preprint, version 6, revised 27 August 2026.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.07972.

Yizhe Chi, Deyao Hong, Dapeng Jiang, Tianwei Luo, Kaisen Yang, Boshi Zhang, Zhe Cao, Xiaoyan Fan, Bingxiang He, Han Hao, Weiyang Jin, Dianqiao Lei, Qingle Liu, Houde Qian, Bowen Wang, Situ Wang, Youjie Zheng, Yifan Zhou, Calvin Xiao, Eren Cai, and Qinhuai Na. Frontier-Eng: Benchmarking self-evolving agents on real-world engineering tasks with generative optimization, 2026. URL https://arxiv.org/abs/2604.12290v2. arXiv preprint, version 2.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. HSEvo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using LLMs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):26931–26938, 2025. doi: 10.1609/aaai.v39i25.34898. URL https://ojs.aaai.org/index.php/AAAI/article/view/34898.

Marquita Ellis and Paul Castro. Don't gamble, GAMBLe: An analytical framework for AI-driven research systems, 2026. URL https://arxiv.org/abs/2606.02863. arXiv preprint.

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution. In *ICLR 2026 Workshop on Recursive Self-Improvement*, 2026. URL https://openreview.net/forum?id=QSWFqDcveB.

Can Gurkan, Forrest Stonedahl, and Uri Wilensky. Mutation without variation: Convergence dynamics in LLM-driven program evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1392–1407. Association for Computing Machinery, 2026. doi: 10.1145/3795101.3814735. URL https://doi.org/10.1145/3795101.3814735. GECCO Workshop on LLMs for/with Evolutionary Computation.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2506.09050.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, pp. 74026–74078, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7886b9bafe76c52fd568db10ff9772df-Abstract-Conference.html.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Fei Liu, Qingfu Zhang, Jialong Shi, Xialiang Tong, Kun Mao, and Mingxuan Yuan. Fitness landscape of large language model-assisted automated algorithm search, 2025. URL https://arxiv.org/abs/2504.19636v3. arXiv preprint, version 3.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131. White paper; arXiv preprint.

Tal Oved, Roi Pony, Oshri Naparstek, and Udi Barzelay. Evolution or illusion? Rethinking evaluation in LLM evolutionary search, 2026. URL https://arxiv.org/abs/2609.19799v1. Preprint, version 1.

Nico Pelleriti, Sree Harsha Nelaturu, Zhanke Zhou, Zongze Li, Max Zimmer, Bo Han, and Sebastian Pokutta. What do evolutionary coding agents evolve?, 2026. URL https://arxiv.org/abs/2605.20086. arXiv preprint; EvoTrace and EvoReplay are artifacts introduced in this work.

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. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

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

Niki van Stein and Thomas Bäck. LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793. URL https://doi.org/10.1109/TEVC.2024.3497793.

Niki van Stein, Anna V. Kononova, Lars Kotthoff, and Thomas Bäck. Code evolution graphs: Understanding large language model driven design of algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 943–951. Association for Computing Machinery, 2025a. doi: 10.1145/3712256.3726328. URL https://doi.org/10.1145/3712256.3726328.

Niki van Stein, Anna V. Kononova, Haoran Yin, and Thomas Bäck. BLADE: Benchmark suite for LLM-driven automated design and evolution of iterative optimisation heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2336–2344. Association for Computing Machinery, 2025b. doi: 10.1145/3712255.3734347. URL https://doi.org/10.1145/3712255.3734347.

Niki van Stein, Haoran Yin, Anna V. Kononova, Thomas Bäck, and Gabriela Ochoa. Behaviour space analysis of LLM-driven meta-heuristic discovery. In *Computational Intelligence*, volume 2828 of *Communications in Computer and Information Science*, pp. 367–385. Springer, 2026. doi: 10.1007/978-3-032-15635-8_23. URL https://doi.org/10.1007/978-3-032-15635-8_23.

Chunhui Wan, Xunan Dai, Zhuo Wang, Minglei Li, Yanpeng Wang, Yinan Mao, Yu Lan, and Zhiwen Xiao. LoongFlow: Directed evolutionary search via a cognitive plan-execute-summarize paradigm, 2025. URL https://arxiv.org/abs/2512.24077. arXiv preprint.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608, 2024. doi: 10.52202/079017-1381. URL https://papers.nips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Rui Zhang and Zhichao Lu. Rethinking code similarity for automated algorithm design with LLMs. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HIUqeO9OOr.

Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Understanding the importance of evolutionary search in automated heuristic design with large language models. In *Parallel Problem Solving from Nature – PPSN XVIII*, volume 15149 of *Lecture Notes in Computer Science*, pp. 185–202. Springer, 2024. doi: 10.1007/978-3-031-70068-2_12. URL https://doi.org/10.1007/978-3-031-70068-2_12.

Xinhao Zhang, Xi Chen, François Portet, and Maxime Peyrard. What makes an LLM a good optimizer? A trajectory analysis of LLM-guided evolutionary search. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24987–25011. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.findings-acl.1252. URL https://aclanthology.org/2026.findings-acl.1252/.

Yansen Zhang, Yilu Liu, Tianyu Liu, Jiamin Chen, Xiaokun Zhang, Kai Xie, Xue Liu, Chen Ma, and Yiyan Qi. Budget-aware LLM discovery via cost-calibrated frontier utility, 2026b. URL https://arxiv.org/abs/2607.26828v1. Preprint, version 1.

A Complete Outcomes and Framework Selection

Endpoint comparisons cover 26 model/reasoning configurations, four frameworks, and five tasks, with three independent repeats per combination (1,560 runs); MiniMax uses its default mode. Appendix F gives the evaluation and statistical methods.

Native scores make framework differences interpretable in task units. Table 4 combines mean performance and repeat variation. Means and SDs are first computed within each configuration and then averaged over 26 configurations. Native transformations are applied to individual runs before aggregation.

Table 4: Native outcomes and repeat variation. Means and within-configuration repeat SDs are averaged across configurations. Flowshop Q is shown in units of 10^{-4}; SDs use native units.

Framework	Mean native score	Mean repeat SD	Mean Q
Circle packing: Sum of radii ↑			
EoH	2.5565	0.0161	0.9699
OpenEvolve	2.5192	0.0179	0.9557
ShinkaEvolve	2.5709	0.0153	0.9754
LoongFlow	2.5525	0.0257	0.9684
Erdős overlap: C_5 bound ↓			
EoH	0.3875	0.0022	0.9831
OpenEvolve	0.3850	0.0022	0.9897
ShinkaEvolve	0.4065	0.0113	0.9430
LoongFlow	0.3820	0.0004	0.9973
Flowshop: Makespan ↓			
EoH	3498.80	5.91	2.8574
OpenEvolve	3457.62	9.08	2.8916
ShinkaEvolve	3547.92	9.07	2.8180
LoongFlow	3505.33	6.25	2.8520
Online bin packing: Excess gap (%) ↓			
EoH	1.4855	0.1654	0.9854
OpenEvolve	2.2821	0.5071	0.9778
ShinkaEvolve	1.2928	0.2371	0.9873
LoongFlow	2.0000	0.3403	0.9804
TSP: Nonnegative gap (%) ↓			
EoH	0.005182	0.006112	0.9952
OpenEvolve	0.030571	0.041080	0.9754
ShinkaEvolve	0.007839	0.010215	0.9924
LoongFlow	0.147877	0.163127	0.9166

A.1 Which differences persist across configurations?

Table 5 provides six pairwise comparisons among the four frameworks for each of the five tasks, giving 30 contrasts on the oriented quality scale. Each contrast averages equally over all 26 evaluated configurations. They are not an independent random sample from a population of models. We resample configurations and then independently resample three runs within each selected configuration and framework (50,000 replicates). The reported 95% percentile intervals describe each contrast under this empirical resampling scheme. Configuration resampling varies the composition of the observed condition set, while run resampling varies the repeated-search outcomes. It does not model dependence among related configurations or provide population-level coverage for unseen models. Equal-family weighting and leave-one-family-out analyses separately examine composition sensitivity. Differences and intervals use task-specific display multipliers given in the table headings; its native makespan remains in Table 4.

Table 5: Complete framework contrasts. All 26 configurations; differences are $Q_A - Q_B$. Intervals are pointwise 95% hierarchical-bootstrap percentile intervals. Display units: differences and interval limits use the multiplier in each task heading.

Framework A	Framework B	ΔQ	95% interval
Circle packing ($\times 10^{-3}$ in Q units)			
EoH	OpenEvolve	+14.157	[+8.4801, +22.635]
EoH	ShinkaEvolve	-5.4694	[-9.2981, -2.1319]
EoH	LoongFlow	+1.5165	[-4.3991, +11.141]
OpenEvolve	ShinkaEvolve	-19.627	[-28.148, -14.149]
OpenEvolve	LoongFlow	-12.641	[-23.019, -1.4567]
ShinkaEvolve	LoongFlow	+6.986	[+1.4046, +16.449]
Erdős overlap ($\times 10^{-3}$ in Q units)			
EoH	OpenEvolve	-6.5666	[-9.7536, -2.0205]
EoH	ShinkaEvolve	+40.161	[+19.964, +65.125]
EoH	LoongFlow	-14.136	[-17.625, -11.27]
OpenEvolve	ShinkaEvolve	+46.727	[+26.368, +72.005]
OpenEvolve	LoongFlow	-7.5694	[-12.142, -4.7792]
ShinkaEvolve	LoongFlow	-54.297	[-80.744, -32.746]
Flowshop ($\times 10^{-6}$ in Q units)			
EoH	OpenEvolve	-3.4261	[-4.3383, -2.6907]
EoH	ShinkaEvolve	+3.9411	[+3.3562, +4.4656]
EoH	LoongFlow	+0.53343	[+0.16497, +0.9374]
OpenEvolve	ShinkaEvolve	+7.3671	[+6.6292, +8.0904]
OpenEvolve	LoongFlow	+3.9595	[+3.0253, +5.0882]
ShinkaEvolve	LoongFlow	-3.4076	[-4.1511, -2.5827]
Bin packing ($\times 10^{-3}$ in Q units)			
EoH	OpenEvolve	+7.5786	[+4.3179, +10.859]
EoH	ShinkaEvolve	-1.8892	[-3.6099, +0.071852]
EoH	LoongFlow	+4.9452	[+2.7061, +7.5352]
OpenEvolve	ShinkaEvolve	-9.4679	[-12.355, -6.636]
OpenEvolve	LoongFlow	-2.6335	[-7.2309, +2.0583]
ShinkaEvolve	LoongFlow	+6.8344	[+3.9928, +9.6268]
TSP ($\times 10^{-3}$ in Q units)			
EoH	OpenEvolve	+19.804	[+5.9635, +40.204]
EoH	ShinkaEvolve	+2.7399	[-5.2742, +9.1809]
EoH	LoongFlow	+78.529	[+38.109, +126.68]
OpenEvolve	ShinkaEvolve	-17.064	[-37.211, -3.4985]
OpenEvolve	LoongFlow	+58.725	[+10.753, +111.76]
ShinkaEvolve	LoongFlow	+75.789	[+33.294, +127.44]

A.2 Selection across model configurations

These supplementary analyses examine whether the observed framework rankings persist across model configurations and independent search repeats under the fixed evaluation protocol. Winning coverage and removal of individual model families test whether task leaders depend on a narrow set of configurations (Table 6). Specific model pairings can still reverse the task-average ranking (Table 7). Leave-one-repeat selection measures the consequence of choosing a framework from the other two repeats (Table 8).

Table 6: Winning coverage and model-family sensitivity.

Task	Task-mean leader	Unique first-place configs.	Tied-first configs.	Equal-family-weighted leader	Leader retained after leaving one family
Circle	ShinkaEvolve	19/26	0/26	ShinkaEvolve	5/5
Erdős	LoongFlow	25/26	0/26	LoongFlow	5/5
Flowshop	OpenEvolve	25/26	0/26	OpenEvolve	5/5
Online bin packing	ShinkaEvolve	20/26	0/26	ShinkaEvolve	5/5
TSP	EoH	7/26	13/26	EoH	5/5

Table 7: Configuration-specific departures from the task-average leader. All configurations whose mean exceeds the task-average leader. Differences are configuration-best minus task-leading framework; standardization uses the task endpoint SD across all runs. Display units: ΔQ uses the multiplier in each task heading; task-SD values are unchanged.

Config.	Best framework	ΔQ	Δ/task-SD
Circle packing ($\times 10^{-3}$ in Q units)			
C03	LoongFlow	0.171839	0.010
C07	EoH	7.21045	0.400
C10	LoongFlow	5.06132	0.281
C16	LoongFlow	4.52854	0.251
C17	EoH	0.61598	0.034
C19	LoongFlow	4.88709	0.271
C21	EoH	5.47861	0.304
Erdős overlap ($\times 10^{-3}$ in Q units)			
C02	OpenEvolve	0.770133	0.019
Flowshop ($\times 10^{-6}$ in Q units)			
C02	LoongFlow	1.45967	0.441
Online bin packing ($\times 10^{-3}$ in Q units)			
C02	OpenEvolve	0.213462	0.028
C05	OpenEvolve	2.25037	0.295
C16	LoongFlow	5.80178	0.761
C19	LoongFlow	11.7165	1.536
C21	EoH	4.37568	0.574
C22	EoH	7.06885	0.927
TSP ($\times 10^{-3}$ in Q units)			
C05	LoongFlow	11.638	0.131
C05	OpenEvolve	11.638	0.131
C26	ShinkaEvolve	0.110429	0.001
C14	ShinkaEvolve	5.90105	0.066
C15	ShinkaEvolve	6.73325	0.076
C16	ShinkaEvolve	3.38831	0.038
C18	ShinkaEvolve	64.6826	0.727

Table 8: Framework selection with a held-out search repeat. Two search repeats select frameworks and the third evaluates them on the same task inputs; training ties receive equal weight. Differences are configuration-specific minus task-wide selection, standardized by training-run quality SD; positive values favor configuration-specific selection.

