AUTOMATED DISCOVERY HAS NO UNIVERSALLY SUPERIOR HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous discovery systems such as OpenEvolve and TTT-Discover are often used as general-purpose harnesses. However, in practice, these are composite systems combining several design choices about archives, parent selection, exploration, and budget allocation into a single recipe. Because discovery runs are expensive and inherently stochastic, existing harnesses are often compared using too few independent trials to distinguish key methodological improvements from run-to-run variance. We systematically decompose OpenEvolve-style evolutionary search and the TTT-Discover search harness into their constituent components and systematically evaluate 30 budget-matched harnesses across 12 model–problem pairs using more than 3.1 million LLM rollouts and repeated-trial statistical analysis. Our results show that **discovery harnesses have a generalization problem**: No fixed harness is reliably superior across the model–problem pairs, and variants of **OpenEvolve generally underperform** simpler alternatives. Thus, **harness choice is better viewed as a hyperparameter** rather than as a universal recipe, and should be tailored to the problem and underlying model. We also find that early discovery progress predicts final performance, and use this property to present a budget-matched adaptive-allocation experiment that starts multiple harnesses, prunes weak partial runs, and reallocates compute to stronger survivors, outperforming both commitment to sampled fixed harnesses and non-adaptive harness ensembles. **Together, these results motivate shifting from fixed harness selection to online adaptation guided by early performance.** We will release all run pools, including baseline null distributions for every model–problem pair, as statistical infrastructure for evaluating future harness proposals.

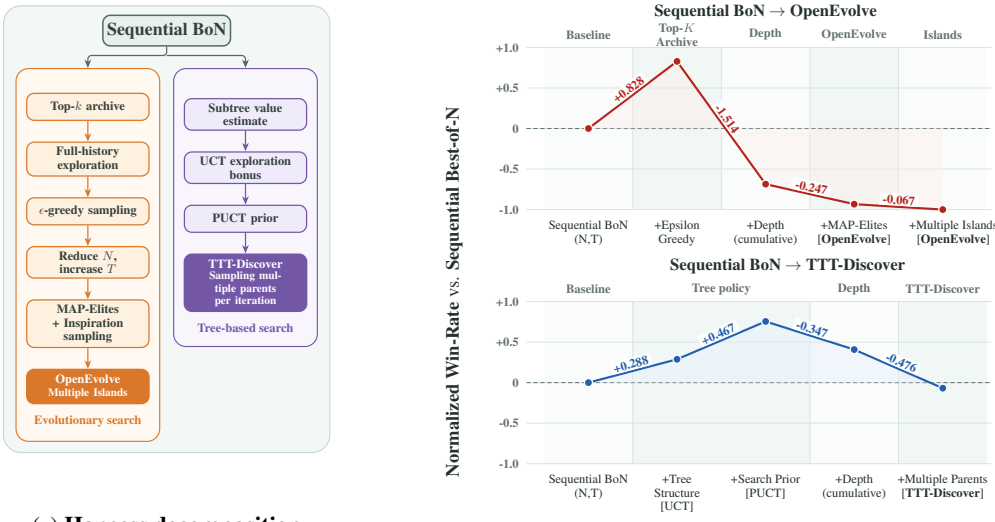
1 INTRODUCTION

LLM-guided discovery systems increasingly use iterative generate–evaluate–improve loops, where a *harness* decides what to expand next from previously evaluated artifacts. Systems such as FunSearch (Romera-Paredes et al., 2024), AlphaEvolve (Novikov et al.), OpenEvolve (Sharma, 2025), and TTT-Discover (Yuksekgonul et al., 2026) bundle many harness choices (archives, exploration, tree search, budget allocation) into composite recipes. While this scaffolding has enabled impressive results, it has also presented a new methodological challenge: existing discovery harnesses are multi-component systems evaluated under substantial stochasticity, yet they are often reported with too few independent trials to separate robust effects from run-to-run variance. As a result, it is often unclear which components actually help, and whether their benefits transfer across models and problems. Our central question in this paper is not whether LLM-guided discovery systems can solve challenging tasks. Recent work has already shown that they can. Instead, we ask a more diagnostic question: *when a discovery system succeeds, do its harness choices transfer as general methodological principles, or are they model- and problem-dependent hyperparameters?*

To answer this question, we decompose two representative harness families: OpenEvolve-style evolutionary search (Sharma, 2025) and TTT-Discover-style PUCT search[1] (Yuksekgonul et al., 2026). Starting from a greedy Sequential Best-of-N (BoN) baseline, we rebuild these harnesses one design choice at a time (Figure 1a) and evaluate 30 budget-matched configurations across 12 model–problem pairs spanning 4 instruction-tuned LLMs (3B–120B parameters) using repeated stochastic

[1] For TTT-Discover, we ablate the search harness, not the test-time-training component.

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.



(a) **Harness decomposition**

(b) **Cross-pair generalization**

Figure 1: **Harnesses in discovery systems are composite recipes whose ingredients do not transfer uniformly.** Panel (a) decomposes two representative discovery harness families into building blocks, starting from greedy sequential Best-of-N. Panel (b) summarizes cross-pair majority-win performance across our 12 model–problem pairs using Sequential BoN as the baseline. Lightweight archive exploration and UCT/PUCT-style search can improve aggregate performance across our suite, but adding the full OpenEvolve-style or TTT-Discover style machinery does not consistently help and can reverse gains from simpler variants.

trials and bootstrap tests against Sequential BoN reference pools[2], resulting in more than 3.1 million rollouts. To assess transfer beyond any setting, we also perform a cross-pair analysis over all 12 model–problem pairs. We will release the full rollout pools (with per-step evaluator scores and baseline null distributions) to enable exact reproduction and future comparisons.

We find that **harnesses for discovery systems have a generalization problem – no single fixed harness is best across all model–problem pairs**, and surprisingly, more complex OpenEvolve-style harnesses generally rank below simpler lightweight-exploration and PUCT-style variants (Figure 1b, Section 4.2). As each harness excels in different settings, we propose to **treat harness choice as a model- and problem-dependent hyperparameter rather than a universally transferable recipe**. This creates a practical dilemma on a new problem: how much budget should be spent repeating a chosen harness, and how much on exploring alternatives? Because early discovery progress strongly predicts final performance (Section 5.1), **harness choice is better made online** – running several harnesses partially, pruning less promising runs by their partial-run evaluator scores, and reallocating that compute to stronger survivors (Section 5.2).

Contributions. We make four main contributions: (i) to our knowledge, we provide the most extensive statistically controlled evaluation of discovery harnesses to date, using more than 3.1 million rollouts to study 30 budget-matched harnesses across 12 model–problem pairs, with repeated stochastic trials and pair-level and cross-pair statistical tests. (ii) evidence that no fixed discovery harness is universally superior across the evaluated model–problem pairs. (iii) a demonstration that choosing among harnesses online as rollouts progress alleviates this generalization problem, motivating discovery systems that adapt their harnesses during problem solving. (iv) a release of 3.1 million rollout records, including per-step evaluator scores and baseline null distributions, as reusable statistical infrastructure for evaluating future discovery harnesses.

2 Unified View of Discovery Harnesses

A *harness* specifies how an LLM-based generate–evaluate loop selects previously evaluated artifacts to expand upon. In this section, we present a unified view of discovery harnesses, focusing on

[2]We create a 100-run baseline pool for Qwen models, and a 30-run baseline pool for GPT-OSS models.

two popular systems: OpenEvolve (Sharma, 2025), based on evolutionary search, and the *search* component of TTT-Discover (Yuksekgonul et al., 2026), which uses PUCT-style parent selection (Silver et al., 2017). To study these, we start from a greedy sequential Best-of-N baseline, and systematically add one design decision at a time to move it towards either the OpenEvolve or TTT-Discover-style search harness. For TTT-Discover, we study only the search harness, not the test-time-training component.

Let x denote a candidate program and $S(x)$ the evaluator score. Let $\mathcal{H}_t$ be the history of evaluated programs at iteration t. At each iteration, a harness selects one (or more) programs from $\mathcal{H}_t$, generates N children, evaluates them, and appends them to the history. For T iterations, the rollout budget is $B = NT$, and we compare harnesses under matched B within each model–problem pair. Full notation details, including elite views and archive sampling are in Appendix B.1.

The Baseline: Greedy Sequential Best-of-N. Sequential BoN always expands the current best program. Let p_t denote the parent selected at iteration t, represented by $p_t = \arg\max_{x \in \mathcal{H}_t} S(x)$. Starting from an initial program, the model generates N candidate modifications from p_t, evaluates them, and updates $\mathcal{H}_{t+1}$. OpenEvolve and TTT-Discover harnesses can be derived from Sequential BoN, which makes this a useful baseline for isolating the contributions of more complex harnesses.

From Sequential BoN to OpenEvolve. OpenEvolve-style evolutionary harnesses can be seen as adding four design choices to Sequential BoN: (i) replace top-1 selection with selecting from a Top-K archive: the set of K programs in $\mathcal{H}_t$ with the highest evaluator scores; (ii) add exploration by sampling from the full history $\mathcal{H}_t$ with probability ϵ; (iii) trade breadth for depth by reducing N and increasing T under fixed B; and (iv) include *inspiration sampling* to add additional programs from $\mathcal{H}_t$ in context, MAP-Elites to maintain high program diversity, and multi-island evolution to parallelize multiple search processes with crossovers. We present experiments for each of the above design decisions that get added on top of Sequential BoN to create OpenEvolve in Section 4.1. Figure 1a summarizes this progression; formal definitions are in Appendix B.2.

From Sequential BoN to TTT-Discover search. Sequential BoN can be seen as the greediest tree policy where parent selection happens without taking into account the subtree of a node. TTT-Discover-style search makes three specific changes: (i) score a node by a value estimate of its subtree (the *maximum* score found so far in that subtree); (ii) add a UCT exploration bonus based on visit counts; and (iii) include a prior term, yielding a PUCT-style rule. The harness may also sample multiple parents per time step. When sampling P parents at a step, we keep B fixed so $B = NT$ counts total evaluated children/rollouts and does not scale with P (Figure 1a).

3 Experimental Setup and Statistical Evaluation

We evaluate harnesses at two complementary levels. First, for each model–problem pair, we test whether a candidate harness produces an unusually strong best-of-five outcome relative to a budget-matched Sequential BoN reference pool. Second, we aggregate these outcomes across all 12 pairs to measure whether the same fixed harness transfers broadly.

3.1 Models, problems and rollout budgets

Models. We run discovery experiments with the following models: Qwen2.5-3B-Instruct (Qwen Team et al., 2025), Qwen3-4B-Instruct-2507 (Yang et al., 2025), GPT-OSS-20B and GPT-OSS-120B (OpenAI et al., 2025). The models span sizes from 3B to 120B parameters, allowing us to study whether harness behavior is consistent across model scales and capabilities.

Problems. We evaluate three mathematical discovery tasks: circle packing, Heilbronn triangle, and the second autocorrelation inequality (Sharma, 2025). These tasks span a range of problem structures, from visually interpretable geometric optimization to a more abstract inequality-based discovery problem. More details on task descriptions are provided in Appendix C.

Rollout Budgets. For each model–problem pair, all harnesses are compared under the same per-run rollout budget, matched by evaluator calls/rollouts. For the Qwen models, each independent discovery run uses 1600 LLM rollouts. The Sequential BoN baseline realizes this budget as 100 iterations with 16 children generated per iteration. For the GPT-OSS models, each run uses 320 and 160 rollouts, respectively, realized in Sequential BoN as 40 iterations with 8 and 4 children per

iteration. Each harness is evaluated with five independent runs. To construct the Sequential BoN reference pool, we use 100 runs for each Qwen model–problem pair and 30 runs for each GPT-OSS model–problem pair. We use the smaller rollout budget and baseline pool for the GPT-OSS models as they were more expensive for inference.

3.2 Best-of-Five Significance Test Against Sequential BoN

We first ask whether a candidate harness produces an observed best-of-five outcome that is unusually strong for a particular model–problem pair. For a given pair, let $\mathbf{s}^0 = (s_1^0, \ldots, s_{n_0}^0)$ be the Sequential BoN reference pool and let $\mathbf{s}^h = (s_1^h, \ldots, s_5^h)$ be five [3] independent runs of harness h. We represent the results for harness h by the best-of-five maximum $T_h = \max(\mathbf{s}^h)$ and estimate a one-sided empirical p-value by bootstrapped resampling from the reference pool: in trial r, we sample five baseline scores with replacement and take $T_r^0 = \max(\mathbf{s}_r^0)$. For each candidate harness, the null hypothesis is that its observed maximum T_h could have arisen from a five-run sample of the Sequential BoN baseline. We therefore test the candidate maximum against the null distribution of five-run baseline maxima:

$$\begin{aligned} &H_0 : T_h \text{ is drawn from the Sequential BoN distribution of best-of-five maxima,} \\ &H_1 : T_h \text{ is larger than expected under that distribution.} \end{aligned} \tag{1}$$

Since the final-score distribution is often skewed, heavy-tailed, non-continuous, and task-dependent, we avoid normality or any parametric assumptions. Instead, we approximate the null distribution of the five-run maximum using bootstrap resampling from the baseline reference pool. For $R = 100{,}000$ bootstrap trials, we sample a five-score tuple $\mathbf{s}_r^0$ with replacement from $\mathbf{s}^0$, compute its maximum $T_r^0 = \max(\mathbf{s}_r^0)$, and estimate the one-sided empirical bootstrap p-value conditional on the observed baseline pool as

$$\widehat{p}_h = \frac{1 + \sum_{r=1}^{R} \mathbf{1}\{T_r^0 \geq T_h\}}{R+1}. \tag{2}$$

Small $\widehat{p}_h$ indicates that the observed best-of-five maximum for h is unusually large relative to the budget-matched Sequential BoN reference distribution for that model–problem pair. When T_h exceeds the maximum observed in the baseline pool, the estimator attains its minimum value $1/(R{+}1)$; we interpret these unadjusted, pair-level p-values as nominal, exploratory evidence conditional on the observed baseline pool. Failure to reject the null does not imply that the candidate harness is incapable of producing stronger outcomes in future runs. Additional statistics (mean/median) and full p-value tables are in Appendix F.