Task	Mean difference	Held-out r1	Held-out r2	Held-out r3
Circle packing	-0.087	-0.086	-0.054	-0.121

Table 8 (continued)

Task	Mean difference	Held-out r1	Held-out r2	Held-out r3
Erdős overlap	+0.001	-0.001	+0.002	+0.001
Flowshop	-0.012	-0.011	+0.000	-0.024
Online bin packing	+0.046	+0.009	-0.030	+0.159
TSP	-0.094	-0.119	-0.040	-0.124

For each task, we select its task-average leading framework and compare that framework's existing results on the other four tasks with their respective task-average leaders. The mean quality losses range from 0.482 to 0.892 task-SD, with a maximum individual loss of 2.225. Each loss is divided by the target task endpoint SD across all runs. This retrospective summary quantifies the cost of reusing a task leader within the evaluated task set; it is not a held-out transfer or deployment test.

A.3 Fixed-configuration simple searches

The auxiliary comparison evaluates one fixed model/reasoning configuration per task. It contains 30 additional IID/GFS runs and 60 reused framework runs, with three repeats per method and task. Table 9 defines these five settings; Table 10 reports their native outcomes. Both baselines generate one complete candidate program per proposal and evaluate it with the task evaluator. The IID/GFS budget protocol counts input and output tokens from successfully completed LLM requests, including reasoning tokens within the output total without double counting. Failed API or transport requests do not count toward the budget, even if an error response contains usage metadata. A completed generation remains chargeable when its candidate subsequently fails parsing, compilation, or evaluation. IID requests receive the fixed task specification, interface constraints, shared initial program when supplied, proposal number, and token-budget status. They receive no previously generated candidates or evaluation feedback. GFS additionally receives the best-so-far and latest candidate programs, their quality, validity, and error information, and a chronological summary of earlier evaluations. This summary records candidate identifiers, cumulative token use, validity, quality, and whether each candidate improved the run best; it does not contain every earlier program. When the input limit is reached, the oldest summary entries are removed first, while the task specification and best-so-far and latest candidate information are retained. The GFS prompt asks the model to improve an existing solution or redesign it using this feedback. The accompanying code package includes the method prompts, input-construction implementation, and per-run configurations.

Table 9: Configurations for simple search.

Task	Fixed model and reasoning mode
Circle packing	Qwen3.6-27B Thinking
Erdős overlap	MiniMax-M2.7 default
Flowshop	Qwen3.5-397B-A17B Thinking
Online bin packing	Qwen3.6-27B Non-thinking
TSP	Qwen3.6-27B Thinking

Table 10: Simple-search repeat statistics. Native task scores.

Method	Mean score	Sample SD	Minimum	Maximum
Circle packing: Sum of radii ↑				
IID	2.5821	0.0256	2.5671	2.6116
GFS	2.5980	0.0659	2.5219	2.6360
EoH	2.5091	0.0683	2.4468	2.5821
LoongFlow	2.5603	0.0127	2.5486	2.5739
OpenEvolve	2.5346	0.0234	2.5191	2.5615
ShinkaEvolve	2.6031	0.0398	2.5575	2.6307

Continued on next page

Table 10 (continued)

Method	Mean score	Sample SD	Minimum	Maximum
Erdős overlap: C_5 bound ↓				
IID	0.3839	0.0009	0.3832	0.3849
GFS	0.3821	0.0008	0.3812	0.3826
EoH	0.3838	0.0009	0.3828	0.3844
LoongFlow	0.3812	0.0001	0.3811	0.3814
OpenEvolve	0.3816	0.0003	0.3814	0.3820
ShinkaEvolve	0.3984	0.0246	0.3816	0.4266
Flowshop: Makespan ↓				
IID	3476.11	3.40	3473.67	3480.00
GFS	3477.33	4.67	3472.67	3482.00
EoH	3465.98	6.39	3459.82	3472.58
LoongFlow	3501.22	10.15	3489.67	3508.67
OpenEvolve	3332.33	0.00	3332.33	3332.33
ShinkaEvolve	3474.78	2.46	3472.00	3476.67
Online bin packing: Excess gap (%) ↓				
IID	2.3678	1.2522	1.4991	3.8032
GFS	3.9374	0.0647	3.8636	3.9843
EoH	0.6170	0.0807	0.5330	0.6940
LoongFlow	2.7132	1.0514	1.6601	3.7630
OpenEvolve	2.0123	1.5905	0.7647	3.8032
ShinkaEvolve	1.3382	0.2602	1.1068	1.6199
TSP: Nonnegative gap (%) ↓				
IID	0.000000	0.000000	0.000000	0.000000
GFS	0.010654	0.015454	0.000000	0.028378
EoH	0.000000	0.000000	0.000000	0.000000
LoongFlow	0.000000	0.000000	0.000000	0.000000
OpenEvolve	0.181347	0.250128	0.000000	0.466697
ShinkaEvolve	0.000000	0.000000	0.000000	0.000000

Figure 7 shows how task-level leadership coexists with configuration-specific departures and shared TSP ranks. Mean ranks should be interpreted alongside difference magnitudes and held-out-repeat selection results.

B Model Effects: Starting Quality and Search Gain

A model change can alter both starting quality and subsequent improvement. For each run, I is the starting quality at the zero-token checkpoint of its analysis trajectory, Q is its endpoint quality, and $G = Q - I$ is the gain relative to that starting value. These quantities are computed per run before averaging within conditions. For two conditions A and B,

$$\Delta Q = Q_A - Q_B = \Delta I + \Delta G, \tag{4}$$

where $\Delta I = I_A - I_B$ and $\Delta G = G_A - G_B$ are the between-condition differences in initial quality and subsequent gain. The differences use the same task endpoint SD when standardized. The 220 task–model–framework Thinking contrasts each compare three independent runs per mode. Table 11 reports the pooled decomposition; Figure 9(a) reports endpoint effects by task. Main-figure bars carry framework-specific pointwise 95% intervals. Thinking intervals resample the 11 configuration pairs (20,000 draws); size/generation intervals resample the five tasks (5,000 draws). Pooled intervals concern across-framework means and do not substitute for framework-specific uncertainty. These intervals are distinct from the multiplicity-adjusted interaction tests in Appendix F.

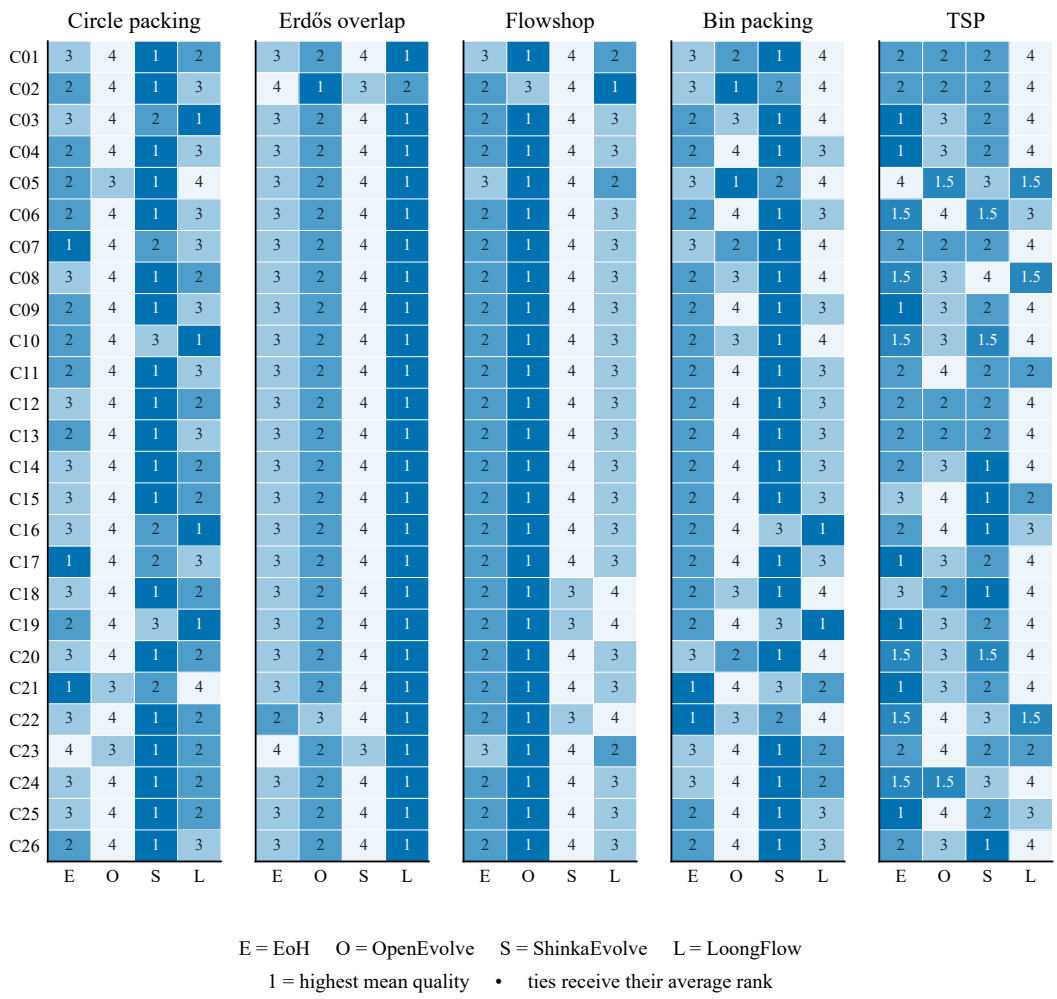

Figure 7: Framework ranks across every configuration. Each cell ranks mean endpoint quality within each task and configuration; lower rank is better. Ranks use an absolute tolerance of 10^{-12} on unrounded mean Q, with average ranks for ties. Configuration IDs match those in the accompanying data.

Table 11: Thinking-effect decomposition. T minus NT in task-SD units; intervals use joint resampling.

Thinking component	Effect (task-SD; T−NT)	Joint-bootstrap 95% interval
Initial quality	+0.386	[-0.550, 1.279]
Search gain	-0.295	[-1.163, 0.614]
Endpoint	+0.091	[-0.036, 0.224]

Figure 8 separates starting-quality and gain changes: opposite-sign coordinates indicate offsetting contributions, and their sum determines the endpoint change. All model-effect tables, pooled summaries, and the initial–gain scatter use the fixed sample SD of endpoint quality across all 312 runs per task (26 configurations, four frameworks, and three repeats; $n-1$ denominator). Thinking contrasts retain the 11 model configurations with both modes.

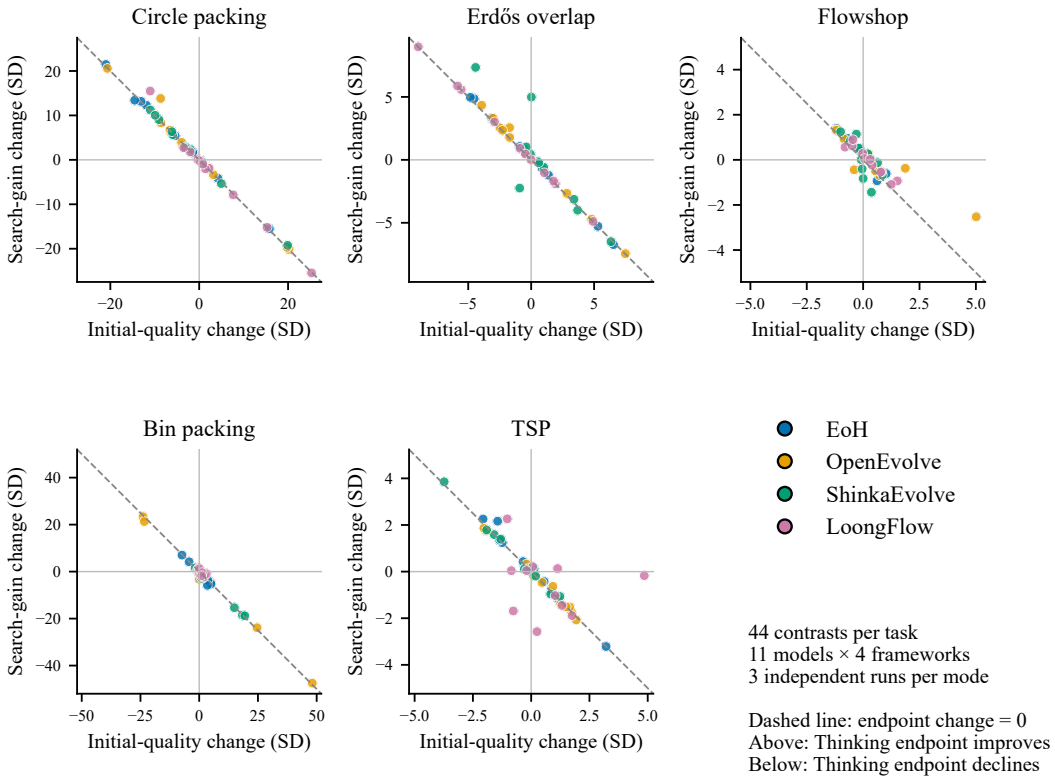

Figure 8: Starting quality and later gain. Each point is a Thinking-minus-Non-thinking contrast; both axes use task endpoint-quality SD units. The dashed line is zero endpoint change: initial and gain changes sum to the endpoint difference. Axis ranges vary by task.

B.1 Named model-size and generation comparisons

Table 24 in Appendix F.3 defines all named model changes. Table 12 retains the complete task-by-framework comparison for Qwen3.5 122B versus 35B in Non-thinking mode. Cross-task intervals treat the five tasks as clusters.

Table 12: Qwen3.5-122B versus 35B by task and framework. 122B minus 35B, Non-thinking, task-SD units.