3.3 Evaluating Harness Transfer Across Model–Problem Pairs

Pair-level tests in Section 3.2 identify settings where a harness produces an unusually strong outcome, but they do not measure whether a fixed harness transfers broadly across models and problems. We therefore define a cross-pair majority-win statistic that summarizes how often a harness beats a matched Sequential BoN reference. Let $\mathcal{G} = \{g_1, \ldots, g_{12}\}$ denote the 12 pairs (4 models $\times$ 3 tasks). For harness h, let M_j^h be its observed best-of-five maximum on pair g_j, and let $M_j^{0,r}$ be a matched Sequential BoN best-of-five maximum obtained by bootstrap resampling from the baseline pool in trial r. We define

$$\widehat{P}_{\mathrm{maj}}(h) = \frac{1}{R} \sum_{r=1}^{R} \mathbf{1}\left\{ \sum_{j=1}^{12} \mathbf{1}\left\{ M_j^h > M_j^{0,r} \right\} \geq 7 \right\}, \qquad R = 100{,}000. \tag{3}$$

Thus, $\widehat{P}_{\mathrm{maj}}(h)$ estimates the probability that h strictly beats a matched Sequential BoN outcome on at least 7 of the 12 pairs (ties are not counted as wins).

[3] Our five-run choice is larger than the common three-run protocol used in prior work (Cemri et al., 2026; Wang et al., 2025; Liu et al., 2026; Assumpção et al., 2026)

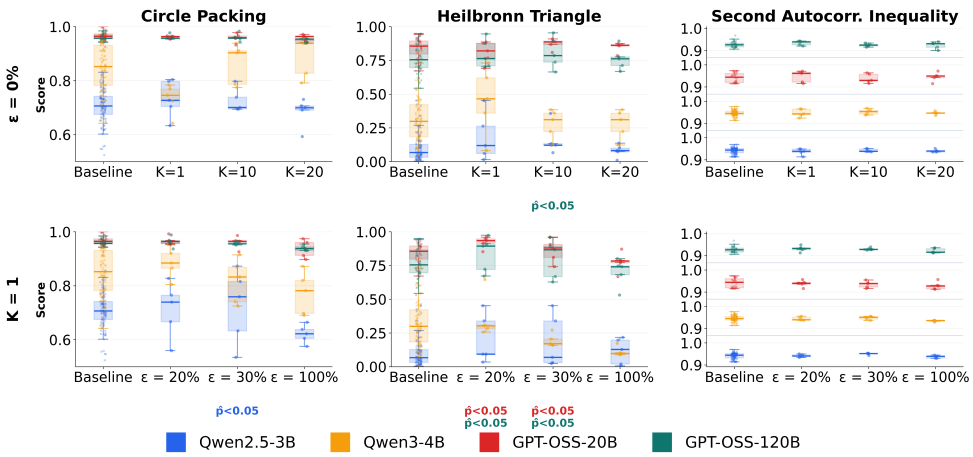


Figure 2: **Archive and Exploration Ablations.** The top row varies only Top-K archive size with $\epsilon = 0\%$; the bottom row fixes $K = 1$ and varies exploration probability ϵ. Nominal pair-level results with $\hat{p}_h < 0.05$ are indicated at the bottom of the plot.

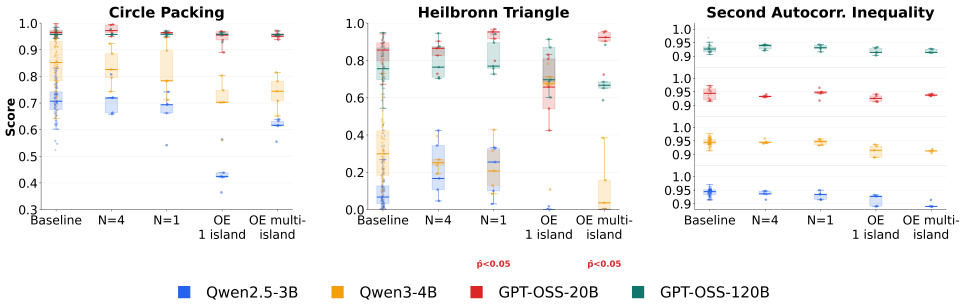


Figure 3: **Progression toward OpenEvolve-style search.** Configurations shift budget from breadth to depth and add MAP-Elites, inspiration sampling, and multiple islands.

Cross-pair aggregate significance test. We complement the descriptive majority-win score with a cross-pair aggregate significance test to quantify if a fixed harness outperforms sequential BoN across the 12 pairs. Let $T_h = \widehat{P}_{\mathrm{maj}}(h)$ denote the observed majority-win statistic for harness h. Under the null hypothesis that h is equivalent to Sequential BoN on every model–problem pair, candidate and baseline outcomes are exchangeable within each pair. We therefore compute a one-sided permutation p-value by repeatedly shuffling, with each pair, which five runs are treated as "candidate" versus "baseline," recomputing the resulting majority win statistic $T_h^{*,r}$(Equation 3), where r represents the permutation trial. We estimate

$$\hat{p}_{\mathrm{cross}}(h) = \frac{1 + \sum_{r=1}^{R} \mathbf{1}\{T_h^{*,r} \geq T_h\}}{R+1}, \tag{4}$$

with $R = 100{,}000$ Monte Carlo trials. We report both nominal p-values and Holm step-down corrected p-values across harness configurations; unless otherwise specified "significant" refers to corrected $\hat{p}_{\mathrm{cross}}(h) < 0.05$.

4 Fixed Harnesses Do Not Generalize Uniformly Across Model–Problem Pairs

We next apply the pair-level and cross-pair analyses from Section 3.3 to study which harness components help and whether they transfer broadly across the 12 evaluated model–problem pairs.

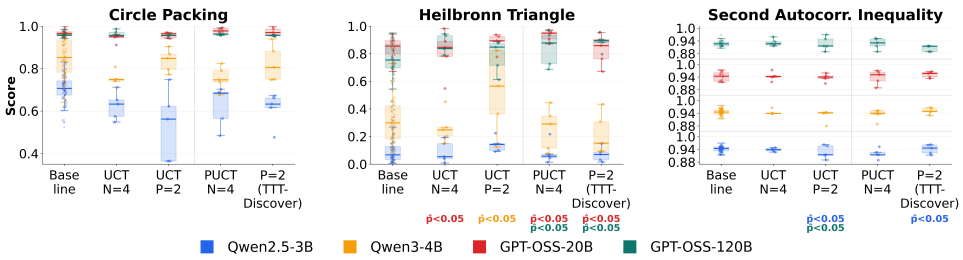


Figure 4: **TTT-Discover-style ablations.** Configurations vary tree-search policy, branching width, and the number of parents expanded per iteration.

4.1 Ablating Harness Components

Archive and Exploration Choices. Figure 2 examines two lightweight ways of adding exploration to Sequential BoN. ϵ denotes the probability of sampling a parent from the full search history $\mathcal{H}_t$, whereas Top-K values represent sampling from the top K solutions so far, which is a separate set from the full history. The top row fixes $\epsilon = 0\%$, which means solutions are not sampled from the full history, and varies the Top-K archive size, allowing the search to revisit the Top-K high-scoring solutions with different values of K. The bottom row fixes $K = 1$ and varies ϵ, which means that the best solution is picked with probability $1 - \epsilon$, and a solution is sampled from $\mathcal{H}_t$ with probability ϵ. Full-history exploration produces nominally significant gains ($\hat{p}_h < 0.05$) for select model–problem pairs, including Qwen2.5-3B on Circle Packing and both GPT-OSS models on Heilbronn Triangle, but lowers performance for others and is not universally significant as we will see in Section 4.2. Joint K–ϵ sweeps are provided in Appendix D, where we similarly see that the $\hat{p}_h$ threshold is reached in selected settings, but its benefits do not transfer uniformly across model–problem pairs.

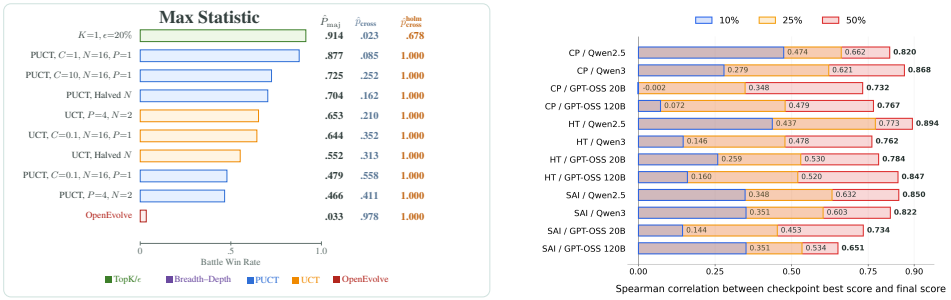
From Sequential BoN to OpenEvolve. We next ask whether progressively adding OpenEvolve components improves discovery under a fixed rollout budget. Starting from the best lightweight exploration configuration, we shift compute from breadth to depth, then add inspiration sampling with MAP-Elites, followed by multi-island evolution. Figure 1b summarizes the cumulative cross-pair trend, while Figure 3 shows the corresponding score distributions. Additional components do not produce monotonic gains: full OpenEvolve-style configurations meet the $\hat{p}_h$ threshold only for GPT-OSS-20B on Heilbronn Triangle while often reducing performance on the other two tasks. **Overall, OpenEvolve machinery does not consistently improve transfer, and the complete recipe underperforms the baseline that strips its components away.**

From Sequential BoN to TTT-Discover Search. We next follow the tree-search branch of Figure 1a, progressively replacing greedy parent selection with UCT, adding the prior-weighted exploration term (PUCT), shifting the fixed rollout budget toward deeper search, and finally expanding multiple parents per iteration as in the search component of TTT-Discover. Figure 1b summarizes the cumulative cross-pair effects, while Figure 4 shows the corresponding pair-level score distributions. UCT/PUCT-based parent selection introduces the largest gains but subsequent changes to search depth and multi-parent expansion do not consistently meet the $\hat{p}_h$ threshold and sometimes reverse those gains.

4.2 Choosing One Fixed Harness

The component ablations show that harness choices can produce significant gains for individual model–problem pairs while hurting others. We therefore ask: *if one fixed harness must be selected without knowing in advance which configuration is best for a given model–problem pair, which one should it be?* We use the cross-pair permutation p-value ($\hat{p}_{\text{cross}}$) defined in Section 3.3.

Figure 5a ranks representative fixed harnesses by $\hat{P}_{\text{maj}}$, with cross-pair p-values before and after Holm's correction. The strongest observed configuration is an epsilon-greedy-style harness with $K = 1$ and $\epsilon = 20\%$, which has $\hat{P}_{\text{maj}} = 0.914$ and $\hat{p}_{\text{cross}} = 0.023$; after Holm's correction it is no longer significant ($\hat{p}^{holm}_{\text{cross}} = 0.678$). In contrast, the complete OpenEvolve-style configurations have the lowest observed majority-win score in the evaluated suite ($\hat{P}_{\text{maj}} = 0.033$; $\hat{p}_{\text{cross}} = 0.978$). Full results are in Appendix E.



(a) **Cross-pair generalization.** (b) **Early predictiveness.**

Figure 5: **Fixed harnesses do not reliably generalize, but early progress is informative.** Panel (a) shows the cross-pair majority-win statistic $\hat{P}_{\text{maj}}$ for representative fixed harnesses across all 12 model–problem pairs along with the cross-pair p-values before and after Holm's correction. Panel (b) shows checkpoint-to-final rank correlation, motivating early-feedback-based allocation.

Together with the pair-level results in Section 4.1, these findings show that the strongest observed fixed choice changes across models and problems; the evidence does not support replacing one universal harness recipe with another.

Finding 1. No tested fixed harness is significantly better than a simple baseline in our evaluation suite, and the strongest observed harness varies across settings.

5 Early Feedback Enables Online Harness Allocation

Section 4 shows that the strongest fixed harness varies across the evaluated model–problem pairs, making it risky to commit the entire discovery budget to one configuration in advance. We therefore study whether early evaluator feedback can support allocating compute across harnesses online.

5.1 Early Discovery Progress Predicts Final Performance

Across discovery runs, we compute the Spearman rank correlations between the best scores recorded at 10%, 25%, and 50% of allocated steps and final best scores. This tests whether configurations that are early leaders remain strong at completion.

Results. Figure 5b shows that correlations are weak-to-moderate at 10% (0.000 to 0.474) but exceed 0.70 for 11 of the 12 model–problem pairs by the 50% checkpoint (and remain positive at 0.651 for GPT-OSS-120B on the Second Autocorrelation Inequality). This indicates that partial-run scores are informative enough to rank partial runs and support pruning and reallocation under a fixed budget.

Finding 2. Partial-run performance becomes increasingly correlated with final performance as search progresses, providing a useful signal at early checkpoints and a strong signal by the halfway point.

5.2 Budget-Matched Online Harness Allocation

Partial-run scores predicting final performance in Section 5.1 motivate an adaptive harness experiment, where we allocate compute online by starting multiple harnesses, pruning them at intermediate checkpoints, and completing the strongest survivors (Gomes & Selman, 2001; Jamieson & Talwalkar, 2015; Li et al., 2018; 2020).

Budgeted policy design. We propose an allocation policy over the fixed harnesses from Section 4 called *Adaptive Harness Ensemble*. We start multiple harness configurations, rank partial runs by

Table 1: **Budget-matched online harness allocation.** We report mean final scores over 100,000 empirical resampling trials per model–problem pair under a budget of five full-run equivalents. Scores are reported as percentages. Winrate is the number of the 12 model–problem pairs on which a method scores higher than the Sequential BoN reference; ties do not count as wins. Higher is better, and bold marks the best value in each score column.

Strategy / schedule	CP Q3B	CP Q4B	CP GPT 20B	CP GPT 120B	HT Q3B	HT Q4B	HT GPT 20B	HT GPT 120B	SA Q3B	SA Q4B	SA GPT 20B	SA GPT 120B	Winrate (/12)
Sequential BoN reference, 5 full	78.42	93.60	98.63	97.56	24.50	**56.05**	91.29	89.80	95.38	96.09	**96.59**	94.27	N/A
Single-harness baseline, 5 full	74.46	89.61	97.77	96.90	25.39	43.90	91.69	89.81	95.06	95.47	95.65	94.20	3/12
Unpruned harness portfolio, 5 full	77.33	93.13	98.37	97.37	29.11	51.08	94.01	91.89	95.36	95.97	96.15	**94.73**	4/12
Single-stage pruning at 25%													
25%: 17→1	77.74	94.78	98.68	97.39	35.82	45.48	93.92	92.57	95.44	96.30	96.03	94.27	7/12
25%: 14→2	78.68	**94.79**	98.72	97.60	**35.93**	48.59	94.60	93.55	95.49	**96.32**	96.26	94.29	**10/12**
Single-stage pruning at 50%													
50%: 9→1	78.13	94.07	98.50	97.58	32.97	50.58	94.25	92.73	95.45	96.17	96.18	94.53	8/12
50%: 8→2	78.55	94.06	98.59	97.59	32.69	52.29	94.65	93.07	95.48	96.18	96.25	94.50	9/12
Two-stage pruning at 25%/50%													
10→6→2	**79.24**	94.39	98.67	97.69	34.03	52.87	94.90	93.40	95.50	96.26	96.32	94.62	**10/12**
12→6→1	78.79	94.48	98.66	97.69	34.73	51.29	94.82	93.44	95.49	96.25	96.23	94.62	**10/12**
14→4→1	78.56	94.67	98.72	97.69	35.58	50.17	95.05	**93.78**	**95.51**	96.28	96.22	94.38	**10/12**
Three-stage pruning at 25%/50%/75%													
8→6→5→1	78.43	94.04	98.50	97.57	32.64	54.21	94.63	92.70	95.47	96.18	96.21	94.71	9/12
10→6→3→1	79.08	94.39	98.63	97.69	34.02	53.62	94.96	93.25	95.49	96.25	96.28	94.66	9/12
12→5→2→1	79.17	94.58	**98.73**	**97.72**	35.07	52.42	**95.11**	93.62	**95.51**	96.28	96.26	94.51	**10/12**

best observed score, and prune weaker runs at one or more checkpoints. We fix the budget at $B_e = 5$ full-run equivalents, matching the paper's best-of-five setting, and each policy returns a final score.

Pruning policies. We define a *multi-stage policy*, which has $L-1$ pruning stages giving us s final survivors. At each stage $\ell \in \{1, \ldots, L-1\}$, m_ℓ partial runs advance to a fraction $q_\ell \in (0,1)$ of a full run, where $0 = q_0 < q_1 < \cdots < q_L = 1$ and $m_1 > \cdots > m_L = s$. The policy is subject to:

$$\sum_{\ell=1}^{L} m_\ell (q_\ell - q_{\ell-1}) \leq B_e. \tag{5}$$

A *single-stage policy* sets $L = 2$, where the sum reduces to $m_1 q_1 + m_2(1-q_1) \leq B_e$. For example, advancing 17 runs to 25% and completing one survivor costs $17(0.25)+1(0.75) = 5$ runs. We prune at $q_l \in \{25\%, 50\%, 75\%\}$ and vary survivor count $s \in \{1,2,3,4\}$. If $s > 1$, we take the highest score among the survivors. We provide algorithms in Appendix A and full results in Appendix G.

Baselines. We compare against (i) five-run *Sequential BoN*, (ii) a *single-harness* baseline that samples one harness and runs it five times, and (iii) an *unpruned ensemble* that samples five harnesses and completes one run of each. The unpruned ensemble is the key control: it draws from the same harness pool as the adaptive policy but does not use checkpoint feedback, so their comparison isolates the benefit of reallocation beyond harness diversity. Figure 6 compares the policies. We exclude the highest-ranked configuration from Section 4.2 because its retrospective ranking does not define a policy under this resampling protocol.

Evaluation. For each model–problem pair, we simulate 100,000 $B_e = 5$ budgets by sampling empirical runs with replacement, applying pruning decisions, and recording the best final score. We compare each policy via head-to-head wins across the 12 pairs.

Results. Table 1 summarizes policy outcomes under the same $B_e = 5$ budget; Appendix Table 13 gives additional schedules. In head-to-head comparisons, every adaptive pruning schedule wins on a majority of the 12 pairs against the five-run Sequential BoN reference. The strongest schedule, $12 \to 5 \to 2 \to 1$ with pruning at 25%, 50%, and 75% wins on 10/12 pairs against Sequential BoN and on 11/12 pairs against the unpruned ensemble. Although the other multi-stage pruning methods reach a similar win-rate against the baseline, we see that $12 \to 5 \to 2 \to 1$ is the only schedule to achieve a strict majority of per-pair wins against every other schedule, as shown in Table 17. Because the adaptive and unpruned ensembles draw from the same harness pool, their gap supports reallocating compute using partial-run feedback. **Overall, the strongest policies begin with broader portfolios and progressively concentrate the budget as additional feedback becomes available.**

Finding 3. Adapting harness choice online improves average final performance over both single-harness commitment and an unpruned harness portfolio under the same budget.

6 Related Work

LLM-guided discovery systems and evolutionary coding agents. LLM-guided discovery systems iterate between proposing, evaluating, and updating candidate programs. FunSearch (Romera-Paredes et al., 2024), LLMs as Optimizers (Yang et al., 2024), AlphaEvolve (Novikov et al.), and TTT-Discover (Yuksekgonul et al., 2026) demonstrate this paradigm for mathematical, algorithmic, and scientific discovery. Many follow-ups add harness ingredients—e.g., novelty filters, adaptive optimization, islands, or inspiration sampling—including OpenEvolve (Sharma, 2025), ShinkaEvolve (Lange et al., 2025), DeltaEvolve (Jiang et al., 2026), AdaEvolve (Cemri et al., 2026), CodeEvolve (Assumpção et al., 2026), DeepEvolve (Liu et al., 2025b), and GigaEvo (Khrulkov et al., 2025); related work also evolves heuristics directly (Ye et al., 2024; Liu et al., 2024; 2025a). Rather than proposing another composite recipe, we study the harness itself: we decompose representative designs and test which ingredients transfer across model–problem pairs under matched budgets.

Evaluation rigor and simple baselines. Method comparisons are often confounded by variance and incomplete reporting: deep RL is seed-sensitive (Henderson et al., 2019), random search is a strong NAS baseline (Li & Talwalkar, 2019; Yang et al., 2020), repeated sampling can be surprisingly effective for LLMs (Brown et al., 2024), and agent benchmarks may omit seeds and costs (Kapoor et al., 2024). In LLM discovery, Gideoni et al. (2026) show that simple baselines can match complex pipelines, Tanveer (2026) show that harness redesign can substitute for a larger mutation model, and Pelleriti et al. (2026) analyze evolutionary agents via annotated traces.

Search algorithms and budgeted allocation. Our ingredient choices connect to UCT (Kocsis & Szepesvári, 2006), PUCT-style search (Silver et al., 2017), and archive-based diversity (MAP-Elites (Mouret & Clune, 2015); Novelty Search (Lehman & Stanley, 2011)). On allocation, we build on algorithm selection/portfolios (Rice, 1976; Gomes & Selman, 2001) and fixed-budget schedulers such as Successive Halving (Jamieson & Talwalkar, 2015), Hyperband (Li et al., 2018), and ASHA (Li et al., 2020). Xing et al. (2026) and Ishibashi et al. (2026) study related depth–breadth tradeoffs, while AdaEvolve's island scheduler (Cemri et al., 2026) allocates within a run; our Adaptive Harness Ensemble allocates *across* harness configurations as a causal test.

Harness adaptation and meta-harness search. Another line treats the harness as a search object. ADAS (Hu et al., 2025), AgentSquare (Shang et al., 2025), and Archon (Saad-Falcon et al., 2025) search modular designs or compose inference-time techniques. In discovery, Meta-Harness (Lee et al., 2026) searches harness programs and EvoX (Liu et al., 2026) meta-evolves strategies. Recent work automates synthesis/modification (Zhang et al., 2026; Lin et al., 2026; Lou et al., 2026; Chen et al., 2026; Ning et al., 2026), tunes feature flags (HARBOR (Sengupta & Wang, 2026)), or adapts online (Continual Harness (Karten et al., 2026)). We explore when fixed harnesses fail to transfer and if early feedback is informative, motivating online allocation over one-shot commitment.

7 Conclusion

Through large-scale, statistically controlled evaluation, we find that fixed discovery harnesses do not reliably generalize across models and problems: components that help in one setting may provide little benefit—or even hurt—in another, and no fixed configuration emerges as reliably superior. Harness choice is therefore better treated as a model- and problem-dependent decision than as a transferable methodological recipe. At the same time, partial-run performance indicates which harnesses are promising; using it to prune weaker harnesses and reallocate their budget improves average final performance over both committing to a single harness and running an unpruned ensemble. This points toward discovery systems that adapt not only their candidate solutions, but also the harness generating them. We release our rollout-level data and statistical artifacts for future research.

USE OF AI TOOLS

AI tools were used in the process of creating this work. We utilized them to help implement code for running experiments and finding related work during the literature review. In addition, we used generative AI tools as writing aids in the drafting stages, including generating figures/images, but the final version of this paper is heavily edited and rewritten by the human authors, and we take full responsibility for the claims and results in this paper. We have not used generative AI for generating synthetic datasets, proving mathematical claims, designing experiments, providing feedback on experiments, or interpreting results. We take full responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The settings for each harness configuration can be found in Appendix F. When a configuration field is omitted in a row, it inherits the Sequential BoN default. All evaluated runs were scored using evaluator code from SkyDiscover, an open-source automated discovery library, which will be in the code we release. Appendix A.1 contains pseudocode for baseline, unpruned, and adaptive harness implementations. More detailed runnable code implementations of our OpenEvolve, PUCT, and other sampling methods will be provided in our released code after double-blind review, along with all 3.1 million LLM rollouts used by our study. All main experiments were run from March to September 2026.

REFERENCES

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. Codeevolve: An open source evolutionary coding agent for algorithmic discovery and optimization, 2026. URL https://arxiv.org/abs/2510.14150.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive llm driven zeroth-order optimization, 2026. URL https://arxiv.org/abs/2602.20133.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems, 2026. URL https://arxiv.org/abs/2606.01779.

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution, 2026. URL https://arxiv.org/abs/2602.16805.

Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1):43–62, 2001. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(00)00081-3. URL https://www.sciencedirect.com/science/article/pii/S0004370200000813. Tradeoffs under Bounded Resources.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019. URL https://arxiv.org/abs/1709.06560.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025. URL https://arxiv.org/abs/2408.08435.

Yoichi Ishibashi, Taro Yano, and Masafumi Oyamada. Effective harness engineering for algorithm discovery with coding agents, 2026. URL https://arxiv.org/abs/2605.15221.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization, 2015. URL https://arxiv.org/abs/1502.07943.

Jiachen Jiang, Tianyu Ding, and Zhihui Zhu. Deltaevolve: Accelerating scientific discovery through momentum-driven evolution, 2026. URL https://arxiv.org/abs/2602.02919.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter, 2024. URL https://arxiv.org/abs/2407.01502.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

Valentin Khrulkov, Andrey Galichin, Denis Bashkirov, Dmitry Vinichenko, Oleg Travkin, Roman Alferov, Andrey Kuznetsov, and Ivan Oseledets. Gigaevo: An open source optimization framework powered by llms and evolution algorithms, 2025. URL https://arxiv.org/abs/2511.17592.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pp. 282–293, 2006.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. 2025. URL https://arxiv.org/abs/2509.19349.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, June 2011. ISSN 1063-6560. doi: 10.1162/EVCO_a_00025. URL https://doi.org/10.1162/EVCO_a_00025.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019. URL https://arxiv.org/abs/1902.07638.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning, 2020. URL https://arxiv.org/abs/1810.05934.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018. URL https://arxiv.org/abs/1603.06560.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

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, 2024. URL https://arxiv.org/abs/2401.02051.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design, 2025a. URL https://arxiv.org/abs/2508.03082.

Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. Scientific algorithm discovery by augmenting alphaevolve with deep research, 2025b. URL https://arxiv.org/abs/2510.06056.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alvin Cheung, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. Evox: Meta-evolution for automated discovery, 2026. URL https://arxiv.org/abs/2602.23413.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness, 2026. URL https://arxiv.org/abs/2603.03329.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015. URL https://arxiv.org/abs/1504.04909.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. Code as agent harness. 2026. URL https://arxiv.org/abs/2605.18747.

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.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card. 2025. URL https://arxiv.org/abs/2508.10925.

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.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

John R. Rice. The algorithm selection problem. In Morris Rubinoff and Marshall C. Yovits (eds.), *Advances in Computers*, volume 15, pp. 65–118. Elsevier, 1976. doi: https://doi.org/10.1016/S0065-2458(08)60520-3. URL https://www.sciencedirect.com/science/article/pii/S0065245808605203.

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://doi.org/10.1038/s41586-023-06924-6.

Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, E. Kelly Buchanan, Mayee Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. Archon: An architecture search framework for inference-time techniques, 2025. URL https://arxiv.org/abs/2409.15254.

Biswa Sengupta and Jinhua Wang. Harbor: Automated harness optimization, 2026. URL https://arxiv.org/abs/2604.20938.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space, 2025. URL https://arxiv.org/abs/2410.06153.

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

Temoor Tanveer. Levi: Stronger search architectures can substitute for larger llms in evolutionary search, 2026. URL https://arxiv.org/abs/2605.09764.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Thetaevolve: Test-time learning on open problems, 2025. URL https://arxiv.org/abs/2511.23473.

Sixue Xing, Haoyu He, Kerui Wu, Zhuo Yang, Haozheng Luo, Tianfan Fu, and Aarthy Nagarajan. Compute allocation for self-evolving llms: From depth-breadth to multi-armed bandits, 2026. URL https://arxiv.org/abs/2605.29268.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard, 2020. URL https://arxiv.org/abs/1912.12522.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024. URL https://arxiv.org/abs/2309.03409.

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, 2024. URL https://arxiv.org/abs/2402.01145.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026. URL https://arxiv.org/abs/2606.09498.

APPENDIX

A ONLINE BUDGET ALLOCATION ALGORITHMS

A.1 BUDGET CONSTRAINTS AND PRUNING SCHEDULES

We measure compute in full-run equivalents and set $B_e = 5$. A single-stage policy starts m runs, advances them to checkpoint $q \in (0, 1)$, retains the s highest-scoring partial runs, and completes those survivors. Its cost satisfies

$$mq + s(1 - q) \leq B_e. \tag{6}$$

Increasing the survivor count leaves less budget for initial exploration. For a multistage policy, let $0 = q_0 < q_1 < \cdots < q_L = 1$ define the checkpoint intervals and let m_ℓ be the number of active runs advanced over interval $(q_{\ell-1}, q_\ell]$. The total cost is

$$\sum_{\ell=1}^{L} m_\ell (q_\ell - q_{\ell-1}) \leq B_e. \tag{7}$$

The evaluated schedules prune at 25%, 50%, and/or 75% of a full run, with one to four final survivors. For example, the $12 \to 5 \to 2 \to 1$ schedule costs $(12 + 5 + 2 + 1)/4 = 5$ full-run equivalents. At each checkpoint, runs are ranked only by their best score observed so far. These policies reallocate effort across complete harness configurations rather than changing the search rule within a surviving run.