Task	EoH	LoongFlow	OpenEvolve	ShinkaEvolve
Circle packing	-0.2204	-0.6784	-0.0656	-0.3643
Erdős overlap	+0.0043	-0.0115	-0.0413	+4.4256
Flowshop	+0.0780	+0.0666	+0.1440	+0.3495
Online bin packing	+0.4550	-0.8255	+2.2011	+0.8941
TSP	+0.1941	-0.9697	+0.2984	+0.1560
Equal-task mean	+0.1022	-0.4837	+0.5073	+1.0922

C Source Patterns, Program Controls and Task Outcomes

A bin-packing mechanistic case study tests how whole-function replacements affect bin selection and final bin counts. We compare each program with a best-fit control that replaces its complete scoring function on five fixed streams of 5,000 items, with bin capacity 100. The driver, streams, feasible-bin filtering and tie rule are held fixed. The comparison includes 18 programs from 18 runs and 11 intermediate versions from those runs, giving 29 program versions and 145 paired program–stream evaluations. Intermediate versions share their source runs, and the same five streams are reused across programs; these evaluations are not independent program samples. The contrasts mea-

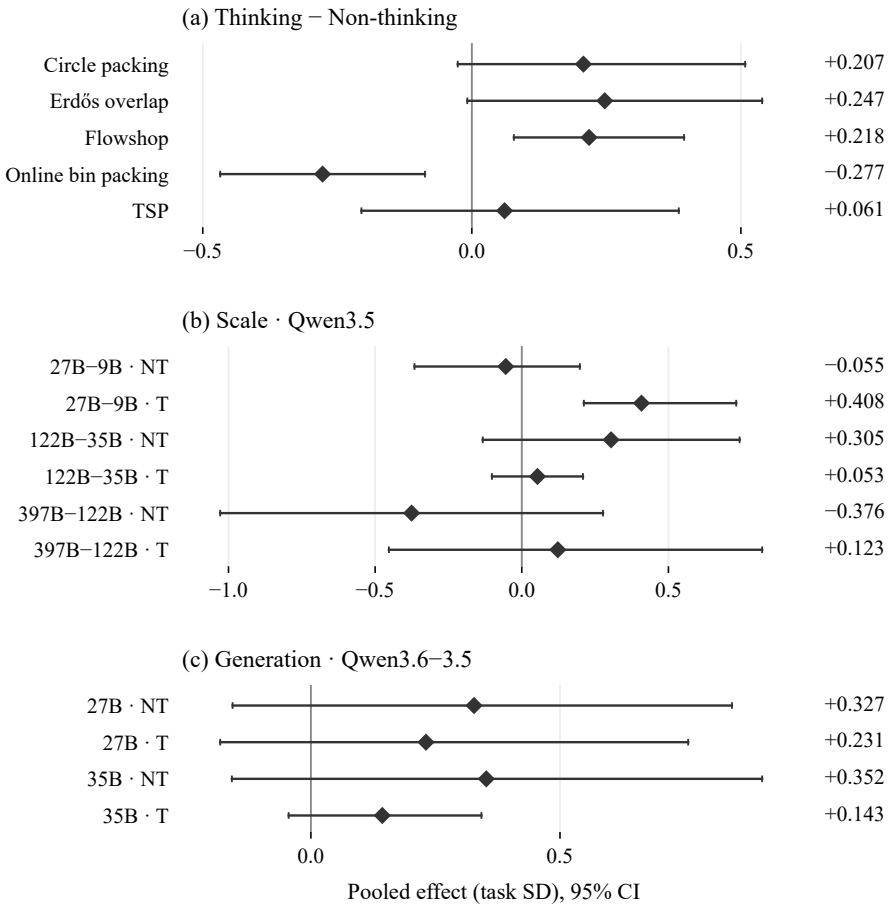

Figure 9: Pooled effects of model changes. Diamonds show across-framework means, horizontal lines their 95% intervals, and the right column the means. Each estimate averages the four framework-specific effects for the corresponding contrast in Figure 4; (b,c) additionally average five tasks. Units are task endpoint SD; NT/T denote Non-thinking/Thinking.

sure effects in the selected programs under this fixed protocol, rather than effects of framework components.

The case study combines whole-function comparisons with two execution-level checks: a returned-vector trace that explains a changed choice, and an order-preserving score transformation that leaves choices unchanged. Figure 10 reports the whole-function outcomes: 11 of the 18 programs save bins relative to their controls, and seven tie. An intermediate version from one of these OpenEvolve runs provides the returned-vector example.

C.1 A score vector can change the executed choice

The OpenEvolve stage program (Qwen3.5-9B, Non-thinking, repeat 1) computes a hybrid tightness score and writes its maximum into a vector initialized to -1. The downstream driver applies argmax to the returned vector. A selected raw score below -1 lets an unselected entry outrank it. Thus the internal best index and the driver's chosen bin can differ. Figure 11 shows the exact relevant source lines, with intervening code omitted.

For the displayed state, item 48 encounters 4999 feasible bins. The tightest bin has capacity 52 and returned score -1.022957, whereas the executed bin has capacity 100 and returned score -1. This state follows from the program's preceding placements on the same stream. Across the five complete streams, the program uses 10,127 bins versus 10,335 for its score-only control (208 fewer). This control measures the joint effect of replacing the score formula and return logic. The trace identifies

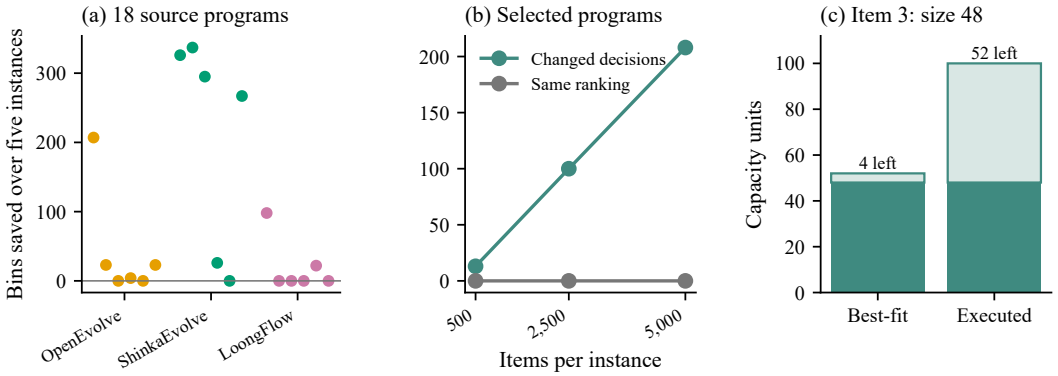

Figure 10: Scores, decisions and final packing. (a) Each point is one program, ordered by mode and repeat within framework; bin savings sum five fixed streams. (b) Cumulative savings for two displayed programs at 500, 2,500 and 5,000 items per stream. (c) The first non-best-fit decision in instance 0 of the OpenEvolve stage program. Dark fill is the incoming item; light fill is the capacity left after placement.

why the executed choice differs from the internal best index in this state, but does not attribute the total bin savings to the return-vector logic alone.

C.2 A formula change can leave every decision unchanged

The ShinkaEvolve Thinking repeat-2 program scales negative remaining capacity by the positive item size (Figure 11). For a fixed positive item, this is a strictly increasing transformation of the control scores. With the same tie rule, the maximizing set and selected bin are unchanged. All 25,000 placements match best-fit decisions, and total bin counts tie on all five instances.

Full scoring functions and code diffs accompany the data. The four TSP program-edit controls use three paired evaluation seeds each; their per-seed results accompany the data.

C.3 Exploratory source-pattern associations across tasks

All 70 task–operation–outcome combinations are reported in Table 14: seven source-code operation categories, two outcomes, and five tasks. The analysis checks the full experimental run roster. Within each run, the registered quality-independent sample contains at most 20 candidate identities; parsed, executed programs are deduplicated by source hash. A run contributes when both operation-bearing and comparison groups contain at least two distinct programs. Quality comparisons additionally require valid profiles and observed quality in both groups. Operation labels are static source-code predicates; program execution does not by itself establish that every tagged operation was dynamically invoked.

Source-code operation definitions. The detector walks the full Python abstract syntax tree and inspects function-call nodes. For a call such as np.clip(...), it matches the final attribute name, clip; for a direct call, it matches the function name. A category is present if at least one call name matches its case-sensitive list in Table 13. Categories are nonexclusive. Matching uses names without resolving imports, aliases, or the runtime call target, and includes calls inside function bodies and conditional branches. Thus, the labels describe source patterns rather than verified execution of a particular library operation. For example, if alone does not trigger clipping/piecewise, and x**2 alone does not trigger nonlinear score. The random-sampling category includes generator initialization and seed-setting calls.

(a) Changed decisions: OpenEvolve, Non-thinking, r1

Score-only control

```
15  def score(item, bins, *args, **kwargs):
16      import numpy as np
17      capacities = np.asarray(bins, dtype=np.float64)
18      value = float(np.asarray(item).reshape(-1)[0])
19      residual = capacities - value
20      return np.where(residual >= 0.0, -residual, -np.inf)
```

Program (earlier score omitted)

```
68
69      # Select best bin
70      best_idx = np.argmax(final_scores)
71      best_val = final_scores[best_idx]
72
73      # Construct output vector
74      res = np.full_like(bins, -1.0)
75      res[best_idx] = best_val
76
77      return res
```

Decision state: item 48 | tightest capacity 52 → executed capacity 100

Five streams: 10,335 control bins → 10,127 program bins (208 fewer)

(b) Same decisions: ShinkaEvolve, Thinking, r2

Score-only control

```
6   def score(item, bins, *args, **kwargs):
7       import numpy as np
8       capacities = np.asarray(bins, dtype=np.float64)
9       value = float(np.asarray(item).reshape(-1)[0])
10      residual = capacities - value
11      return np.where(residual >= 0.0, -residual, -np.inf)
```

Program

```
12      remaining = bins - item
13      # Avoid division by zero if item is 0
14      if item == 0:
15          return -remaining
16      return -remaining / item
```

Positive item: scores are rescaled, so their ordering is unchanged.

25,000 / 25,000 decisions match best-fit | 10,335 bins in both variants

Source excerpts and line numbers. Each control replaces the complete scoring function.

Figure 11: Bin-packing mechanistic case study: code, decision and outcome. Program code and score-only control excerpts, with original line numbers. Panel (a) contrasts a returned score vector with best-fit scores; the intervention replaces the whole score function. Panel (b) preserves best-fit ordering for a fixed positive item.

Table 13: Static operation-label definitions. Any listed call name activates the category. Examples illustrate the matching rule and are not additional experimental programs.

Category	Matched call names	Example
Optimizer call	SLSQP, differential_evolution, dual_annealing, least_squares, linprog, minimize, minimize_scalar, solve	minimize(f, x0)
Geometry / distance	cdist, hypot, norm, pdist, sqrt	np.linalg.norm(x)
Integral / convolution	convolve, correlate, fft, ifft, quad, simpson, trapezoid, trapz	np.convolve(a, b)
Clipping / piecewise	clip, fmax, fmin, maximum, minimum, select, where	np.where(c, x, y)
Nonlinear score	exp, log, log1p, power, sigmoid, tanh	np.log1p(x)
Ranking / index selection	argmax, argmin, argpartition, argsort, sort, sorted	np.argsort(x)
Random sampling	choice, default_rng, normal, rand, randn, random, seed, uniform	np.random.seed(0)

Effects are computed within runs and averaged with equal run weights. Validity effects are probability differences; conditional-quality effects use Q. Identical source programs connect runs into source clusters, which are resampled 5,000 times to obtain marginal 95% intervals. The registered interval rule requires at least ten clusters. Table entries marked NC have no eligible within-run contrast; IC indicates insufficient clusters for the registered interval calculation, while the descriptive effect is retained. These exploratory associations generate hypotheses for targeted program controls. The intervals are marginal, not simultaneous across the comparison family; excluding zero does not establish a multiplicity-adjusted discovery. Within-run contrasts hold task, framework, and model configuration fixed, but do not adjust for candidate length, search stage, or co-occurring source patterns. Eligible runs can also differ across comparisons. These contrasts do not identify causal operation effects or explain between-framework performance differences.

Of the 70 task–code-category–outcome comparisons, 65 have eligible within-run contrasts, and 55 meet the requirements for cluster-based 95% intervals. Table entries marked NC have no eligible contrast; IC indicates too few source clusters to calculate an interval. The reported run counts indicate the search runs contributing to each comparison.

Operation associations and program contrasts answer different questions. For example, ranking/index selection is positively associated with Flowshop validity, while its valid-quality interval crosses zero. Higher validity and better task quality are distinct outcomes of an operation.

Figure 12 resolves whole-function comparisons by program version and input stream, placing bin-count changes alongside the fraction of decisions that differ from best-fit. Changed decisions are assessed through their final bin counts, while the unchanged-decision control supplies a zero-effect reference.

Table 14: Exploratory task–source-pattern associations. V: validity; Q: quality conditional on validity. Flowshop Q effects and interval bounds are multiplied by 10^6. Cfg: model configurations; Cl: source clusters. NC: no eligible within-run contrast, so no effect or interval is estimable; IC: effect available, but fewer than ten source clusters for interval estimation. All four frameworks are represented in the source population; per-comparison framework support is supplied separately.