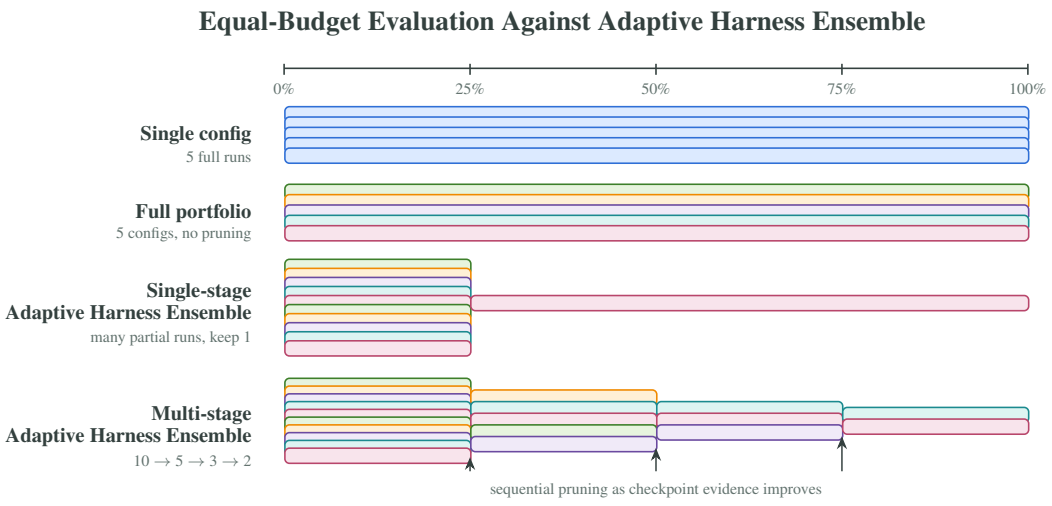


Figure 6: **Adaptive allocation schematic.** The comparison is budget-matched across fixed repeated-run baselines, a non-adaptive full ensemble, single-stage Adaptive Harness Ensemble, and multi-stage Adaptive Harness Ensemble. Starting with the full-ensemble baseline, different hues denote distinct search configurations; in the Adaptive Harness Ensemble rows, retained configurations keep their hue after pruning.

Algorithm 1 Single-Harness Baseline

Require: Harness pool $\mathcal{H}$; run budget $R = 5$
Ensure: Best final score $s^\star$
Sample one harness $h \sim \mathrm{Uniform}(\mathcal{H})$
$S \leftarrow \emptyset$
for $r = 1, \ldots, R$ **do**
    $s \leftarrow \textsc{RunToCompletion}(h)$
    $S \leftarrow S \cup \{s\}$
end for
$s^\star \leftarrow \max S$
return $s^\star$

Algorithm 2 Unpruned Harness Ensemble

Require: Harness pool $\mathcal{H}$; run budget $R = 5$
Ensure: Best final score $s^\star$
Sample ensemble $\mathcal{P} \subseteq \mathcal{H}$ with $|\mathcal{P}| = R$
$S \leftarrow \emptyset$
for all $h \in \mathcal{P}$ **do**
    $s \leftarrow \textsc{RunToCompletion}(h)$
    $S \leftarrow S \cup \{s\}$
end for
$s^\star \leftarrow \max S$
return $s^\star$

Algorithm 3 Adaptive Harness Ensemble

Require: Harness pool $\mathcal{H}$; checkpoints $(c_1, \ldots, c_K)$; survivor counts $(m_0, m_1, \ldots, m_K)$
Ensure: Best final score $s^\star$
Sample initial active set $\mathcal{A} \subseteq \mathcal{H}$ with $|\mathcal{A}| = m_0$
for $k = 1, \ldots, K$ **do**
    for all $h \in \mathcal{A}$ **do**
        Run h until checkpoint c_k
        $q_h \leftarrow \textsc{CurrentBestScore}(h)$
    end for
    $\mathcal{A} \leftarrow \textsc{TopM}(\mathcal{A}, \{q_h : h \in \mathcal{A}\}, m_k)$
end for
$S \leftarrow \emptyset$
for all $h \in \mathcal{A}$ **do**
    $s \leftarrow \textsc{RunToCompletion}(h)$
    $S \leftarrow S \cup \{s\}$
end for
$s^\star \leftarrow \max S$
return $s^\star$

B Detailed Search-Harness Definitions

B.1 Search-Harness Ablation Details

Section 4 presents the ablations as two compact progressions: Sequential BoN → archive search → exploration → OpenEvolve-style refinement, and Sequential BoN → UCT → PUCT. The appendix expands each step with the full task-level sweeps, model-specific results, and implementation details needed to interpret the compressed main-text figures. Table 2 provides a guide to this material.

The detailed sweeps diagnose individual mechanisms, whereas Section 4.2 asks whether a complete configuration transfers across model–problem pairs. The two analyses therefore answer comple-

Table 2: **Guide to the appendix ablations.** Each row links a compressed main-text comparison to its complete task-level evidence.

Main-text step	Detailed figures	What the detailed view adds
Sequential BoN baseline	Figure 12	Baseline score distributions by task and model, used to interpret later bootstrap comparisons.
Top-K archive	Figure 13	Full task-level archive-size sweep with significance markers against Sequential BoN.
General-history exploration	Figure 14	Full ϵ-probability sweep with $K = 1$, showing where exploration helps or destabilizes search.
Full epsilon-greedy grid	Figures 11a, 11b, and 11c	Interaction between elite archive size and ϵ probability across each task.
OpenEvolve progression	Figure 10	Breadth–depth and island-style refinements showing that OpenEvolve-style search is competitive but not uniformly dominant.
UCT/PUCT family	Figures 15, 16, and 17	Detailed tree-search knob sweeps for breadth–depth, number of parent programs, and exploration constant.

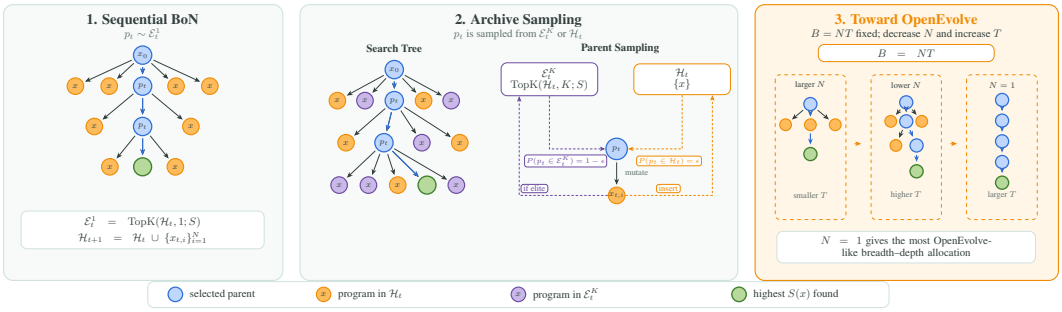


Figure 7: **Progression from Sequential BoN to OpenEvolve-style search.** Sequential BoN samples the parent from the top-1 elite archive $\mathcal{E}_t^1$. Archive sampling generalizes this to the top-K elite archive $\mathcal{E}_t^K$ and, with probability ϵ, the full history $\mathcal{H}_t$; each generated child is inserted into $\mathcal{H}_{t+1}$ and also enters $\mathcal{E}_{t+1}^K$ if it is elite. Holding the total generation budget $B = NT$ fixed and decreasing N shifts search toward longer refinement horizons, with $N = 1$ giving the most OpenEvolve-like breadth–depth allocation.

mentary questions: the ablations identify where particular design choices help or hurt, and the battle analysis measures whether those gains persist across settings.

B.2 From Sequential BoN to OpenEvolve

This section explains how we can systematically build towards OpenEvolve from a simple Sequential BoN baseline.

B.2.1 Adding a Top-K Elite Archive

Sequential BoN keeps only the current best program active, which corresponds to sampling from $\mathcal{E}_t^1$. A direct generalization is to sample parents from the Top-K elite archive,

$$\mathcal{E}_t^K = \mathrm{TopK}(\mathcal{H}_t, K, S), \qquad p_t \sim \mathcal{E}_t^K. \tag{8}$$

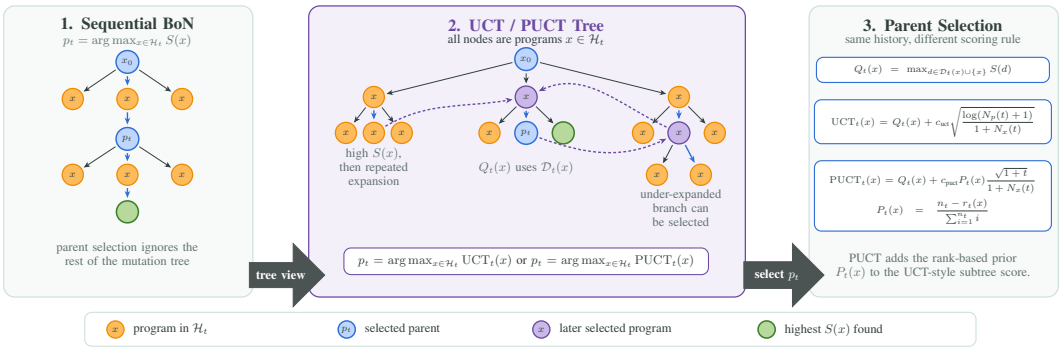


Figure 8: **UCT/PUCT as tree-based search.** Sequential BoN selects p_t using only the highest evaluator score in the full history $\mathcal{H}_t$. UCT and PUCT instead score each program $x \in \mathcal{H}_t$ using the induced mutation tree: $Q_t(x)$ propagates the best score in the descendant set $\mathcal{D}_t(x)$, visitation counts adjust exploration, and PUCT additionally uses the rank-based prior $P_t(x)$. The LLM generation step and history update remain the same after p_t is selected.

The case $K = 1$ recovers Sequential BoN, while $K > 1$ allows the search to revisit several high-scoring programs rather than committing all future mutations to a single best-so-far lineage. This preserves a small amount of diversity while still focusing compute on programs that have already achieved high evaluator scores.

B.2.2 Adding Epsilon-Greedy Exploration

The elite archive can still become too greedy when its high-scoring programs represent closely related lineages. To permit occasional lineage changes, OpenEvolve mixes elite-archive sampling with sampling from the full history. Let ϵ denote the exploration probability. The parent-selection rule is

$$p_t \sim \begin{cases} \mathcal{E}_t^K, & \text{with probability } 1-\epsilon, \\ \mathcal{H}_t, & \text{with probability } \epsilon. \end{cases} \tag{9}$$

When $K = 1$ and $\epsilon = 0$, this reduces to Sequential BoN. When $K > 1$, exploitation samples among the Top-K programs, and when $\epsilon > 0$, exploration samples uniformly from all previously evaluated programs in $\mathcal{H}_t$. This separates two sources of non-greediness: keeping multiple elite lineages through K, and occasionally revisiting non-elite historical programs through ϵ.

B.2.3 Trading Breadth for Depth

The archive and epsilon-greedy variants generate N children from each selected parent before updating the history. OpenEvolve-style search is more depth-oriented: it generates fewer children per expansion and performs more sequential updates. We compare these regimes at a fixed rollout budget, $B = NT$. Decreasing N therefore requires increasing T. Larger N spends more budget exploring alternatives around one parent, whereas larger T permits longer chains of refinement. The setting $N = 1$ is the most depth-oriented and most closely matches the OpenEvolve allocation.

B.2.4 OpenEvolve-Specific Context and Parallelism

After adding elite-archive sampling, epsilon-greedy exploration, and depth-oriented budget allocation, the remaining OpenEvolve components modify the context and parallelization structure rather than the basic history update rule:

1. **MAP-Elites inspiration sampling.** In addition to the selected parent p_t, OpenEvolve provides the LLM with a small set of inspiration programs sampled from a MAP-Elites grid over past solutions. In our notation, this augments the context with additional programs from $\mathcal{H}_t$ while leaving the parent-selection rule unchanged.
2. **Multi-island evolution and migration.** OpenEvolve runs several searches in parallel rather than maintaining a single history. Each island maintains its own history and elite

archive, and high-scoring programs are periodically migrated between islands. This corresponds to applying the archive update rule to several histories with occasional exchange of elite programs.

B.3 From Sequential BoN to TTT-Discover

The first step in this progression is Upper Confidence Bounds applied to Trees (UCT) (Kocsis & Szepesvári, 2006). UCT replaces the node-local score $S(x)$ with a value estimate for the entire subtree rooted at x. Let $\mathcal{D}_t(x)$ denote the descendants of program x in the mutation tree induced by $\mathcal{H}_t$. Following the evolutionary-search formulation used in Yuksekgonul et al. (2026), we propagate the best descendant score rather than the average rollout return used in conventional Monte Carlo Tree Search:

$$Q_t(x) = \max_{d \in \mathcal{D}_t(x) \cup \{x\}} S(d). \tag{10}$$

UCT then selects the next parent by adding an exploration bonus based on visitation counts,

$$\begin{aligned} \mathrm{UCT}_t(x) &= Q_t(x) + c_{\text{uct}} \sqrt{\frac{\log(N_p(t)+1)}{1+N_x(t)}} \\ p_t &= \arg\max_{x \in \mathcal{H}_t} \mathrm{UCT}_t(x), \end{aligned} \tag{11}$$

where $N_x(t)$ is the number of times program x has been expanded, $N_p(t)$ is the visitation count of its parent node, and c_{uct} controls the strength of exploration. Relative to Sequential BoN, UCT makes two changes. First, a parent is credited for the best solution discovered anywhere in its descendant subtree. Second, repeatedly expanded nodes receive a smaller exploration bonus, allowing the search to revisit branches that are promising but not yet exhausted. Once UCT selects p_t, the LLM generation step and history update are unchanged.

TTT-Discover (Yuksekgonul et al., 2026) adds a rank-based prior over archived programs to estimate which nodes are likely to be useful parents. For $n_t = |\mathcal{H}_t|$, let $r_t(x) \in \{0, \ldots, n_t - 1\}$ denote the rank of program x in decreasing order of evaluator score. The PUCT score is

$$\begin{aligned} \mathrm{PUCT}_t(x) &= Q_t(x) + c_{\text{puct}} P_t(x) \frac{\sqrt{1+t}}{1+N_x(t)} \\ P_t(x) &= \frac{n_t - r_t(x)}{\sum_{i=1}^{n_t} i} \\ p_t &= \arg\max_{x \in \mathcal{H}_t} \mathrm{PUCT}_t(x), \end{aligned} \tag{12}$$

where $P_t(x)$ assigns larger prior mass to higher-ranked programs and c_{puct} controls the strength of the prior-weighted exploration bonus. Sequential BoN, UCT, and PUCT all select a parent from the evaluated history, but differ in the score used for that selection: node-local evaluator score for Sequential BoN, subtree value plus a visit-count bonus for UCT, and subtree value plus a prior-weighted bonus for PUCT.

C Task Details and Qualitative Strategy Analysis

C.1 Task Definitions

Figure 9 shows representative examples from each task.

- **Circle Packing.** The goal is to fit 26 non-overlapping circles inside a unit square while maximizing the sum of their radii. This task provides an intuitive geometric optimization setting in which candidate improvements can be inspected visually.

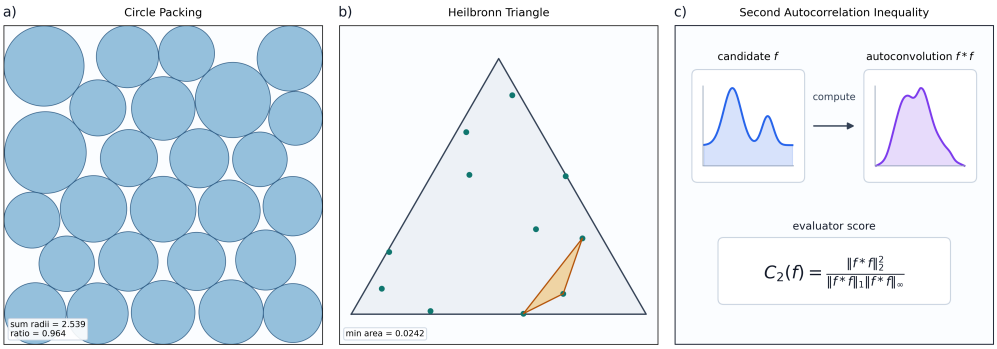


Figure 9: **Representative discovery tasks.** The tasks include two geometric optimization problems and one non-geometric autocorrelation objective.

- **Heilbronn Triangle.** The goal is to place points inside an equilateral triangle while maximizing the minimum area of any triangle formed by three points. The objective is globally coupled because moving one point can affect many of the induced triangles.
- **Second Autocorrelation Inequality.** The goal is to construct a non-negative function, represented numerically on a fixed grid, that performs well under an autocorrelation-based objective. This task tests whether search strategies that work on geometric problems transfer to a more abstract, non-geometric setting.

For the second autocorrelation inequality task, a candidate program defines a vector of non-negative values representing a discretized function. The evaluator computes the discrete autoconvolution of this vector as a numerical approximation to the continuous autoconvolution

$$(f * f)(x) = \int_{\mathbb{R}} f(t) f(x - t)\, dt. \tag{13}$$

The resulting vector is scored using a discretized version of

$$C_2(f) = \frac{\|f * f\|_2^2}{\|f * f\|_1 \|f * f\|_\infty}. \tag{14}$$

The search therefore seeks a non-negative vector whose autoconvolution has a favorable balance among its L_1, L_2, and L_∞ norms. The reported task score is a benchmark-normalized version of this quantity.

C.2 Qualitative Strategies on Circle Packing

The circle packing task reveals several qualitative strategy transitions. Because its constraints are local and pairwise, all four models can improve geometric templates, but they differ in whether they treat the task as direct construction or as an explicit optimization problem.

Qwen2.5-3B. Qwen2.5-3B mostly remains in a "better template plus better local heuristic" regime. It often starts with a ring-style placement of circle centers and attempts to improve it through clipping changes, simple spacing adjustments, and greedy radius-update rules rather than reformulating the task as a solver-based search problem. The first score boundary, around 0.63, appears to be a template-viability wall: successful crossings usually come from tightening clipping bounds and stabilizing naive overlap handling. Around 0.68, the model reaches a heuristic-refinement regime in which modest improvements in spread, clipping, and radius updates can help. Roughly 0.73 is a common plateau. Above this range, the model still tends to edit the packing directly instead of discovering the stronger abstraction that centers should be searched systematically and radii solved globally.

Qwen3-4B. Qwen3-4B shows stronger geometric reasoning. Compared with Qwen2.5-3B, it more often abandons weak ring templates in favor of hexagonal, staggered-row, symmetric, or carefully balanced layouts. The broad 0.77–0.84 region is therefore less a single sharp wall than a transition from weaker constructions to denser row and hex templates. Around 0.91, successful crossings

are commonly associated with stronger row or hex geometry, better balance across rows, and more deliberate structured layouts. Near 0.95, Qwen3-4B reaches its main ceiling: it can design strong dense packings, but usually does not make the next conceptual leap to linear-programming-based radius solving combined with systematic center search. Thus, Qwen3-4B largely replaces random jitter with near-optimal-looking geometric templates, but still tends to engineer the construction itself rather than formalize the underlying optimization problem.

GPT-OSS-20B. GPT-OSS-20B is the first model that consistently treats circle packing as an optimization problem rather than only a construction problem. Its strongest solutions decompose the task into choosing candidate centers, solving the best radii globally, and then searching over center positions using hill climbing, annealing, restarts, or related local-search procedures. Around the 0.95 boundary, explicit optimization begins to matter substantially, with successful crossings adding stronger search behavior on top of already reasonable layouts. The clearest high-score wall appears near 0.9645, where many runs plateau until they combine LP-based radius solving with better center search, stronger hexagonal or staggered seeds, and larger restart or annealing budgets. Above this wall, near 0.98, the final improvements appear to come from refining the same decomposition rather than switching to a new winning mechanism: LP-based radius solving paired with stronger center search, better seeds, and longer annealing schedules. This decomposition of choose centers, solve radii, then search centers, is the key abstraction that allows GPT-OSS-20B to keep improving after the smaller models plateau.

GPT-OSS-120B. GPT-OSS-120B follows the same solver-like pattern as GPT-OSS-20B on circle packing: search over centers, then solve the induced radii problem with an LP. Its high-scoring runs mostly differ in reliability rather than in a new conceptual mechanism, adding larger annealing budgets, occasional swaps, and small-step refinement around the same center-search abstraction. The distribution is tight near the ceiling with only occasional low outliers, suggesting that most configurations find this regime but vary in how often center search gets stuck.

C.3 Qualitative Strategies on Heilbronn Triangle

The Heilbronn triangle task also reveals qualitative differences in the constructions discovered by each model. These differences help explain the separated baseline distributions in Figure 12b.

Qwen2.5-3B. Qwen2.5-3B primarily relies on shallow, local modifications. Its successful solutions are often based on random perturbations, edge or vertex placement heuristics, and simple triangle-aware sampling. The observed score boundaries suggest that moving past roughly 0.096 often requires placing points on edges or vertices, while crossing roughly 0.115 typically requires some form of triangle-aware sampling. In the 0.20–0.45 range, successful crossings are dominated by random-perturbation heuristics. This pattern suggests that Qwen2.5-3B improves mostly through micro-optimizations and stochastic nudges rather than through large structural changes in the point configuration.

Qwen3-4B. Moving from Qwen2.5-3B to Qwen3-4B produces a visible shift from random jitter toward more structured geometric reasoning. Qwen3-4B more often proposes row, ring, lattice, or symmetric triangular layouts. Around the 0.5 score boundary, row and ring lattice constructions become common, and near 0.61 many successful crossings appear to rely on known symmetric lattice templates. At higher thresholds, around 0.72, the successful configurations increasingly resemble edge/interior templates. One possible interpretation is that the larger Qwen model has stronger access to geometric priors or memorized construction patterns, allowing it to abandon purely random perturbation in favor of near-structured triangular arrangements with small degeneracy-breaking offsets.

GPT-OSS-20B. GPT-OSS-20B shows the clearest evidence of phase transitions in strategy. In the lower and middle score ranges, especially around boundaries such as 0.201, 0.372, and 0.384, successful crossings are often associated with introducing an explicit objective optimizer. At the high end, around 0.865, 0.893, and 0.925, the model more frequently relies on annealing-style updates and random restarts. This suggests a two-stage behavior: GPT-OSS-20B first uses direct optimization to reach the mid-range, then switches to temperature-based acceptance and restart budgets to

escape local minima. These strategy transitions suggest that performance gains are not merely due to generating more samples. Stronger models appear to discover qualitatively different algorithmic mechanisms for crossing specific score thresholds.

GPT-OSS-120B. GPT-OSS-120B treats Heilbronn triangle as a bottleneck-repair problem. Its strongest runs start from structured triangular-lattice or edge/interior configurations, then use long simulated-annealing searches to improve them. A common mechanism is to identify the current minimum-area triangle and preferentially move one of its points, while still allowing random moves or larger jumps to escape local minima. Some runs add deterministic local refinement, such as pattern search or moving a bottleneck vertex orthogonally to its opposite side. This makes GPT-OSS-120B look less like pure random restart search and more like targeted repair of the triple that currently controls the objective.

C.4 Qualitative Strategies on the Second Autocorrelation Inequality

Unlike the geometry tasks, this task improves an optimization program rather than a visible construction. Successful programs represent the function as a non-negative vector, compute its autoconvolution with an FFT, and optimize the vector with gradients. The main strategic differences concern the initial function shape, the discretization resolution, and the numerical choices that determine whether optimization reaches a useful solution.

Qwen2.5-3B. Qwen2.5-3B mostly scales the baseline optimizer rather than redesigning it. It increases the number of vector entries and gradient steps, then gradually replaces random or flat initializations with piecewise profiles containing zero regions, plateaus, and ramps. Its progression is therefore from direct coordinate optimization toward finer discretization with a modest prior over the initial function shape.

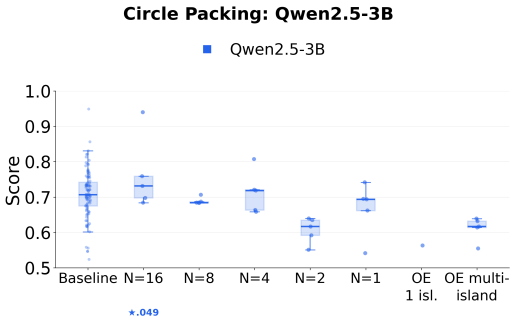
Qwen3-4B. Qwen3-4B introduces stronger structural hypotheses. Its programs often begin with symmetric piecewise-constant functions and evolve toward segmented or self-similar profiles with non-uniform amplitudes. Some lineages also relax symmetry to test asymmetric solutions. The key transition is from treating the vector as independent coordinates to treating it as a structured function whose segments, amplitudes, and symmetry can be designed before gradient refinement.

GPT-OSS-20B. GPT-OSS-20B focuses on conditioning the optimization process. It changes positivity transforms, learning rates, clipping, regularization, and initialization so that the vector can develop sharp step-like regions without unstable gradients. This also explains its broad, approximately bimodal baseline distribution. All 30 plotted runs use the same harness and initial code, but they separate into a smooth low-score basin and a sharper high-score basin. Low-mode programs typically use small learning rates and strong stabilization, while high-mode programs make a larger mutation that relaxes these constraints or introduces a stronger step-like initialization. All final programs with learning rate at least 0.1 enter the high mode, whereas most programs with learning rate at most 0.02 remain in the low mode. This suggests that GPT-OSS-20B alternates between two optimizer regimes rather than gradually moving through one continuous family of solutions.

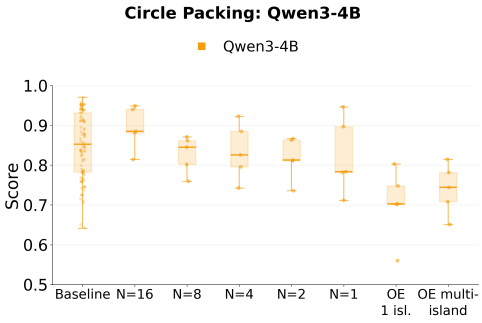
GPT-OSS-120B. GPT-OSS-120B is qualitatively similar to GPT-OSS-20B on the second autocorrelation inequality. Its strongest runs again condition the optimizer toward the sharper step-like basin using step-like initializations, positivity transforms, learning-rate schedules, and higher-resolution discretizations. The main difference is not a new strategy, but a slightly more stable version of the same optimizer-design pattern. The distribution is fairly concentrated, with a few isolated low failures, suggesting that GPT-OSS-120B preserves the same smooth-versus-step-like basin structure rather than discovering a distinct mechanism.

D Full Search-Harness Ablation Results

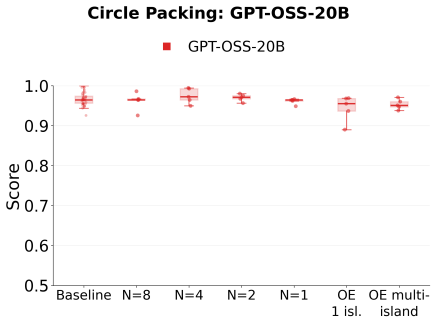
D.1 Additional OpenEvolve Progression Results



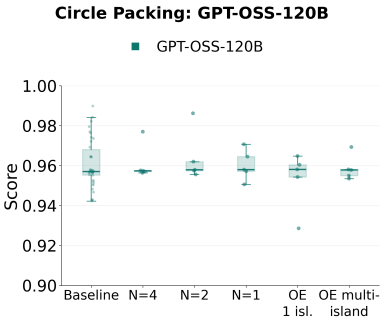
(a) Qwen2.5-3B-Instruct.



(b) Qwen3-4B-Instruct.

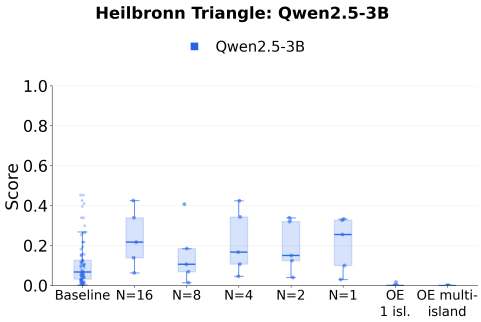


(c) GPT-OSS-20B.

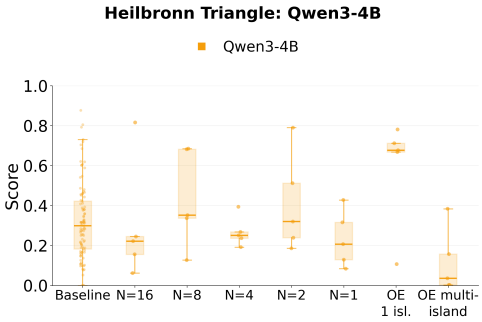


(d) GPT-OSS-120B.

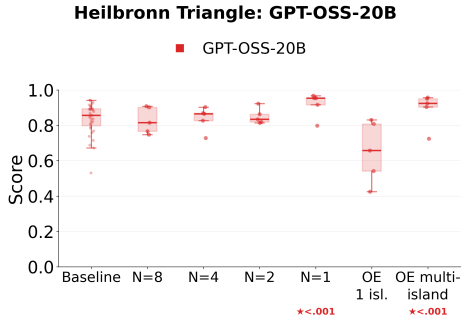
Figure 10: **Progression toward OpenEvolve-style search on circle packing.** The four panels show the model-specific breadth–depth progression and available OpenEvolve variants.