Operation	Outcome	Runs	Cfg	Cl	Effect	95% interval

Table 14 (continued)

Operation	Outcome	Runs	Cfg	Cl	Effect	95% interval
Circle packing						
Optimizer call	V	49	16	46	-0.1115	[-0.2184, +0.0052]
Optimizer call	Q	30	14	28	+0.1760	[+0.1082, +0.2414]
Geometry / distance	V	21	12	15	+0.7759	[+0.6933, +0.8513]
Geometry / distance	Q	0	0	0	NC	NC
Integral / convolution	V	0	0	0	NC	NC
Integral / convolution	Q	0	0	0	NC	NC
Clipping / piecewise	V	77	19	68	+0.3513	[+0.2356, +0.4661]
Clipping / piecewise	Q	31	14	31	+0.0374	[-0.0201, +0.0958]
Nonlinear score	V	98	22	92	+0.1405	[+0.0779, +0.2016]
Nonlinear score	Q	77	20	73	+0.1113	[+0.0635, +0.1568]
Ranking / index selection	V	119	24	111	+0.1353	[+0.0740, +0.1963]
Ranking / index selection	Q	88	23	85	+0.1023	[+0.0620, +0.1438]
Random sampling	V	88	22	79	+0.1081	[+0.0329, +0.1830]
Random sampling	Q	63	18	62	+0.1679	[+0.1191, +0.2173]
Erdős overlap						
Optimizer call	V	48	17	38	-0.0292	[-0.1158, +0.0646]
Optimizer call	Q	35	14	31	+0.0813	[+0.0534, +0.1064]
Geometry / distance	V	44	16	38	+0.0693	[-0.0244, +0.1462]
Geometry / distance	Q	31	14	28	+0.0244	[-0.0161, +0.0633]
Integral / convolution	V	42	17	36	+0.0945	[-0.0829, +0.2648]
Integral / convolution	Q	30	17	29	-0.0329	[-0.0635, -0.0014]
Clipping / piecewise	V	87	21	75	-0.0034	[-0.0657, +0.0664]
Clipping / piecewise	Q	65	20	59	+0.0819	[+0.0533, +0.1105]
Nonlinear score	V	109	21	91	+0.0175	[-0.0348, +0.0752]
Nonlinear score	Q	72	20	64	+0.0416	[+0.0176, +0.0668]
Ranking / index selection	V	88	22	76	+0.0387	[-0.0124, +0.0941]
Ranking / index selection	Q	67	22	58	+0.0210	[-0.0118, +0.0505]
Random sampling	V	68	18	56	+0.0665	[-0.0029, +0.1359]
Random sampling	Q	51	17	46	+0.0498	[+0.0322, +0.0716]
Flowshop						
Optimizer call	V	0	0	0	NC	NC
Optimizer call	Q	0	0	0	NC	NC
Geometry / distance	V	71	20	71	+0.0441	[-0.0033, +0.0908]
Geometry / distance	Q	62	20	62	+0.1232	[-0.1713, +0.4126]
Integral / convolution	V	4	4	4	-0.4656	IC
Integral / convolution	Q	2	2	2	+0.7577	IC
Clipping / piecewise	V	139	23	126	+0.0699	[+0.0127, +0.1305]
Clipping / piecewise	Q	111	22	102	+0.8018	[+0.5263, +1.0809]
Nonlinear score	V	103	23	98	+0.0980	[+0.0515, +0.1463]
Nonlinear score	Q	94	23	90	+0.3194	[+0.0368, +0.6279]
Ranking / index selection	V	54	20	47	+0.3407	[+0.2062, +0.4601]
Ranking / index selection	Q	23	13	22	+0.2128	[-0.4635, +0.8260]
Random sampling	V	126	22	111	+0.0269	[-0.0229, +0.0807]
Random sampling	Q	101	21	91	+1.0426	[+0.7342, +1.3884]

Table 14 (continued)

Operation	Outcome	Runs	Cfg	Cl	Effect	95% interval
Online bin packing						
Optimizer call	V	2	2	2	-0.1125	IC
Optimizer call	Q	2	2	2	-0.0049	IC
Geometry / distance	V	69	22	69	-0.0187	[-0.0750, +0.0373]
Geometry / distance	Q	61	21	61	-0.0065	[-0.0229, +0.0092]
Integral / convolution	V	2	2	2	-0.0801	IC
Integral / convolution	Q	1	1	1	5.90×10^{-5}	IC
Clipping / piecewise	V	124	23	102	+0.1725	[+0.0931, +0.2352]
Clipping / piecewise	Q	100	23	86	+0.0098	$[-9.31 \times 10^{-4}$, +0.0206]
Nonlinear score	V	120	23	101	+0.0972	[+0.0371, +0.1434]
Nonlinear score	Q	101	23	88	+0.0144	[+0.0011, +0.0267]
Ranking / index selection	V	70	21	65	-0.1416	[-0.2214, -0.0635]
Ranking / index selection	Q	51	20	49	+0.0105	[-0.0088, +0.0279]
Random sampling	V	50	17	47	-0.1103	[-0.1901, -0.0292]
Random sampling	Q	41	16	41	+0.0141	[-0.0106, +0.0404]
TSP						
Optimizer call	V	2	2	2	-0.4150	IC
Optimizer call	Q	1	1	1	-0.2956	IC
Geometry / distance	V	87	23	84	+0.0185	[-0.0121, +0.0480]
Geometry / distance	Q	84	23	81	+0.0896	[+0.0467, +0.1332]
Integral / convolution	V	2	2	2	+0.0556	IC
Integral / convolution	Q	2	2	2	+0.2240	IC
Clipping / piecewise	V	131	26	117	+0.0471	[+0.0170, +0.0838]
Clipping / piecewise	Q	119	25	106	+0.1126	[+0.0807, +0.1404]
Nonlinear score	V	149	25	132	+0.0812	[+0.0445, +0.1261]
Nonlinear score	Q	137	25	120	+0.0875	[+0.0407, +0.1271]
Ranking / index selection	V	81	23	68	+0.0181	[-0.0282, +0.0654]
Ranking / index selection	Q	73	23	60	+0.0727	[+0.0060, +0.1241]
Random sampling	V	58	18	55	-0.0675	[-0.1594, +0.0145]
Random sampling	Q	47	16	44	+0.1090	[+0.0505, +0.1730]

D Behavioral Concentration and Candidate Depth

To compare responses within a search, we select up to 20 candidates per run in hash order independently of quality and compute profiles for valid programs. Let S_r index the $n_r \geq 2$ valid profiles in run r, with $\phi_i \in \mathbb{R}^d$ constructed as specified below. For task T, define root-mean-square coordinate distance and subset dispersion by

$$d_T(i,j) = \sqrt{\frac{1}{d}\sum_{u=1}^{d}(\phi_{iu} - \phi_{ju})^2}, \qquad \mathcal{D}_T(S) = \binom{|S|}{2}^{-1} \sum_{i<j\in S} d_T(i,j), \tag{5}$$

where d is the task-specific profile dimension, u indexes coordinates, and $S \subseteq S_r$ contains at least two candidates. Run-level dispersion and mean nearest-neighbor distance are

$$\begin{aligned} D_r &= \mathcal{D}_T(S_r), \\ L_r &= \frac{1}{n_r}\sum_{i\in S_r}\frac{1}{k_r}\sum_{j\in \mathrm{kNN}(i;S_r\setminus\{i\})} d_T(i,j), \qquad k_r = \min(5, n_r - 1). \end{aligned} \tag{6}$$

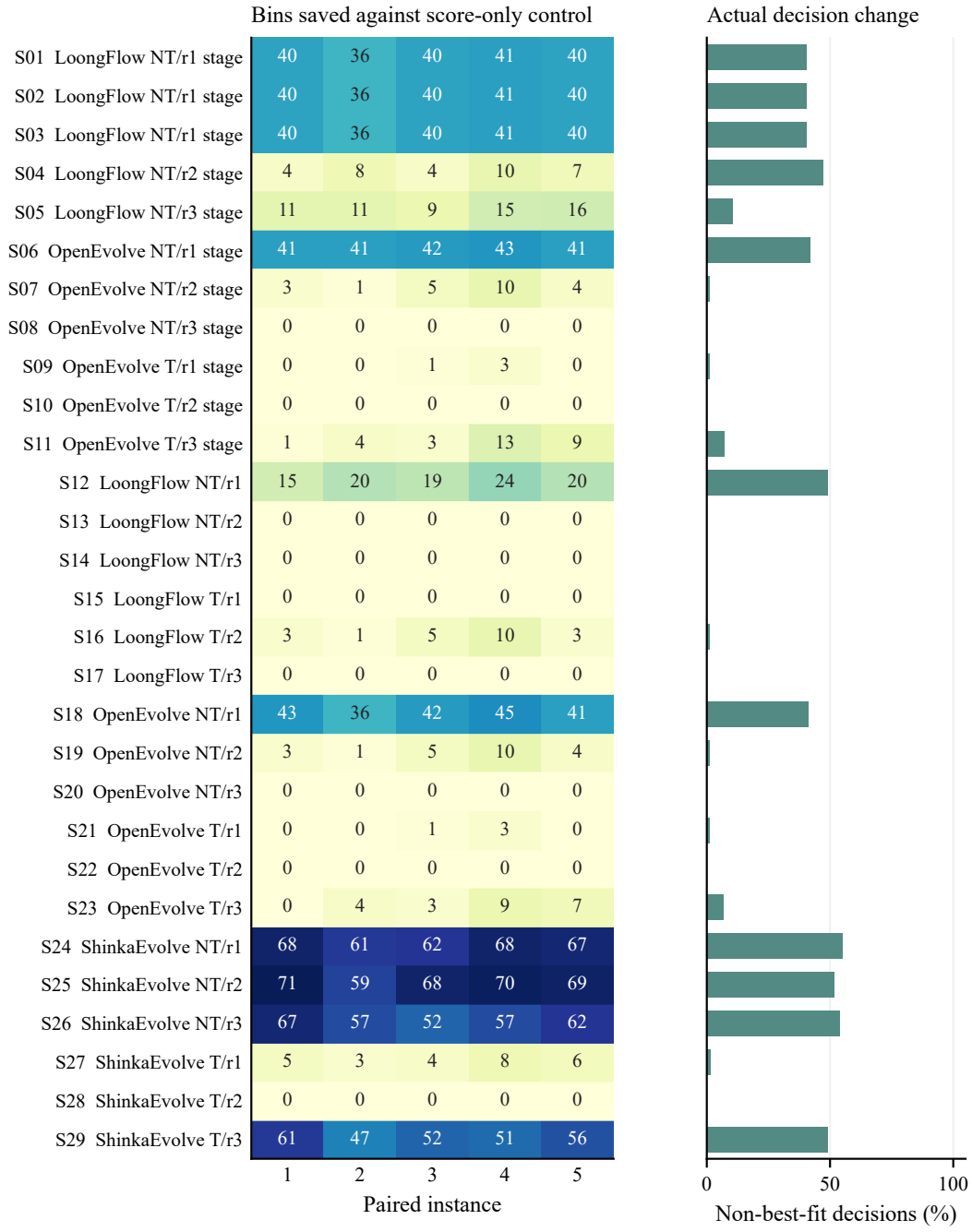

Figure 12: Packing decisions and instance outcomes. Rows represent program versions and columns fixed item streams. Cell values are control minus original bin counts, so positive values favor the original program. Bars show the fraction of its 25,000 decisions that differ from best-fit. S IDs identify program versions in the accompanying dictionary.

The neighbor set contains the k_r candidates closest to i under d_T. Thus D_r measures overall dispersion, whereas L_r is the mean nearest-neighbor distance; lower values indicate more similar behaviors at the respective scale.

Let H_r contain the best $\max(2, \lceil \alpha n_r \rceil)$ candidates by quality plus all exact ties at the cutoff, with $\alpha = 0.2$ and $m_r = |H_r|$. Let $U_r(m_r)$ be uniform over all size-m_r subsets of S_r. The concentration contrast is

$$C_r = \mathcal{D}_T(H_r) - \text{median}_{S \sim U_r(m_r)} \mathcal{D}_T(S). \tag{7}$$

Negative C_r means that high-quality candidates have more similar responses than the median size-matched random subset of the same run's pool. For pools of at most 20, all subsets are enumerated.

D.1 Execution-profile construction

We organize 16,660 execution profiles across the five tasks. Table 15 specifies the raw vector z for each task. We orient and center each vector as $\phi = s_T(z - \bar{z}\mathbf{1})$, where $s_T = +1$ for circle packing and Erdős overlap and $s_T = -1$ for the three instance-based tasks. Centering is across all coordinates of the individual candidate, including the 104 geometric coordinates for circle packing. No division by a candidate or population standard deviation is applied. Multiplication by a common task sign leaves within-task RMS distances unchanged. Distances are compared within tasks; their numerical magnitudes do not share a common scale across tasks.

Table 15: Execution-profile inputs and coordinates. The dimension d is fixed within each task. All vectors undergo the orientation and centering defined above.

Task	d	Raw coordinates and input definition
Circle packing	104	For each of 26 circles: center (x, y), radius divided by $\max(\text{mean radius}, 10^{-12})$, and fraction of the other 25 circles with absolute boundary gap at most 10^{-6}. Circles are ordered by $(y, x, -r)$, with original index breaking exact ties.
Erdős overlap	257	Values of the returned h sequence, linearly interpolated onto 257 equally spaced positions from 0 to 1 in normalized array-index coordinates, including both endpoints.
Flowshop	3	One makespan per fixed evaluation instance, in the evaluator's instance order. Instance sizes are specified in Appendix F.1.
Online bin packing	5	One excess-bin ratio per fixed Weibull stream of 5,000 items; capacity is 100. Ratios are retained before percentage display scaling.
TSP	3	One signed relative tour-length gap per fixed 100-city instance, expressed in percentage units. These per-instance gaps precede the mean-gap clipping used for Q.

Algorithm: Constructing a run's execution-profile distances

1. **Input:** the run's candidate sample, task-specific evaluation inputs, ordered coordinate definition, and extractor settings in Table 15.
2. For each candidate, obtain its evaluation output and raw task vector z. Retain its separately evaluated quality Q.
3. Retain profiles with valid task outputs, finite values and valid flags for every required coordinate. Require the same coordinate order, probe definition and preprocessing settings within the compared pool.
4. Set $\phi = s_T(z - \bar{z}\mathbf{1})$ and collect their candidate indices into S_r. Preserve invalid evaluations in the candidate accounting; do not replace their profiles with zeros.
5. For $n_r \geq 2$, compute the pairwise distances in equation 5. Use this matrix to obtain D_r, L_r, and the quality-selected concentration contrast in equations 6 and 7.
6. **Output:** the valid profile pool, associated qualities, and pairwise distance matrix. Fewer than two valid profiles yield no pairwise-distance summary.

Task–framework summaries. Figure 6(a) reports means over 384 runs matched on experimental conditions across frameworks. Panel (b) uses a separate fixed sample of 100 runs, five per task–framework group. The samples overlap in 43 runs. Of the 100 proposal, request, and token denominators, respectively 1, 27, and 34 are estimated; the rates assume a common process window for each numerator and denominator. Events counts strict running-best improvements; Gain sums their quality increments. Rates use recorded or estimated proposal, request, or token counts, are converted to the displayed units per run, and are then averaged within each group. Appendix E reports rates within recorded proposal or candidate-export windows, using the events and tokens from each window. In panel (a), D, L, C, and Q denote pairwise behavioral dispersion, mean nearest-neighbor distance, high-quality concentration contrast, and endpoint quality, respectively. Shading is scaled within each task and metric; darker C denotes a more negative contrast.

Figure 13 summarizes all 1,560 run records. At candidate cap 20, 1,196 have $C < 0$ (76.7%), indicating lower dispersion in the high-quality subset relative to the reference subsets at this sampled depth. These categories describe the sampled candidate pools; the paired depth comparison below examines their sensitivity.

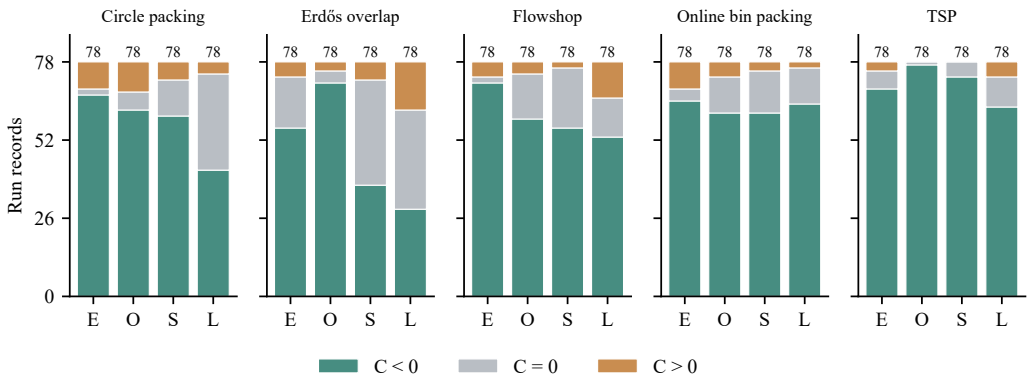

Figure 13: Concentration categories across frameworks. Each task includes 312 run records, with 78 per framework. E, O, S and L denote EoH, OpenEvolve, ShinkaEvolve and LoongFlow.

D.2 Candidate-depth sensitivity

Table 16 and Figure 14 compare the concentration categories of the same 1,560 records at candidate caps 20 and 40. Negative counts are 1,196 and 1,083, respectively. There are 304 strict sign reversals and 255 transitions to or from zero; these sum to 559 category changes (35.8%). Although negative contrasts remain the majority at both caps, run-level categories depend substantially on candidate depth. We therefore interpret C as a supplementary description of a specified candidate pool, rather than a stable run-level classification or an explanation of framework performance.

Table 16: Candidate-depth sensitivity across tasks. Counts compare the concentration categories at candidate caps 20 and 40.

Task	Runs	$C_{20} < 0$	$C_{40} < 0$	Reversed	Via zero
Circle packing	312	231	212	72	52
Erdos overlap	312	193	192	51	89
Flowshop	312	239	219	54	50
Bin packing	312	251	213	78	42
TSP	312	282	247	49	22
Total	1560	1196	1083	304	255

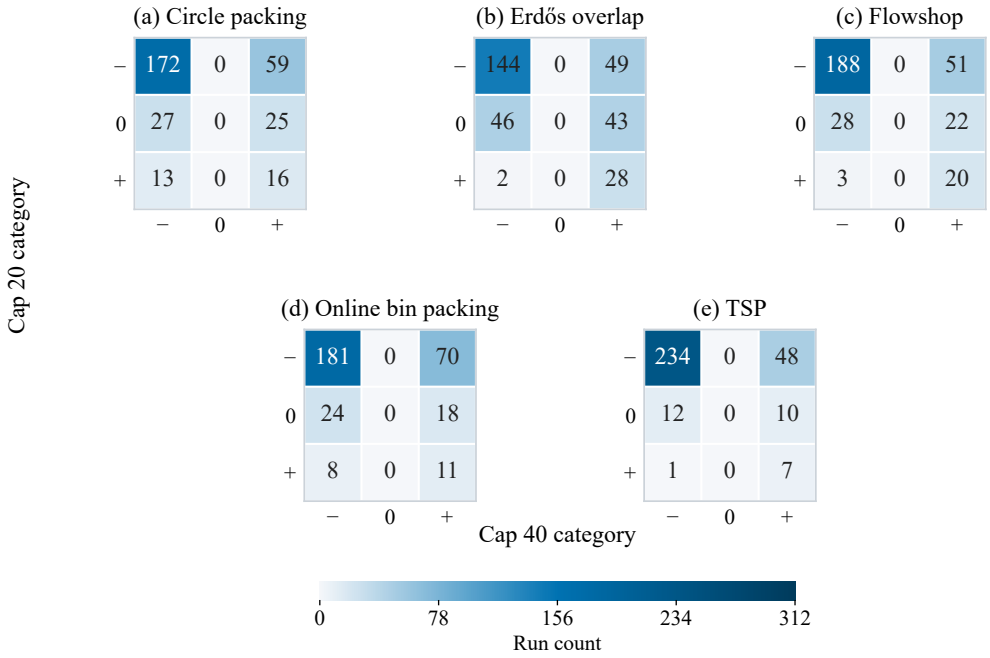

Figure 14: Concentration categories across candidate depths. Rows show cap-20 categories and columns show cap-40 categories. Every task contains 312 runs; all panels use the same color scale.

When the high-quality set equals the entire valid pool, equation 7 gives zero by construction. Thus a zero can reflect pool geometry and quality resolution. For centered instance profiles, a common performance shift across all instances leaves the profile unchanged; the resulting distances describe relative response patterns across instances.

E Search Progress and Efficiency across Tasks and Model Configurations

We analyze the complete 1,560-run experiment across five tasks, four frameworks, and 26 model configurations. The nominal 50M-token request-start threshold permits requests initiated below 50M to finish and contribute their candidates after cumulative use crosses the threshold. Endpoint token use thus includes completed-request overshoot; equal request-start thresholds do not imply equal realized endpoint expenditure. Progress analyses use the accompanying reconstructed budget grid at one-million-token intervals from 0 to 50M. All frameworks within a task share the same reporting grid and target definition. The 50M grid endpoint summarizes this stopping protocol, including completed-request candidates, rather than a strict 50M-consumed-token cutoff. Timing-coordinate sensitivity assesses the supplied reconstruction scenarios; it does not quantify between-framework differences in endpoint overshoot. The grid origin is the stored pre-search baseline quality, $q_r(0) = Q_{r,\text{initial}}$; this anchors the quality scale without assigning zero cost to initialization.

E.1 Quality across the complete configuration panel

Table 17 reports native scores after applying the task transformation separately to each run. Each framework–task entry contains 78 runs, equally weighting all 26 configurations. The complete configuration means accompany the data. Standard deviations describe variation among runs; they are not confidence intervals for a framework effect.

Table 17 places these differences in native units: ShinkaEvolve has the largest mean radius sum and smallest bin-packing gap, LoongFlow the smallest construction bound, OpenEvolve the smallest makespan, and EoH the smallest TSP gap. The following analyses examine how these endpoint differences develop across the budget and vary across configurations.

Table 17: Native endpoint quality across all configurations.

Framework	Mean	SD
Circle packing: Radius sum ↑		
EoH	2.556	0.022
OpenEvolve	2.519	0.051
ShinkaEvolve	2.571	0.021
LoongFlow	2.552	0.064
Erdős overlap: Construction bound ↓		
EoH	0.388	0.005
OpenEvolve	0.385	0.006
ShinkaEvolve	0.406	0.035
LoongFlow	0.382	9.59×10^{-4}
Flowshop: Makespan ↓		
EoH	3498.80	14.465
OpenEvolve	3457.62	34.444
ShinkaEvolve	3547.92	29.121
LoongFlow	3505.33	12.890
Online bin packing: Excess gap (%) ↓		
EoH	1.485	0.338
OpenEvolve	2.282	1.076
ShinkaEvolve	1.293	0.521
LoongFlow	2.000	0.630
TSP: Nonnegative gap (%) ↓		
EoH	0.005	0.019
OpenEvolve	0.031	0.090
ShinkaEvolve	0.008	0.015
LoongFlow	0.148	0.368

E.2 Stagewise improvement

For run r, let $q_r(b)$ denote its quality on the reconstructed reporting grid at budget b. For stages ending at $b \in \{10, 20, 30, 40, 50\}$ million tokens, define

$$\Delta q_{r,b} = q_r(b) - q_r(b-10), \qquad p_{f,t,b} = \frac{1}{78} \sum_{r \in (f,t)} \mathbf{1}[\Delta q_{r,b} > 10^{-12}].$$

Mean stage gain and the proportion of runs with positive stage gain answer different questions: the former measures improvement magnitude, while the latter shows how broadly improvement is distributed. All 7,800 run–stage observations are included, including zero-gain stages. They represent 1,560 independent searches.

Figure 15 displays both quantities. On this grid, LoongFlow's mean circle-packing gain is larger in 10–20M than in 0–10M, while EoH, OpenEvolve, and ShinkaEvolve have their largest mean gains in the first stage. This is a comparison of the reconstructed trajectories. The supplied sensitivity table separately reports each run's 2.5th and 97.5th gain percentiles across 200 timing draws. Averaging those bounds gives a descriptive sensitivity envelope, not a confidence interval or a count of additional independent searches.

Budget-coordinate sensitivity. The reference grid and 200 accompanying timing scenarios contain the same 1,560 run identities. We align records by run and scenario, average the 78 runs within each task–framework group at each checkpoint, and then subtract framework means. Scenario proportions describe sensitivity to the supplied budget coordinates, not probabilities of a true ranking. Table 18 compares the two highest mean-endpoint Q frameworks in each task at 10M and 20M; this selection uses endpoint quality rather than scenario ordering. The accompanying table reports all six framework pairs at 10M, 20M, 30M, 40M, and 50M for every task. Ranges use the empirical 2.5th and 97.5th percentiles with linear interpolation. For Erdős, OpenEvolve leads LoongFlow at 10M in 198 of 200 scenarios and at 20M in 99, making the latter ordering sensitive to budget coordinates. The main-figure bootstrap holds these coordinates fixed and describes run-sampling variation conditional on the reference reconstruction.

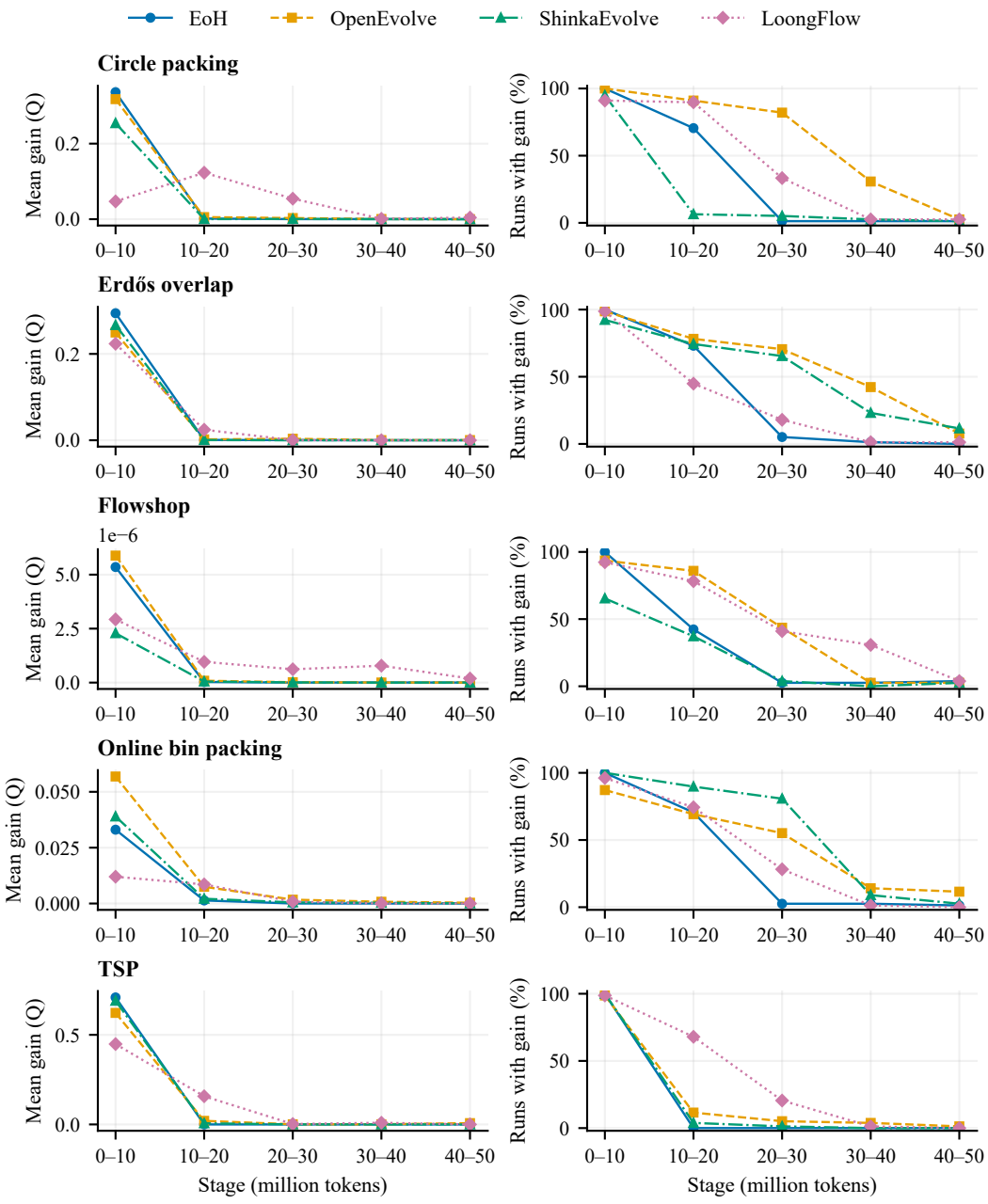

Figure 15: Stagewise search progress. Left: mean stage gain; right: percentage of runs with positive gain on the reconstructed grid. Colors and line styles identify frameworks as in the main budget figure. Both columns use the same runs; Q scales vary by task.

Table 18: Framework ordering under timing sensitivity. Each task compares its two highest mean-endpoint Q frameworks, selected by mean endpoint quality; differences are A minus B. OE, SE, and LF denote OpenEvolve, ShinkaEvolve, and LoongFlow. Ranges span the 2.5th–97.5th scenario percentiles, not sampling confidence intervals.