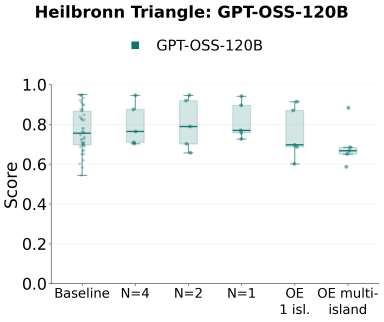
(e) Qwen2.5-3B-Instruct.



(f) Qwen3-4B-Instruct.

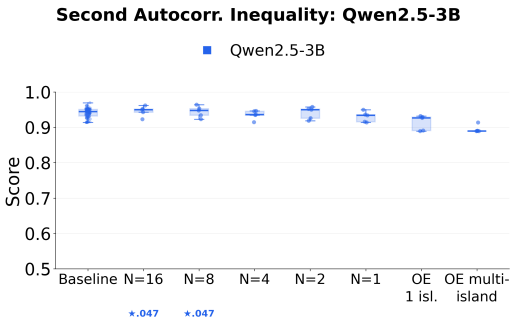


(g) GPT-OSS-20B.

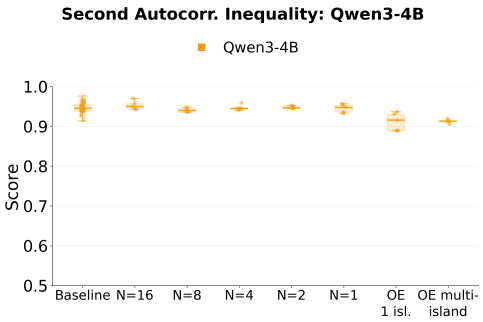


(h) GPT-OSS-120B.

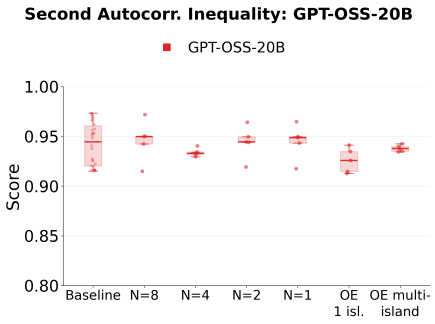
Figure 10: **Progression toward OpenEvolve-style search on the Heilbronn triangle task, continued.**



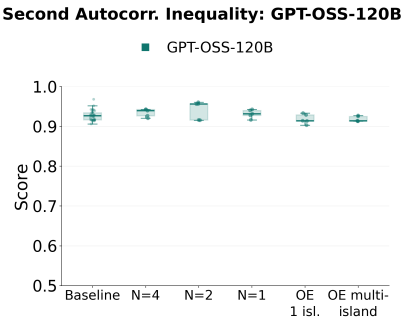
(i) Qwen2.5-3B-Instruct.



(j) Qwen3-4B-Instruct.



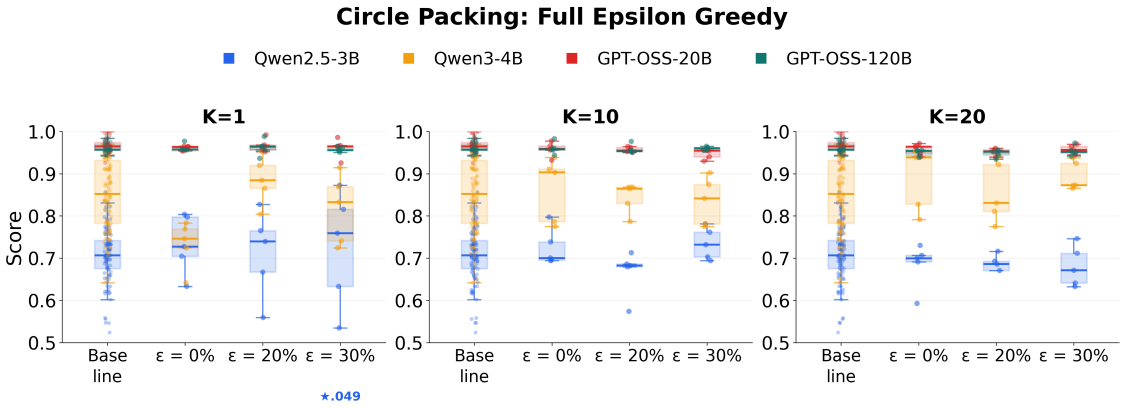
(k) GPT-OSS-20B.



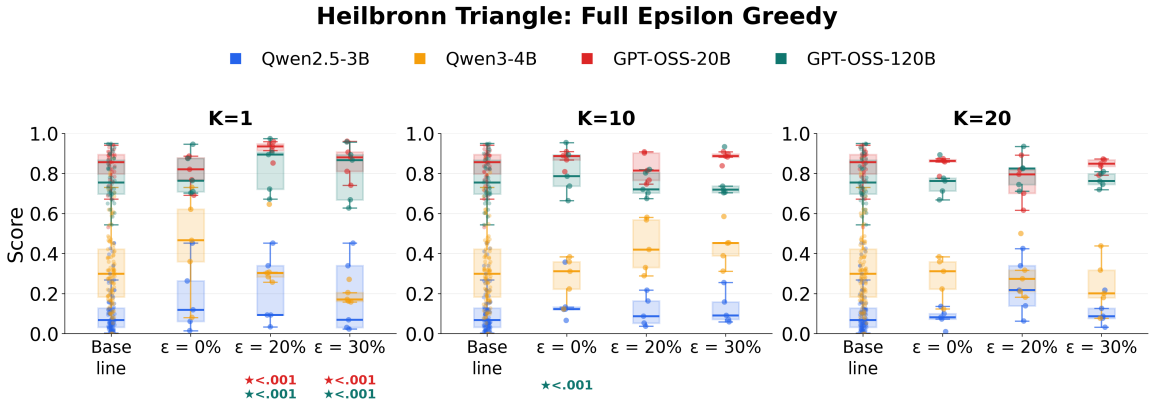
(l) GPT-OSS-120B.

Figure 10: **Progression toward OpenEvolve-style search on the second autocorrelation inequality task, continued.** Intermediate breadth–depth allocations can be competitive, while the OpenEvolve endpoint does not uniformly dominate across models and tasks.

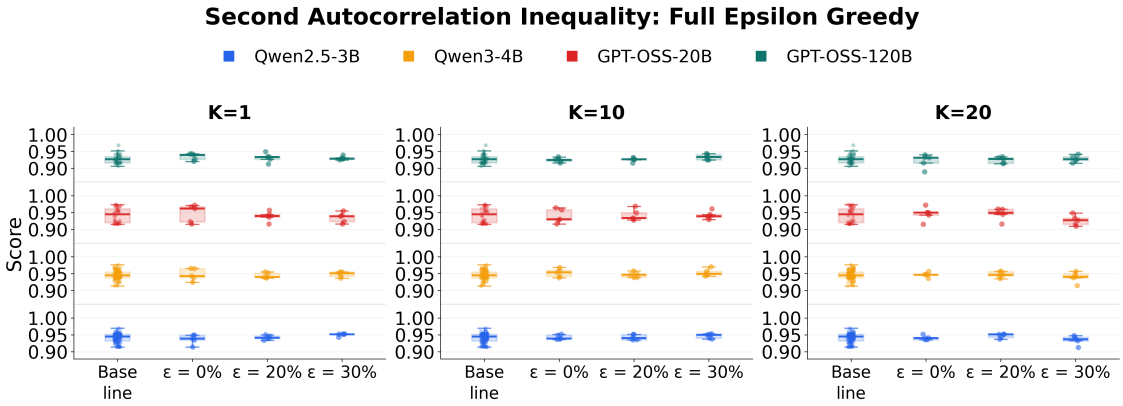
D.2 Full Epsilon-Greedy Results



(a) Circle packing.



(b) Heilbronn triangle.

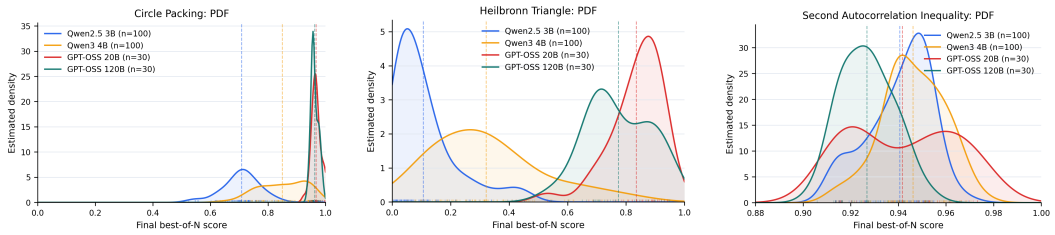


(c) Second autocorrelation inequality.

Figure 11: **Full epsilon-greedy grid across all three tasks.** Each panel varies elite archive size and ϵ probability, with model-specific score distributions overlaid.

D.3 Sequential BoN Baseline

We first evaluate a Sequential Best-of-N (BoN) baseline for each model–problem pair. Sequential BoN is the $K = 1$, $\epsilon = 0$ special case of the archive formulation in Equation 9: it always expands the best program found so far and never samples from the non-elite history. We use the baseline



(a) Circle packing. (b) Heilbronn triangle. (c) Second autocorrelation inequality.

Figure 12: **Sequential BoN baseline distributions.** Kernel-density estimates show final scores for each task and model. Dashed vertical lines mark model-specific means, and rug marks show individual runs. These baseline distributions, with 100 runs for Qwen models and 30 runs for GPT-OSS models, define the empirical reference used in the bootstrap comparisons.

score distribution (30 runs for GPT-OSS models, and 100 runs for Qwen models) as the reference for the archive, exploration, UCT/PUCT, and OpenEvolve-style comparisons. Figure 12 shows kernel-density estimates of the final baseline scores.

The baseline distributions already reveal a large model effect before any search-harness modification is introduced. On circle packing, the GPT-OSS models are tightly concentrated near the top of the score range, indicating both strong performance and low variance. Qwen3-4B reaches high-scoring regions but has a broader distribution, whereas Qwen2.5-3B is centered substantially lower. Consequently, the GPT-OSS models have less headroom for visible improvement, while the Qwen models leave more room for a search method to stabilize or redirect weak runs.

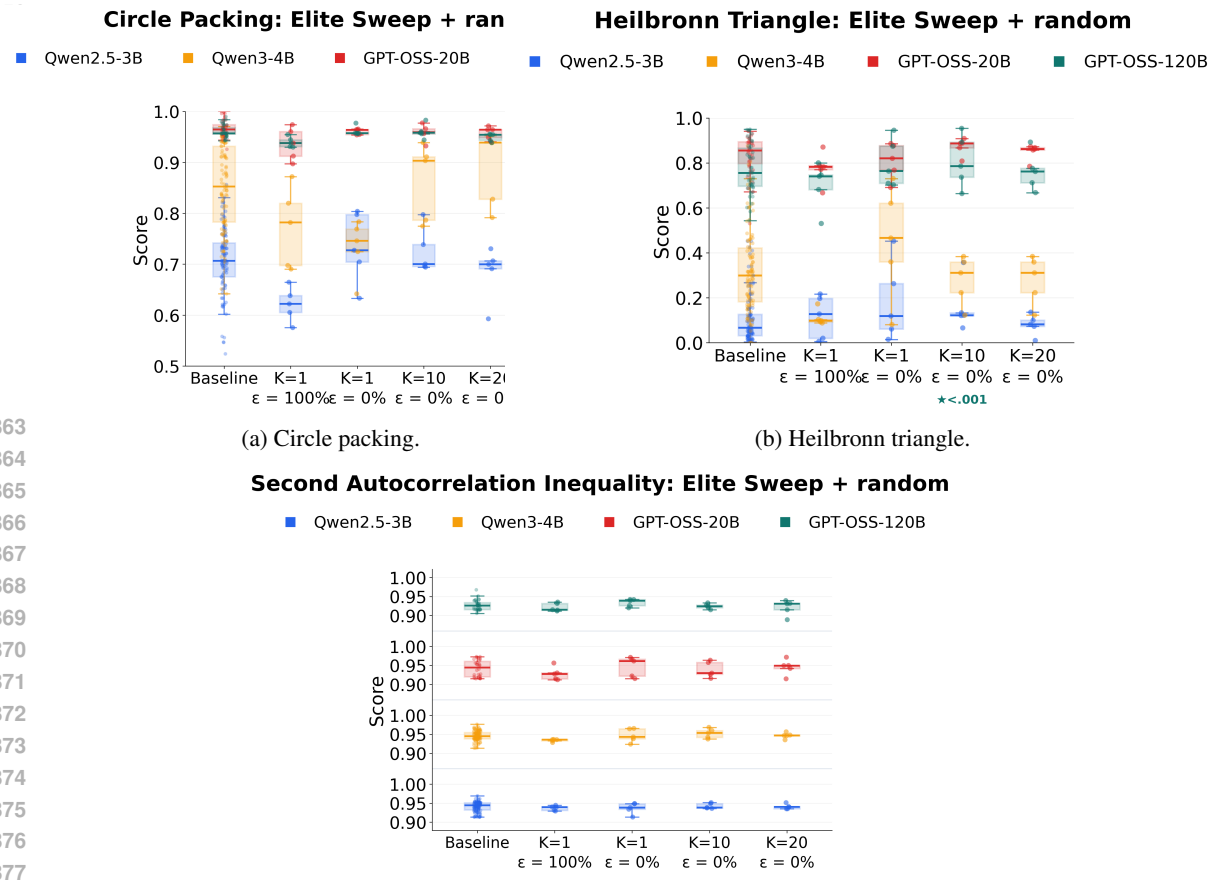
For Heilbronn triangle, the separation between models is even more pronounced. The GPT-OSS models again form a high-score mode, while Qwen2.5-3B is concentrated near very low scores and Qwen3-4B occupies a broad middle range. The wider Qwen3-4B distribution suggests that it occasionally finds useful solutions but does so inconsistently. On the second autocorrelation inequality, the four models are much closer in score, although GPT-OSS-20B has a broader distribution that reflects its distinct low- and high-score optimizer regimes. GPT-OSS-120B follows the same broad GPT-OSS pattern but with a smaller rollout budget, so we interpret its distributions as within-model harness comparisons rather than direct evidence of scale alone. Overall, these baseline KDEs provide an important reference point: later search algorithms should be interpreted relative to both the model's average baseline performance and the spread of its stochastic runs, since improvements for a high-performing, low-variance model mean something different from improvements for a lower-performing, high-variance model.