A minus B	Budget	Reference ΔQ	Central scenario range	A leads
Circle packing				
EoH - SE	10M	-0.0068	[-0.0557, +0.0010]	7/200
EoH - SE	20M	-0.0054	[-0.0346, +0.0044]	15/200
Erdős overlap				
OE - LF	10M	+0.0110	[+0.0106, +0.0628]	198/200
OE - LF	20M	-0.0114	[-0.0153, +0.0210]	99/200
Flowshop				
EoH - OE	10M	-3.35×10^{-6}	$[-3.38 \times 10^{-6}, -2.60 \times 10^{-6}]$	0/200
EoH - OE	20M	-3.41×10^{-6}	$[-3.62 \times 10^{-6}, -2.96 \times 10^{-6}]$	0/200
Online bin packing				
EoH - SE	10M	-0.0003	[-0.0032, +0.0046]	95/200
EoH - SE	20M	-0.0010	[-0.0026, +0.0016]	64/200
TSP				
EoH - SE	10M	+0.0093	[-0.0206, +0.0416]	155/200
EoH - SE	20M	+0.0030	[-0.0111, +0.0065]	106/200

E.3 Common-target attainment

Within each task, all frameworks are evaluated against the same threshold. TSP uses the fixed target $Q \geq 1 - 10^{-12}$. With $Q = 1/(1 + \max(\text{mean_gap}, 0))$, this denotes the evaluator maximum within tolerance rather than mathematical optimality on every instance. Because clipping maps nonpositive mean gaps to the same score, attainment compares reaching the evaluator ceiling and does not rank solutions within that group. For a comparative reference across all five tasks, we also compute the pooled 25th, 50th, and 75th percentiles of the 312 endpoint scores per task. These are retrospective endpoint-derived reference targets; their values are fixed before comparing framework attainment curves. They are not held-out task standards.

At budget b, the attainment fraction is the number of runs with $q_r(b)$ at or above the target divided by all 78 runs of the framework in that task. Runs that never attain the target remain in the denominator. First attainment on the one-million-token grid is recorded separately from right censoring at 50M. Runs ending early retain their best-so-far quality on the reporting grid without adding improvement events or consumed tokens. Figure 16 uses the pooled median for the first four tasks and the fixed task target for TSP. Both quartile reference curves are included in the accompanying tables.

At the fixed TSP target, the endpoint counts are 70/78 for EoH, 60/78 for ShinkaEvolve, 48/78 for OpenEvolve, and 40/78 for LoongFlow. Intermediate attainment budgets inherit the reconstructed budget coordinates; endpoint attainment uses the same final scores as the endpoint comparison.

E.4 Resource conversion within recorded windows

We analyze 26 proposal windows from OpenEvolve and ShinkaEvolve, covering all five tasks, and separately 27 EoH candidate-export windows across Flowshop, bin packing, and TSP. Let N be the number of proposals or exported candidates in the relevant protocol, U the number of strict running-best improvements, G the final minus first valid quality within that window, and T its recorded token total. For $U > 0$,

$$\frac{G}{T} = \frac{N}{T}\,\frac{U}{N}\,\frac{G}{U}.$$

This separates candidate production, improvement frequency, and improvement magnitude within a run. With $U = 0$, the observed zero gain and zero event rate remain in the efficiency summaries; the per-improvement magnitude is defined on windows with improvements. Each rate is calculated within its source window and then averaged equally across source runs. Multiplying group-average factors is not used to estimate group-average efficiency.

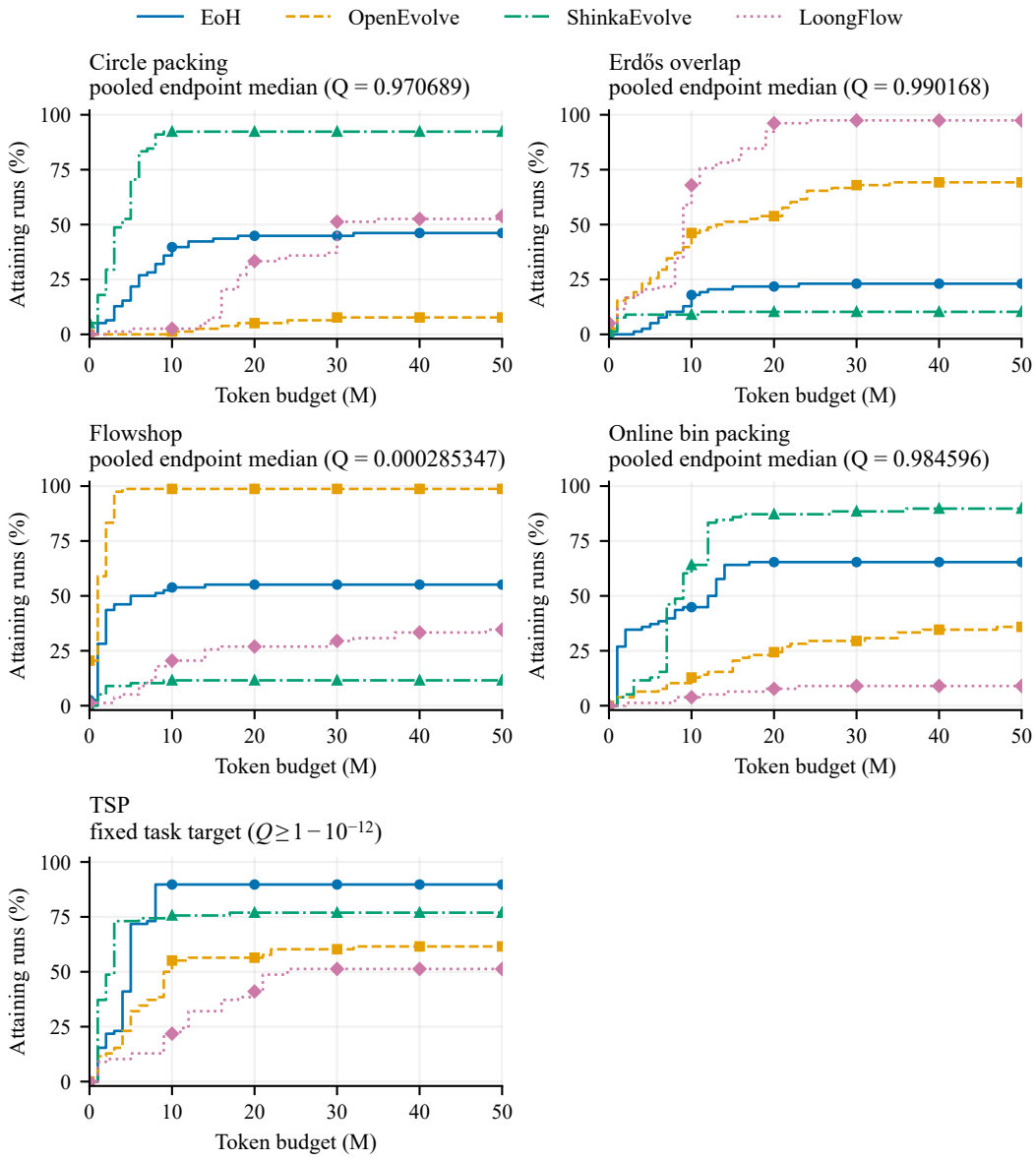

Figure 16: Common-target attainment. Curves use the reconstructed budget grid and retain all 78 runs per framework in the denominator. Colors and line styles match the main budget figure.

Table 19: Attaining runs at common quality targets. The first four tasks use pooled endpoint medians; TSP uses its fixed target.

Framework	10M	20M	50M
Circle packing			
EoH	31/78	35/78	36/78
OpenEvolve	1/78	4/78	6/78
ShinkaEvolve	72/78	72/78	72/78
LoongFlow	2/78	26/78	42/78
Erdős overlap			
EoH	14/78	17/78	18/78
OpenEvolve	36/78	42/78	54/78
ShinkaEvolve	7/78	8/78	8/78
LoongFlow	53/78	75/78	76/78
Flowshop			
EoH	42/78	43/78	43/78
OpenEvolve	77/78	77/78	77/78
ShinkaEvolve	9/78	9/78	9/78
LoongFlow	16/78	21/78	27/78
Online bin packing			
EoH	35/78	51/78	51/78
OpenEvolve	10/78	19/78	28/78
ShinkaEvolve	50/78	68/78	70/78
LoongFlow	3/78	6/78	7/78
TSP			
EoH	70/78	70/78	70/78
OpenEvolve	43/78	44/78	48/78
ShinkaEvolve	59/78	60/78	60/78
LoongFlow	17/78	32/78	40/78

Tables 20 and 21 summarize proposal and candidate-export windows, respectively. The two window protocols retain separate denominators. Proposal windows count proposals and their recorded tokens; candidate-export windows count exported candidates and associated tokens. Recorded-token efficiency is distinct from complete request billing, and these window comparisons describe their represented source runs.

In Table 20, the three OpenEvolve bin-packing windows have mean gain 2.98×10^{-5} per million recorded tokens, compared with 4.87×10^{-4} in the three ShinkaEvolve windows. Zero-gain windows are included for both frameworks. The comparison connects recorded generation and improvement outcomes within these windows; it does not identify a causal effect of a framework component. Table 21 shows positive mean gains for EoH in all three represented tasks, while retaining the zero-gain bin-packing window in the average.

Table 20: Proposal windows. Events/100 uses the protocol-specific candidate denominator; 1M denotes one million recorded tokens. Zero gain is a source-run count.

Framework	Events/100	Events/1M	Gain/1M	Zero gain
Circle packing				
OpenEvolve	0.70	0.38	0.010	0
ShinkaEvolve	2.60	0.52	0.012	0
Erdős overlap				
OpenEvolve	1.13	0.67	0.004	0
ShinkaEvolve	1.51	0.68	0.012	0
Flowshop				
OpenEvolve	0.19	0.13	1.12×10^{-7}	1
ShinkaEvolve	0.16	0.07	3.29×10^{-8}	4
Online bin packing				
OpenEvolve	0.08	0.05	2.98×10^{-5}	1
ShinkaEvolve	0.57	0.26	4.87×10^{-4}	1
TSP				
OpenEvolve	0.11	0.09	0.011	0
ShinkaEvolve	1.58	1.47	0.112	0

Table 21: Candidate-export windows. Events/100 uses the protocol-specific candidate denominator; 1M denotes one million recorded tokens. Zero gain is a source-run count.

Framework	Events/100	Events/1M	Gain/1M	Zero gain
Flowshop				
EoH	1.60	4.84	3.32×10^{-6}	0
Online bin packing				
EoH	2.70	7.81	0.038	1
TSP				
EoH	0.08	0.49	0.058	0

E.5 Stability across configurations and repeats

We count unique first-place and shared first-place configurations using a 10^{-12} tolerance on mean Q. Unique and shared first places are counted separately. For each of the 30 taskwise framework pairs, the accompanying table reports the numbers of positive, negative, and tied configuration differences, along with the range of mean differences obtained by leaving out one configuration at a time. Repeat variation is reported as the within-configuration standard deviation, separately from variation across configuration means.

ShinkaEvolve is uniquely first in 19/26 circle-packing configurations and 20/26 bin-packing configurations. LoongFlow is uniquely first in 25/26 Erdős configurations, and OpenEvolve in 25/26 Flowshop configurations. TSP includes extensive ties: EoH has seven unique and thirteen shared first places. These counts describe stability across the tested model configurations and independent search repeats.

Table 22 reports Thinking minus Non-thinking endpoint differences for the eleven model configurations with both modes, retaining three independent runs per mode. Differences are calculated between mode means within the same model configuration; repeat numbers are not treated as matched random seeds. The displayed average equally weights the eleven paired-mode contrasts. The accompanying 220-row table also separates initial-quality and subsequent-gain differences. Table 22 distinguishes average effect size from sign consistency: for example, LoongFlow has a positive mean circle-packing difference despite negative differences in eight of the eleven mode pairs.

Table 22: Mode differences in endpoint quality. Positive and negative columns count paired-mode contrasts, with a 10^{-12} numerical tolerance.

Framework	Mean ΔQ	Positive	Negative
Circle packing			
EoH	0.002	10	1
OpenEvolve	0.008	5	6
ShinkaEvolve	7.00×10^{-4}	5	6
LoongFlow	0.004	3	8
Erdős overlap			
EoH	0.006	10	1
OpenEvolve	0.008	11	0
ShinkaEvolve	0.025	8	3
LoongFlow	0.001	10	1
Flowshop			
EoH	7.07×10^{-7}	10	1
OpenEvolve	1.31×10^{-6}	10	1
ShinkaEvolve	1.27×10^{-7}	7	4
LoongFlow	7.44×10^{-7}	10	1
Online bin packing			
EoH	-0.003	2	9
OpenEvolve	-0.006	4	7
ShinkaEvolve	-0.003	4	7
LoongFlow	0.003	6	5
TSP			
EoH	0.009	4	0
OpenEvolve	-0.001	3	6
ShinkaEvolve	0.002	5	4
LoongFlow	0.012	4	6

F Protocols, Definitions and Statistical Sensitivity

F.1 Task evaluation and score definitions

Task interfaces and evaluation inputs. All four frameworks share the evaluator and verifier for each task. Flowshop, online bin packing, and TSP use benchmark data and task implementations released with EoH (Liu et al., 2024), with identical fixed instance sets across frameworks, model configurations, and search repeats. Complete-construction tasks return solutions for verification; embedded-operator tasks insert generated functions into fixed solver or allocation drivers.

Execution settings and time limits. The supplied launch configurations set OMP_NUM_THREADS, OPENBLAS_NUM_THREADS, MKL_NUM_THREADS, and NUMEXPR_NUM_THREADS to one. These settings limit numerical-library threading; framework-level search and evaluation concurrency remain controlled by the run-specific native configurations supplied with the code. Three time limits have distinct roles: the LLM request timeout, the framework's candidate-evaluation timeout, and the task solver's per-instance time limit.

Circle packing. The task is to construct 26 non-overlapping circles contained in the unit square while maximizing the sum of their radii. Candidate programs return the circle geometry through the framework-specific task interface. Evaluation checks the construction's geometric feasibility, and the radius sum is expressed relative to the benchmark reference value 2.635862756413698 for cross-framework reporting. All four frameworks use the same geometric verifier, with common containment and non-overlap checks and identical numerical feasibility tolerances. Framework-specific return interfaces do not change these validation rules. This task evaluates a geometric construction rather than performance on a collection of sampled instances.

Erdős minimum overlap. The program returns a discretized function h, its reported bound, and its grid size. On a uniform grid with spacing $2/n$, the evaluator checks $0 \leq h_i \leq 1$ and that the discrete integral is approximately one. It recomputes the maximum full discrete correlation between

h and $1-h$, scaled by the grid spacing, and checks agreement with the reported bound. The integral and bound checks use NumPy's closeness test with absolute tolerances 10^{-3} and 10^{-4}, respectively, and its default relative tolerance. The grid size is returned by the candidate. Smaller verified bounds are better.

Flowshop. The generated function receives the current job sequence and processing-time matrix and returns a perturbed matrix and selected jobs. The supplied driver reads 64 instance files and evaluates the first three: each has 50 jobs, with 20, 13, and 14 machines, respectively. Search starts from an NEH sequence and alternates local search with the generated perturbation rule. Per-instance limits are 1,000 iterations and 30 seconds, stopping when either limit is reached. At most five returned jobs are used for perturbation, and the driver returns to its best sequence every 50 iterations. It resets Python's random generator to 2024 for each instance. The objective is the mean best makespan across the three instances.

Online bin packing. The supplied Weibull dataset contains five fixed sequences of 5,000 items, each with bin capacity 100. For each arriving item, the driver supplies feasible residual capacities to score(item, bins) and selects the first maximum-scoring feasible bin. Let B_j be the number of used bins and $B_j^{\min} = \lceil \sum_i w_{ji}/100 \rceil$ the volume lower bound for sequence j. The evaluator returns $(\overline{B} - \overline{B^{\min}})/\overline{B^{\min}}$, a relative excess above the mean volume bound. This aggregation is a ratio of means. The paired program controls in Appendix C separately compare original and replacement scoring functions on complete fixed streams and report differences in bin counts.

TSP. The generated function updates the edge-distance matrix used by guided local search. We use the TSP benchmark released with EoH (Liu et al., 2024), evaluating its first three 100-city Euclidean instances against the supplied reference tour costs. Initialization uses a nearest-neighbor tour starting at city 0; each instance has a 10-second solver limit, at most 1,000 search iterations, and one perturbation move per step. The evaluator averages the three gaps relative to the supplied reference costs, then the shared wrapper applies the nonnegative clipping used in Q. Nonpositive mean gaps map to the same ceiling score, so these summaries cannot distinguish programs within that ceiling group. Endpoint gaps, tied ranks, and budget-dependent target attainment therefore provide complementary descriptions under this protocol.

Repeated runs and separate evaluations. For circle packing and Erdős overlap, evaluation verifies the returned construction. Flowshop, online bin packing, and TSP use the fixed instance inputs described above for candidate evaluation. Main endpoint comparisons characterize optimization performance under these task protocols. Independent search repeats measure variation across program-search runs. The held-out analysis in Appendix A.2 withholds one search repeat for framework selection on the same evaluation inputs. Program reevaluations, execution-profile measurements, and same-input controls use the protocols specified in Appendices C and D. Their measurements are distinct from the main per-run endpoint summaries. The accompanying task package contains the instance files, evaluator wrappers, and solver implementations defining the protocols above.

Quality transformations and aggregation. All Q values are oriented higher-is-better. Circle radius sum is $2.635862756413698Q$; the Erdős construction bound is $0.38092303510845016/Q$; Flowshop makespan and bin-packing excess gap are $Q^{-1} - 1$. TSP uses

$$Q = [1 + \max(\text{mean_gap}, 0)]^{-1}, \tag{8}$$

where mean_gap is the evaluator's mean relative gap expressed as a percentage numerical value: each instance contributes $100(\ell/\ell_{\text{ref}} - 1)$, where ℓ is the tour length and ℓ_{ref} its reference value. Thus $\max(Q^{-1} - 1, 0)$ already gives the clipped TSP gap in percent, whereas the bin-packing ratio is multiplied by 100 for percentage display. Gap tables use percent and gap differences use percentage points. These transformations are applied per run before averaging. Endpoint comparisons use the unrounded statistic specified for each analysis: mean Q for configuration rankings and mean native score for native-score comparisons. Values within an absolute tolerance of 10^{-12} of the best value are treated as tied, with no relative tolerance. This numerical tie rule does not define practical equivalence or establish statistical superiority. For complete rankings, values are ordered from best to worst; each tie group contains the best remaining value and all remaining values within this tolerance of it, and receives the average of the ranks it occupies. This grouping does not chain together

values whose total separation exceeds the tolerance. Rounding is used only for display; ties in mean Q and mean native score are assessed separately.

F.2 Model interactions and test resolution

Thinking contrasts compare the two modes within each of 11 paired model configurations. Model-size and generation comparisons comprise the 12 named comparisons in Table 24. Each interaction compares the model effects of two frameworks. Overall Thinking tests use the 11 configuration pairs as sign-flip units after averaging over tasks; model-size and generation tests use the five tasks. Taskwise Thinking tests use the 11 pairs within each task. Thus, the endpoint family contains $6 + 12 \times 6 = 78$ tests, and the initial-quality/search-gain family contains 156 tests. The corresponding taskwise Thinking families contain 30 and 60 tests.

The two-sided exact sign-flip resolution is $2/2^{11} = 9.765625 \times 10^{-4}$ for Thinking and $2/2^5 = 6.25 \times 10^{-2}$ for model-size/generation. These resolutions constrain the evidence attainable after Holm correction; a nonsignificant result does not establish equal effects. Table 23 reports all four families. Joint configuration-pair bootstrap summaries use 50,000 replicates.

Table 23: Complete model-interaction test families. Holm correction is applied separately within each family at $\alpha = 0.05$.

Comparison family	Tests	First Holm threshold (10^{-3})	Passing
Overall endpoint	78	0.641	0
Overall initial/gain	156	0.321	0
Taskwise Thinking endpoint	30	1.667	1
Taskwise Thinking initial/gain	60	0.833	0

F.3 Parametric sensitivity

A supplementary parametric-bootstrap analysis evaluates 65 taskwise omnibus tests (13 model contrasts across five tasks) and 390 framework-pair contrasts, with three independent runs per condition. Omnibus probabilities use 200,000 draws and plus-one tail estimates; pairwise tests use conservative t_2 tails. Seven omnibus tests pass Holm correction across 65 tests. Four remain after joint correction across 455 tests using one-sided 95% Monte Carlo upper bounds: Erdős SG05 and SG11, and Flowshop SG08 and SG12. No pairwise contrast passes the joint correction. Zero-variance, nonestimable omnibus cases remain in the correction family with a probability placeholder of one. Calibration with 2,000 outer datasets and 5,000 inner bootstraps gives a 10.45% false-positive rate under skewed, heteroscedastic errors at nominal 0.05, compared with 3.60–5.10% under the three normal-error settings. These results provide sensitivity evidence; the model effects in Section 4.2 remain descriptive comparisons. The accompanying data retain all test results, framework effects, and pairwise intervals. Table 24 defines the model contrasts.

Table 24: Complete model-contrast definitions. All SG comparisons use Qwen models; scale comparisons are within Qwen3.5. The first named configuration minus the second defines the effect. T and NT denote Thinking and Non-thinking. MODE denotes MODE_ALL in the data.

ID	Comparison	Mode
MODE	Thinking minus Non-thinking, averaged equally over the 11 paired models	Paired modes
SG01	27B vs. 9B	NT
SG02	122B-A10B vs. 35B-A3B	NT
SG03	397B-A17B vs. 122B-A10B	NT
SG04	3.6 vs. 3.5, 27B	NT
SG05	3.6 vs. 3.5, 35B-A3B	NT
SG06	27B vs. 9B	T
SG07	122B-A10B vs. 35B-A3B	T
SG08	397B-A17B vs. 122B-A10B	T

Table 24 (continued)

ID	Comparison	Mode
SG09	3.6 vs. 3.5, 27B	T
SG10	3.6 vs. 3.5, 35B-A3B	T
SG11	397B-A17B vs. 35B-A3B	NT
SG12	397B-A17B vs. 35B-A3B	T

G Complete Model Panels

Figures 17, 18, 19, 20, and 21 contain the nine models absent from Figure 3, in the same 3×3 order for every task. Horizontal coordinates estimate attainment of each configuration's own endpoint on the reconstructed grid; they are not budgets to a common quality target. The 15 main-text panels and 45 appendix panels display all 520 task–framework–configuration means covering 1,560 runs. Colors encode frameworks; filled circles, squares and diamonds encode Thinking, Non-thinking and default modes. Each panel marks one configuration with the best native mean (maximum for circle packing, minimum otherwise). Axis limits retain all observations and vary by panel. Bin-packing and TSP gaps are displayed as percentages. TSP uses base-10 symlog axes, linear up to 1M tokens and a gap of 0.01%; the dotted line marks the latter transition. Other axes are linear. Read with the coverage counts in Table 6, the circle-packing and bin-packing panels illustrate broad ShinkaEvolve leadership alongside configuration-specific exceptions; the Erdős and flowshop panels accompany the broader LoongFlow and OpenEvolve coverage. For TSP, coincident near-zero gaps make tied ranks and target attainment more informative than visual separation alone.

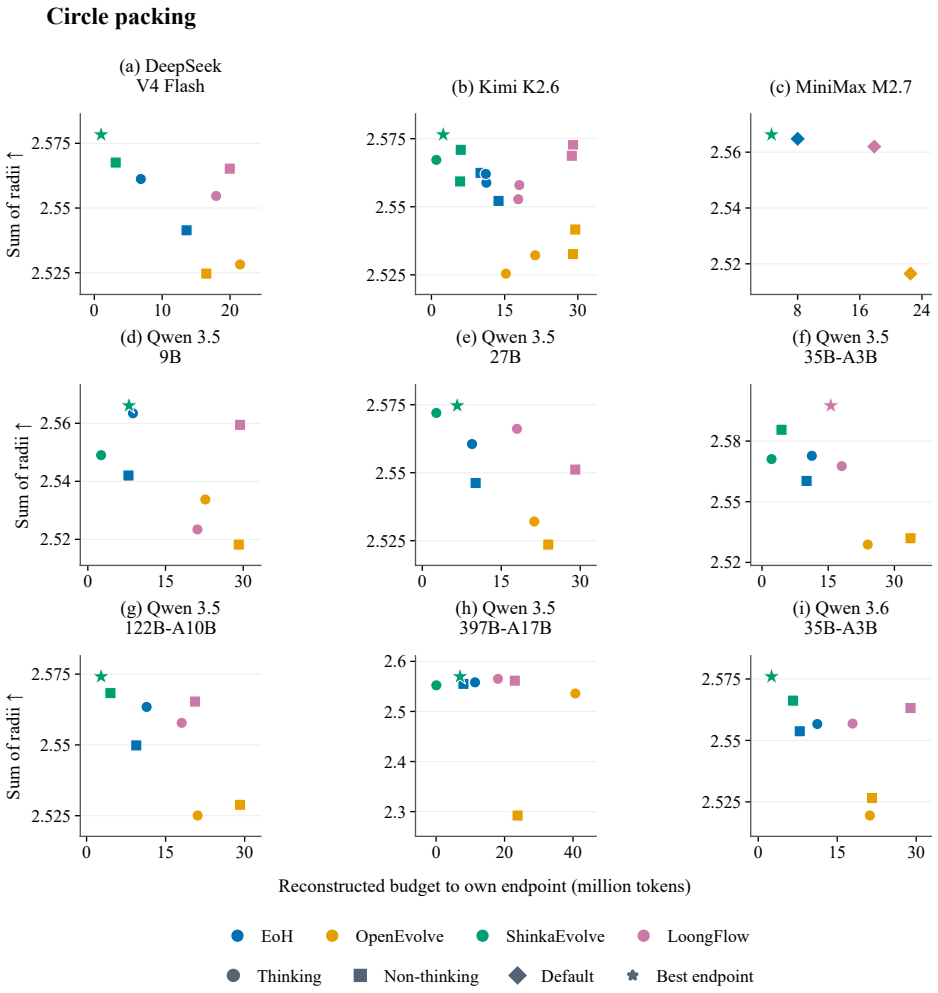

Figure 17: Circle packing. Configuration means for the remaining models.

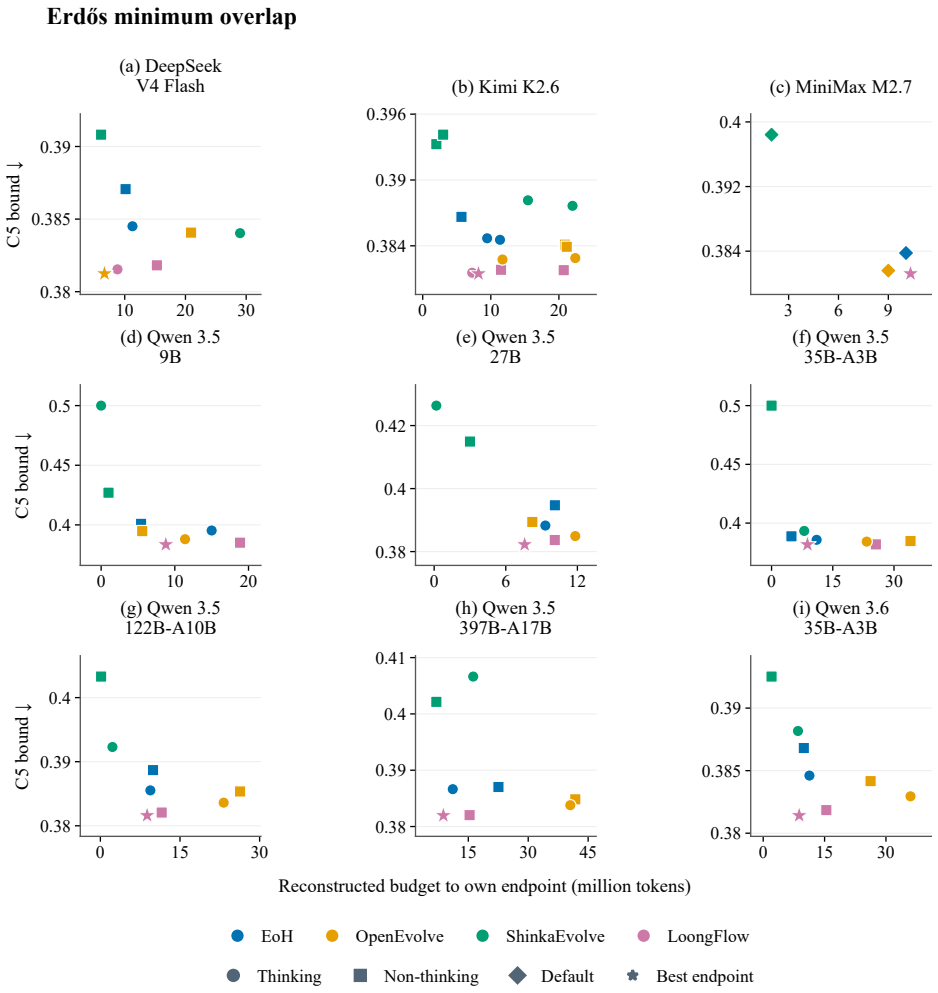

Figure 18: Erdős overlap. Configuration means for the remaining models.