D.4 Adding an Elite Archive

Experimental setup. We next sample parent programs from the Top-K archive. We evaluate $K \in \{1, 10, 20, 30\}$ with zero general-history exploration, and include the $K = 1$ setting with full general-history sampling as a random-selection comparison. Sequential BoN is shown in the first column of each panel.

Figures 13a, 13b, and 13c compare Sequential BoN with increasingly large elite archives. The central pattern is model–problem dependence rather than monotonic improvement with K. The GPT-OSS models start from a high baseline and remain near the top of the score range across archive settings, leaving limited headroom. Qwen3-4B moves more visibly, particularly on circle packing, where some archive settings shift the distribution upward or reduce its variance. Qwen2.5-3B remains more variable and shows weaker evidence of consistent archive-driven gains.

On Heilbronn triangle, the elite archive does not substantially change the overall ordering between models. The GPT-OSS models remain dominant, while the Qwen models continue to exhibit lower and more variable score distributions. The second autocorrelation inequality is similarly stable across archive sizes. The distributions shift only modestly, with no consistent improvement as K increases. This suggests that the elite archive is most useful when the model can already generate multiple plausible high-quality lineages. When candidate quality is poor or highly unstable, increasing K alone does not reliably compensate.



(a) Circle packing.

(b) Heilbronn triangle.

(c) Second Autocorrelation Inequality.

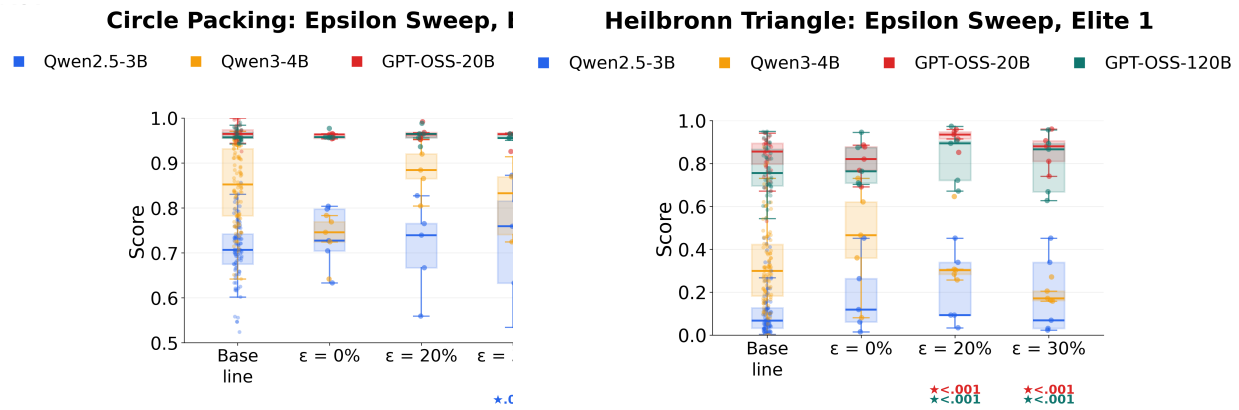
Figure 13: **Elite-archive sweep across all three tasks.** Parent programs are sampled from a Top-K archive under $\epsilon = 0\%$. Colored stars denote configurations whose five-run maximum significantly exceeds the bootstrapped Sequential BoN reference. The absence of a star indicates insufficient evidence under this test, not evidence of no effect.

D.5 Adding Exploration

Experimental setup. We isolate exploration by fixing the elite archive size to $K = 1$ and varying the ϵ probability over $\{0\%, 20\%, 30\%\}$. In the notation of Equation 9, these settings correspond to exploration probabilities $\epsilon \in \{0, 0.2, 0.3\}$. The sweep tests whether occasional access to non-elite historical programs improves over always expanding the best program found so far.

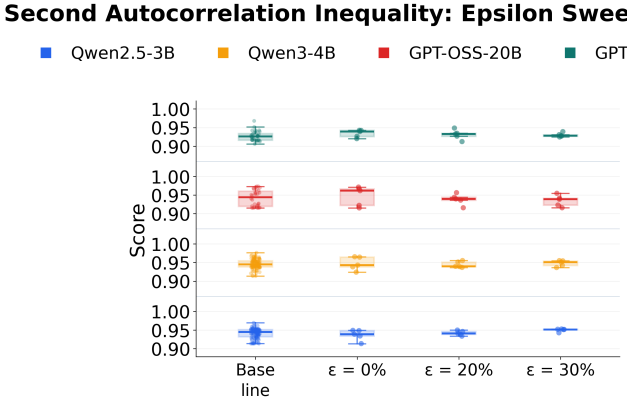
Figures 14a, 14b, and 14c evaluate whether sampling from the general archive of all solutions improves over always exploiting the elite archive, and suggest that adding random access to the general archive is not automatically beneficial. The results do not show a single exploration rate that dominates across models or tasks. On circle packing, GPT-OSS-20B remains strong across the available exploration settings, while Qwen3-4B is competitive but more variable. Qwen2.5-3B exhibits the largest spread, suggesting that exploration can introduce instability when the model's proposal distribution is weaker, as we spend more compute budget experimenting with less performant solutions.

The Heilbronn triangle results are even more model-dependent. GPT-OSS maintains high scores, but the Qwen models show mixed behavior and no clear monotonic trend as exploration increases. On the second autocorrelation inequality, the score distributions remain tightly clustered across ϵ probabilities. Qwen2.5-3B improves slightly at $\epsilon = 30\%$, but this small shift does not transfer consistently to the other models. These results are consistent with viewing general-archive exploration as an anti-premature-convergence mechanism rather than a uniformly better parent-selection rule. Exploration may help recover diverse lineages, but it also spends budget on programs that may be less relevant than the current elite candidates.



(a) Circle packing.

(b) Heilbronn triangle.



(c) Second Autocorrelation Inequality.

Figure 14: **General-history exploration sweep across all three tasks.** The elite archive is fixed at $K = 1$, while the epsilon probability is increased from 0% to 30%. Higher epsilon probability corresponds to more frequent sampling from the full program history.

D.6 Full Epsilon-Greedy Grid

Experimental setup. The full ϵ-greedy sweep combines the previous interventions by varying both elite archive size and ϵ. We evaluate $K \in \{1, 10, 20\}$ and ϵ probabilities in $\{0\%, 20\%, 30\%\}$ where runs are available. The GPT-OSS models have incomplete coverage: the 10% ϵ and $K = 30$ settings are missing because of compute constraints. Empty positions therefore denote unavailable runs rather than zero performance.

Figures 11a, 11b, and 11c show that these two hyperparameters interact rather than contributing independently. There is no globally dominant (K, ϵ) setting. On circle packing, Qwen3-4B often benefits from archive and exploration settings, while GPT-OSS remains near ceiling for the configurations that were run. Qwen2.5-3B remains lower and more sensitive to hyperparameter choice. On Heilbronn triangle, the same sweep is less forgiving: GPT-OSS models stay strong, but the Qwen models show substantial variance and many configurations remain far below the GPT-OSS score range. On the second autocorrelation inequality, most configurations remain close to their model-specific baselines, and changing K or the ϵ probability produces small, non-monotonic shifts rather than a clear winner. Thus, epsilon-greedy search should be treated as a family of policies whose effectiveness depends on the model's ability to produce useful candidate programs, rather than as a single robust improvement over Sequential BoN.

D.7 Progression toward OpenEvolve

Experimental setup. Finally, we compare archive and epsilon-greedy variants with a progression of increasingly OpenEvolve-like configurations. This sweep reallocates a fixed rollout budget from

breadth to depth by decreasing the number of children per expansion, N, and increasing the number of update steps, T. The final settings add OpenEvolve-style island mechanisms. Because the available progression is model specific, we interpret these plots as within-model comparisons rather than a perfectly matched grid across models.

For Qwen2.5-3B and Qwen3-4B, the (N, T) ladder is $(N, T) \in \{(16, 100), (8, 200), (4, 400), (2, 800), (1, 1600)\}$, followed by OpenEvolve 1-island and multi-island variants. For GPT-OSS-20B, the available ladder is $(N, T) \in \{(4, 80), (2, 160), (1, 320)\}$, followed by OpenEvolve 1-island and multi-island variants. GPT-OSS-120B is followed by the same OpenEvolve variants but progresses as follows: $(N, T) \in \{(4, 40), (2, 80), (1, 160)\}$. Figure 10 presents the complete model-specific progression.

The main trend is that deeper OpenEvolve-style refinement is competitive but does not uniformly dominate simpler search variants. On circle packing, GPT-OSS-20B remains strong across the progression, while Qwen3-4B achieves competitive scores in several intermediate settings. On Heilbronn triangle, GPT-OSS-20B again remains the most reliable, but the OpenEvolve-style settings do not consistently improve the Qwen models. The second autocorrelation inequality provides the clearest negative example. Intermediate breadth–depth settings remain near the baseline, while the one-island and four-island endpoints reduce performance for both Qwen models. This suggests that longer refinement chains are valuable when local improvements compound, but they can also overcommit the search to a narrow trajectory. In this sense, OpenEvolve-style search is best understood as one point on a breadth–depth tradeoff rather than as a uniformly superior endpoint.

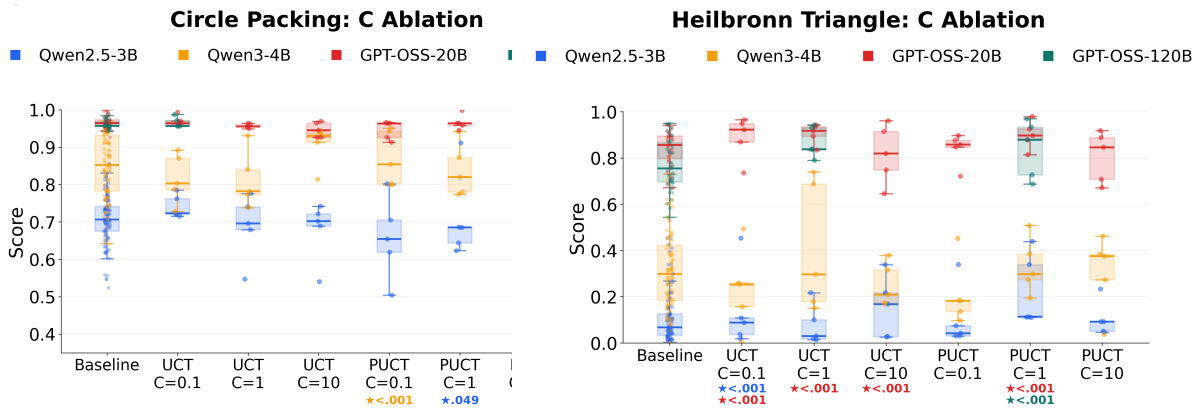
D.8 From Sequential BoN to UCT and PUCT

The previous ablations treat parent selection as a relatively simple archive-sampling problem: we either sample from the elite archive, sample from the broader program database, or interpolate between the two. We next evaluate whether a more structured tree-search-inspired selection rule improves over these simpler Sequential BoN-style baselines. In particular, we compare PUCT and UCT variants that allocate the generation budget across multiple parent-selection branches.

For each task and model, we use the same Sequential BoN score distribution as the baseline. We evaluate UCT and PUCT under matched rollout budgets: 1600 generations for the Qwen models, 320 for GPT-OSS-20B, and 160 for GPT-OSS-120B. The sweeps vary three components: the exploration constant C, the number of children generated per selected parent, and the number of parent programs selected in parallel, P. The exact child batch size differs by model in the P sweep because the GPT-OSS models use smaller total budgets. Those runs are therefore aligned by P rather than by an identical batch size.

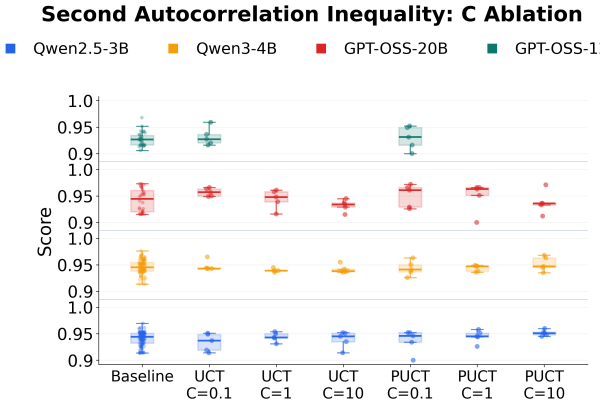
Colored boxes denote model-specific score distributions, and colored stars mark configurations whose five-run maximum significantly exceeds the Sequential BoN reference under the bootstrap test. Because each non-baseline configuration contains only five runs, we emphasize distributional trends and consistency across model–problem pairs rather than treating an individual setting as a definitive optimum.

D.8.1 Exploration-Constant Ablations



(a) Circle packing.

(b) Heilbronn triangle.



(c) Second Autocorrelation Inequality.

Figure 15: **UCT/PUCT exploration-constant ablations across all three tasks.** PUCT and UCT are shown side by side for each value of C, with model-specific score distributions overlaid. The preferred exploration strength varies across model–problem pairs.

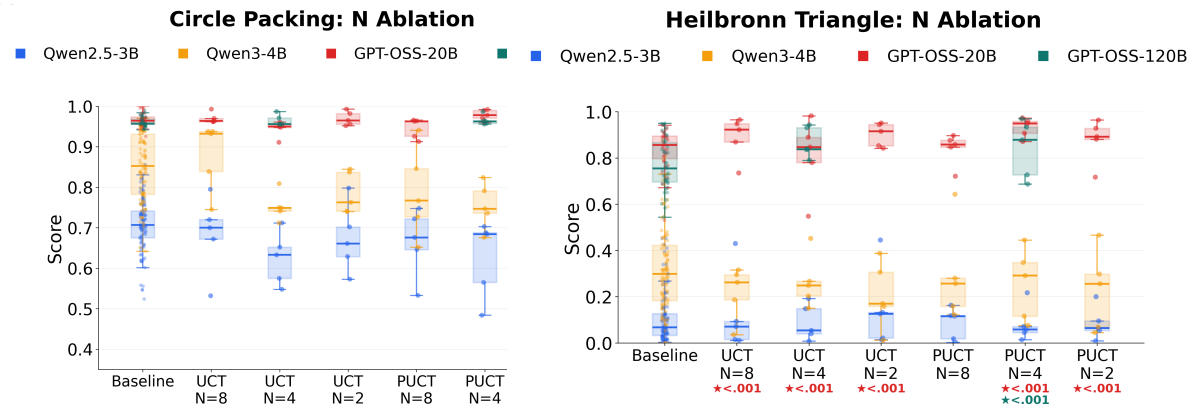
D.8.2 Breadth–Depth Ablations

Figure 16 varies the child batch size $N \in \{8, 4, 2\}$ and adjusts the number of update steps T so that the total rollout budget remains fixed. This tests whether the budget is better spent on broader branching or longer refinement chains. As in the C sweep, no allocation is universally best. GPT-OSS models remain strong across the tested values on circle packing, while the Qwen models are more sensitive. In several settings, decreasing the child batch size does not improve the weaker models, suggesting that deeper refinement helps primarily when the selected lineage is already promising.