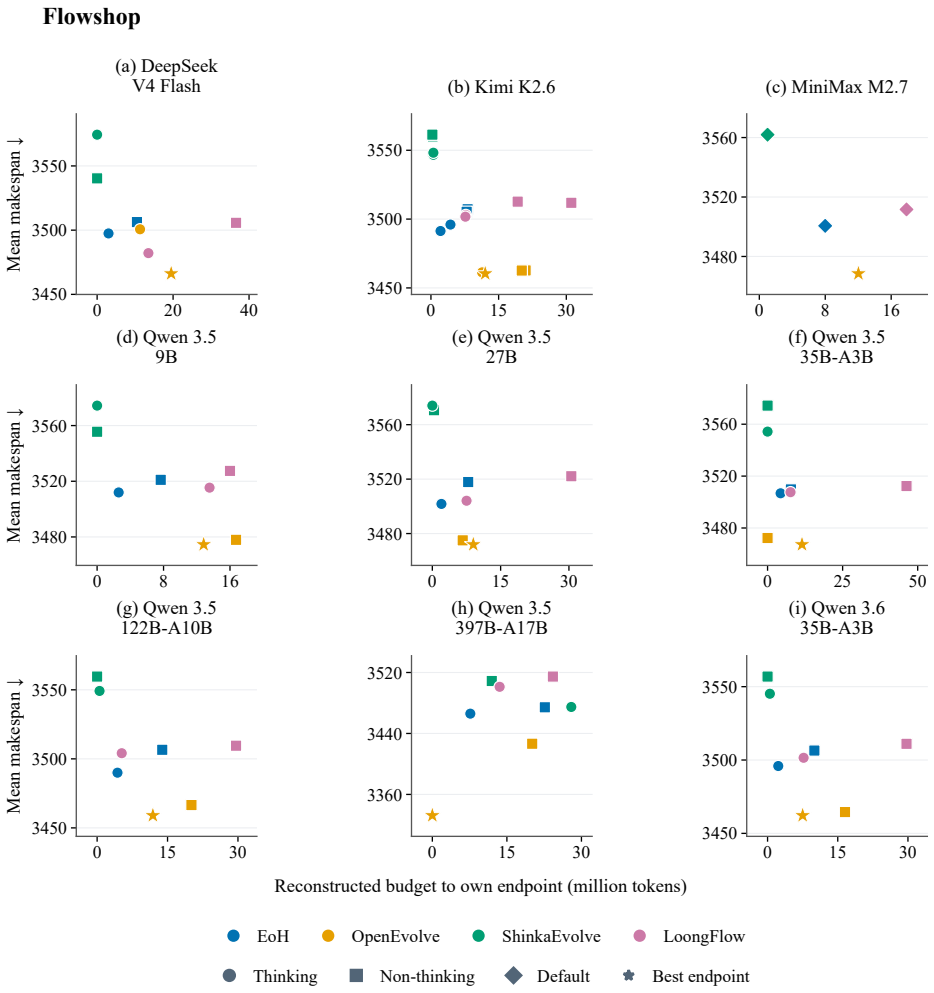

Figure 19: Flowshop. Configuration means for the remaining models.

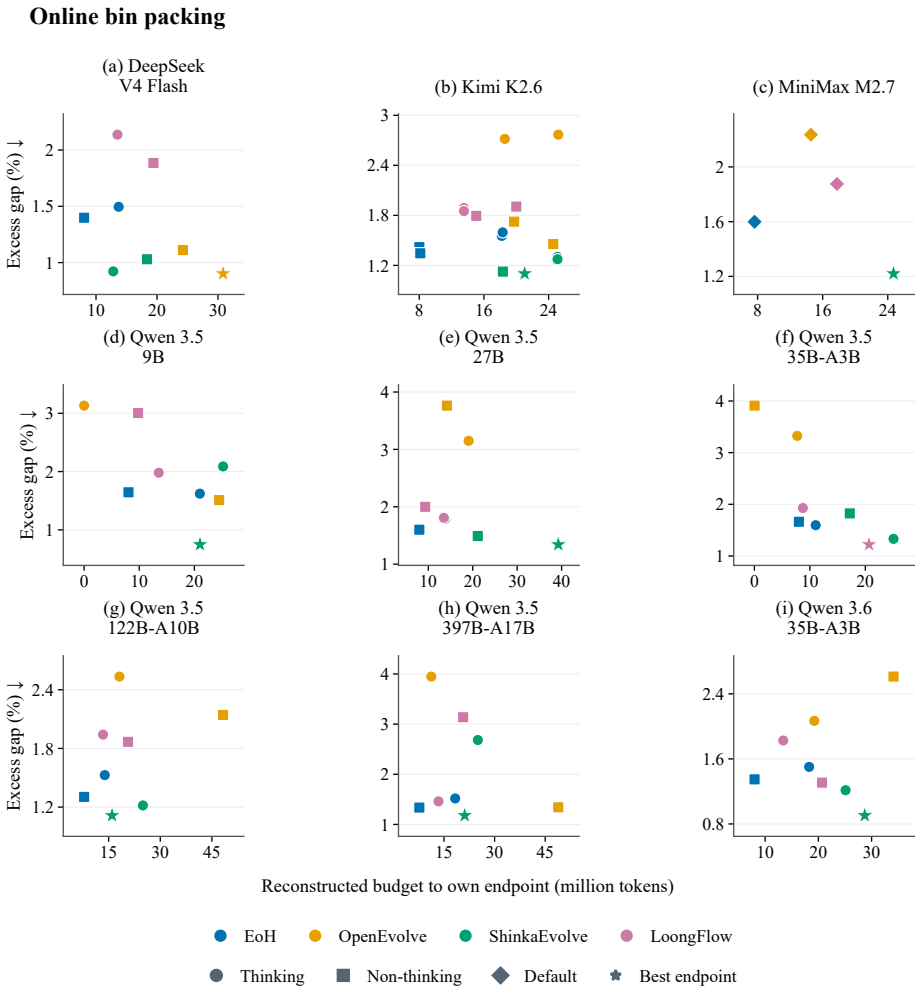

Figure 20: Online bin packing. Configuration means for the remaining models.

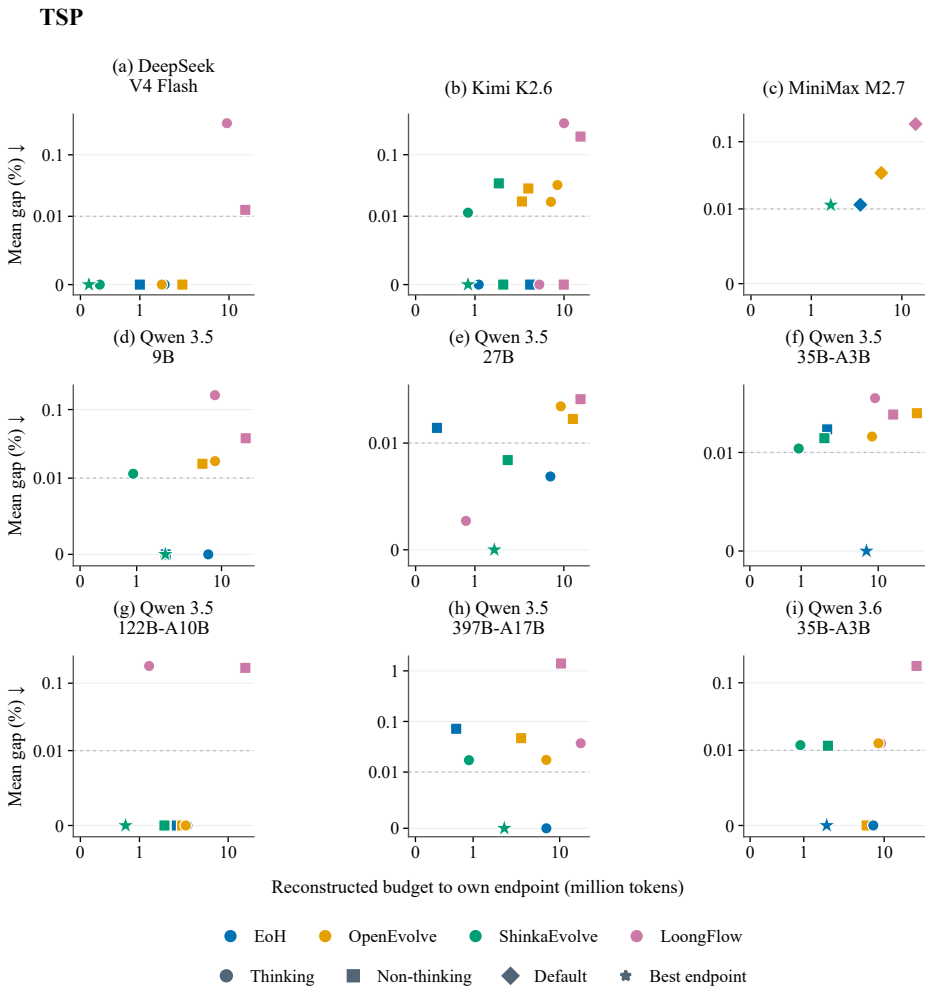

Figure 21: TSP. Configuration means for the remaining models. Symlog axes are linear through 1M tokens and a gap of 0.01%.

H Instance-Response Sensitivity Analysis

Sensitivity scenarios examine flowshop, TSP, and online bin packing, anchored to existing endpoint exports. All additional responses below are simulated, not evaluations of evolved programs. The main benchmark retains five tasks.

Scenario construction. Native scores are transformed as $h = \text{asinh}(y/s)$, with scales of 3500 for flowshop, 0.01 percentage points for TSP, and one percentage point for bin packing. Each scenario response combines a task center, a framework offset multiplied by k, a retained run offset, and shared-instance, framework–instance, and program–instance perturbations scaled to historical within-framework dispersion. Coefficients $k = 1$, 0.5, and -0.5 retain, attenuate, or reverse the transformed framework effect. All scenarios share random draws. Nonnegative back-transformation need not preserve native-scale mean ordering.

Flowshop and TSP. The scenario design uses 61 additional input identities per task and retains 26 configurations and three source runs per framework. Flowshop includes an assumed machine-count adjustment; TSP uses gap percentages directly. Original three-instance aggregates remain separate from scenario responses. Table 25 shows how these assumptions affect ordering: the scenario mean for flowshop favors OpenEvolve at $k = 1$ and ShinkaEvolve at $k = -0.5$. This change follows the scenario coefficient rather than an additional program evaluation.

Table 25: Instance-response sensitivity under the specified simulation model. Entries are scenario means. Flowshop uses an illustrative makespan scale; TSP values are gap percentages. Columns specify the transformed framework-effect coefficient.

Task	Framework	$k = 1$	$k = 0.5$	$k = -0.5$
Flowshop	EoH	3418.55	3420.23	3423.60
Flowshop	LoongFlow	3422.57	3421.13	3418.24
Flowshop	OpenEvolve	3379.41	3400.80	3443.78
Flowshop	ShinkaEvolve	3464.01	3442.17	3398.72
TSP	EoH	0.010037	0.014158	0.026834
TSP	LoongFlow	0.080781	0.055124	0.025277
TSP	OpenEvolve	0.025239	0.023420	0.020132
TSP	ShinkaEvolve	0.009802	0.012084	0.017969

Independent bin-packing streams. Thirty item streams are generated in each of six length/capacity conditions. Relative item sizes follow a specified Weibull distribution with shape 2 and scale 0.5, restricted to $(0, 1]$, then rounded upward after multiplication by capacity. New streams are independent across conditions and shared across frameworks; the original five streams are excluded. Capacity changes preserve the relative-size distribution. Assumed finite-length and capacity-response terms determine scenario performance. Scenario-derived integer bin counts satisfy volume and item-count bounds; no packing assignments are constructed.

Interpretation. Table 26 shows that ShinkaEvolve has the lowest scenario excess in all six conditions at $k = 1$, whereas the ordering changes under attenuation or reversal of the framework effect. Together with Table 25, these results describe sensitivity to specified response assumptions. Changes in ordering follow the assumed framework effects, instance variation, and nonlinear transformation; they do not establish a preferred framework on new instances. The accompanying parameter file and generator specify every coefficient and seed. Scenario quantiles are descriptive summaries, not empirical confidence intervals.

Table 26: Bin-packing sensitivity under the specified simulation model. Mean excess percentages relative to the volume lower bound, using 30 new streams per row. These simulated values are not solver evaluations or verified feasible packing counts.

Items	Capacity	EoH	LoongFlow	OpenEvolve	ShinkaEvolve
Transformed framework-effect coefficient $k = 1$					
5000	100	1.628	2.097	2.312	1.398
1000	100	1.847	2.590	2.759	1.738
10000	100	1.502	1.929	2.169	1.246
50000	100	1.357	1.889	2.065	1.187
5000	50	1.391	1.876	2.374	1.369
5000	200	1.670	2.301	2.104	1.319
Transformed framework-effect coefficient $k = 0.5$					
5000	100	1.731	1.942	2.068	1.610
1000	100	1.955	2.400	2.481	1.984
10000	100	1.599	1.785	1.938	1.442
50000	100	1.448	1.746	1.842	1.377
5000	50	1.484	1.734	2.125	1.579
5000	200	1.775	2.133	1.879	1.523
Transformed framework-effect coefficient $k = -0.5$					
5000	100	1.950	1.659	1.642	2.101
1000	100	2.201	2.067	1.987	2.561
10000	100	1.806	1.519	1.531	1.896
50000	100	1.642	1.485	1.450	1.817
5000	50	1.681	1.474	1.689	2.063
5000	200	1.999	1.829	1.482	1.995