For Heilbronn triangle, changing the breadth–depth allocation has a smaller and less consistent effect. GPT-OSS again remains strong across the sweep, but the Qwen models do not improve reliably as N changes. The second autocorrelation inequality also lacks a consistent breadth–depth preference across models, with most settings remaining close to their corresponding baselines. Search-budget allocation is therefore conditional on proposal quality: deeper search can amplify a promising lineage, but it can also overcommit to a weak trajectory.

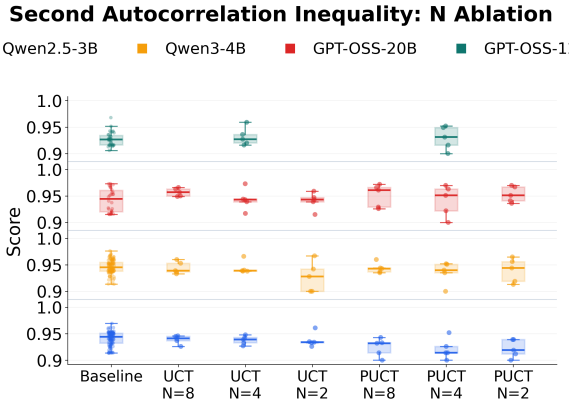
D.8.3 PUCT: P ablations

Figure 17 varies the number of parent programs selected per step, $P \in \{2, 4\}$. We keep the child batch size fixed at four for the Qwen models and use the corresponding budget-matched GPT-OSS runs, adjusting the number of update steps to preserve the total rollout budget. Because the child batch sizes differ across model families, configurations are grouped by P. The figure should there-



(a) Circle Packing.

(b) Heilbronn Triangle.



(c) Second Autocorrelation Inequality.

Figure 16: **UCT/PUCT breadth–depth ablations.** The number of children generated per selected parent varies while the total rollout budget remains fixed. No single allocation is best across all model–problem pairs.

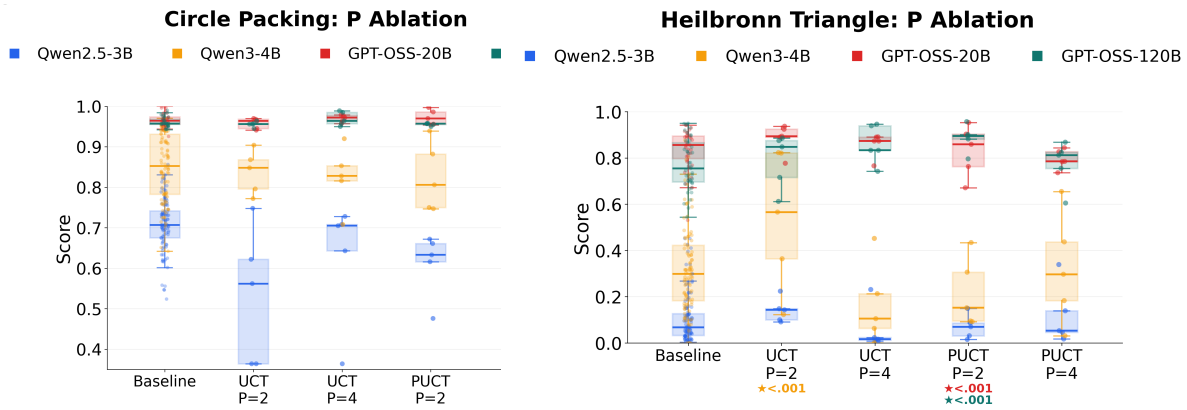
fore be read as a comparison of parallel-parent search structure rather than a strict comparison of identical batch sizes.

Across the three tasks, increasing P does not uniformly improve performance. On Circle Packing, GPT-OSS-20B remains near ceiling for both $P = 2$ and $P = 4$, while Qwen3-4B remains competitive but variable. Qwen2.5-3B again shows weaker and less stable results. On Heilbronn Triangle, the separation between models is especially large: GPT-OSS-20B remains high, whereas Qwen3-4B and Qwen2.5-3B show much lower distributions. On the second autocorrelation inequality, changing the number of root programs again produces only modest and model-dependent shifts. These results suggest that increasing the number of root programs is not by itself sufficient to improve search quality. The benefit depends on whether each root can seed a useful optimization trajectory.

Overall, the PUCT and UCT ablations reinforce the main pattern from the simpler archive-based experiments. Search structure matters, but its effect is conditional on the model's proposal quality. GPT-OSS-20B benefits from or at least remains robust under more structured search, while weaker models show greater sensitivity and less reliable improvement. In this setting, PUCT/UCT should be viewed not as a replacement for model capability, but as a mechanism for allocating search effort when the model can already generate promising candidates.

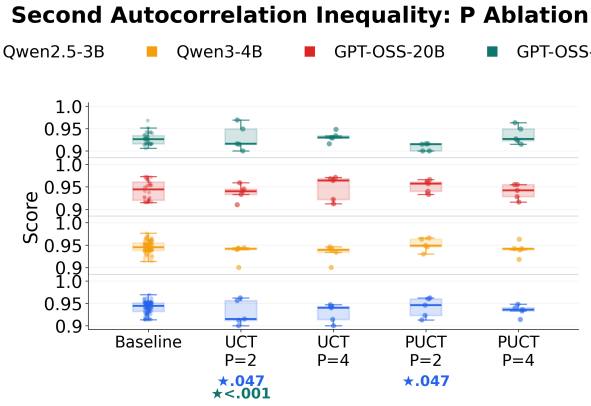
E Additional Config Vector Battle Leaderboards

Construction of the progression plot. The progression plot in Figure 1b is derived from the cross-pair majority-win probability leaderboard for the max statistic. For each configuration, we take the



(a) Circle Packing.

(b) Heilbronn Triangle.



(c) Second Autocorrelation Inequality.

Figure 17: **UCT/PUCT parallel-parent ablations.** Runs are grouped by the number of selected parent programs, P. Because total budgets differ by model, configurations are aligned by P rather than by an identical child batch size.

estimated probability that its best-of-five run beats the bootstrapped Sequential BoN baseline on a majority of the 12 model–problem pairs. Because the baseline has probability 0.5 of beating itself, we first center each configuration by subtracting 0.5. We then rescale the centered value by a factor of two:

$$s(c) = 2\,(\hat{p}_{\mathrm{maj}}(c) - 0.5)\,.$$

This maps the baseline to 0, configurations that always beat the baseline to $+1$, and configurations that never beat the baseline to -1. The plotted points are ordered to match the experimental progressions in Figure 1a: the OpenEvolve panel moves from the Sequential BoN baseline through elite-archive and depth-budget ablations to OpenEvolve variants, while the TTT-Discover panel moves from the same baseline through tree-policy choices, depth-budget reductions, and the final $P = 4$ TTT-Discover setting. Positive values therefore indicate improvement over the baseline majority-win rate, while negative values indicate degradation. The main trend is that simpler search variants, especially UCT/PUCT and small archive settings, remain above baseline, whereas adding the full OpenEvolve machinery drives performance substantially below baseline.

Figures 18, 19, and 20 show the complete cross-pair leaderboard used to construct the compressed progression plot. Each bar reports how often a configuration beats the Sequential BoN baseline across a majority of model–problem pairs under the corresponding summary statistic. The ranking is consistent with the main-text message: lightweight tree-search and archive-based variants occupy most of the top positions, while the full OpenEvolve-style configurations appear near the bottom.

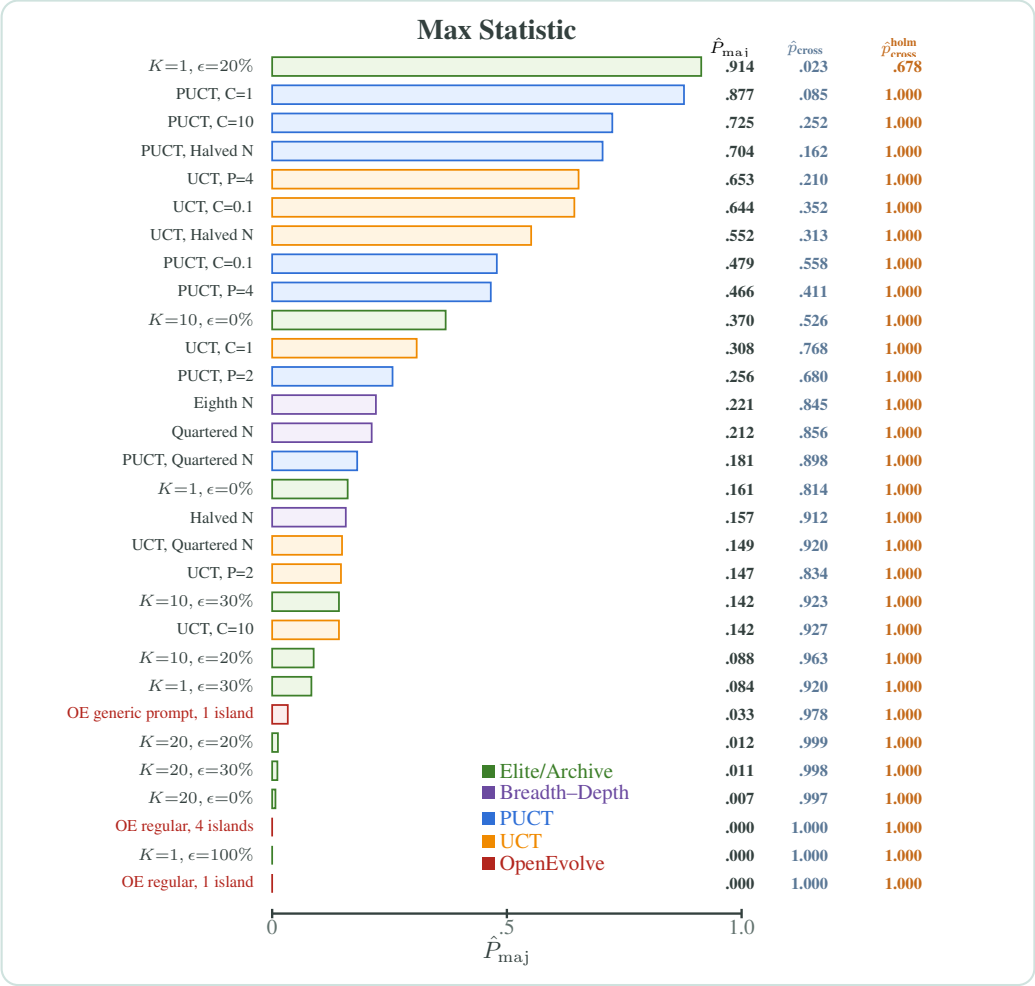


Figure 18: **Full cross-pair majority-win probability leaderboard (max statistic).** All 30 configurations ranked by battle win rate using the max score across all dimensions of the config vector.

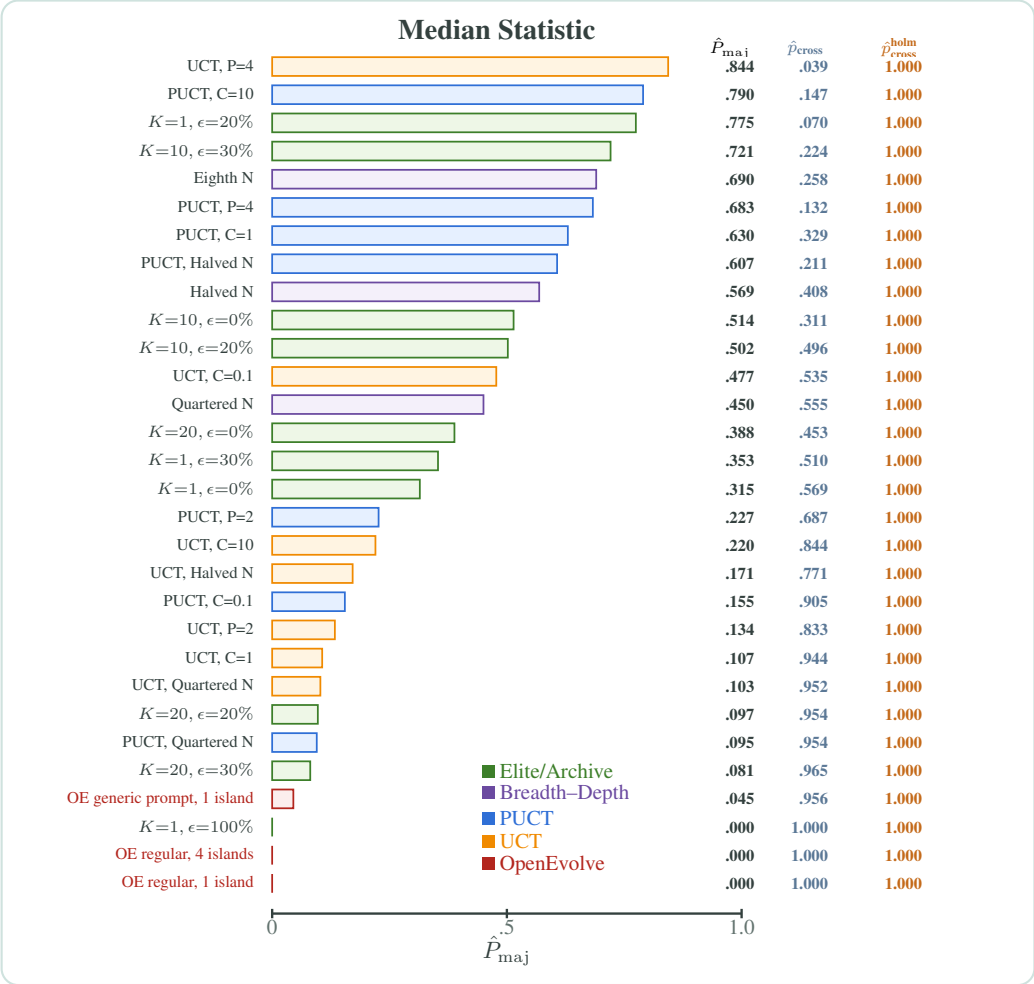


Figure 19: **Full cross-pair majority-win probability leaderboard (median statistic).** All 30 configurations ranked by battle win rate using the median score across dimensions.

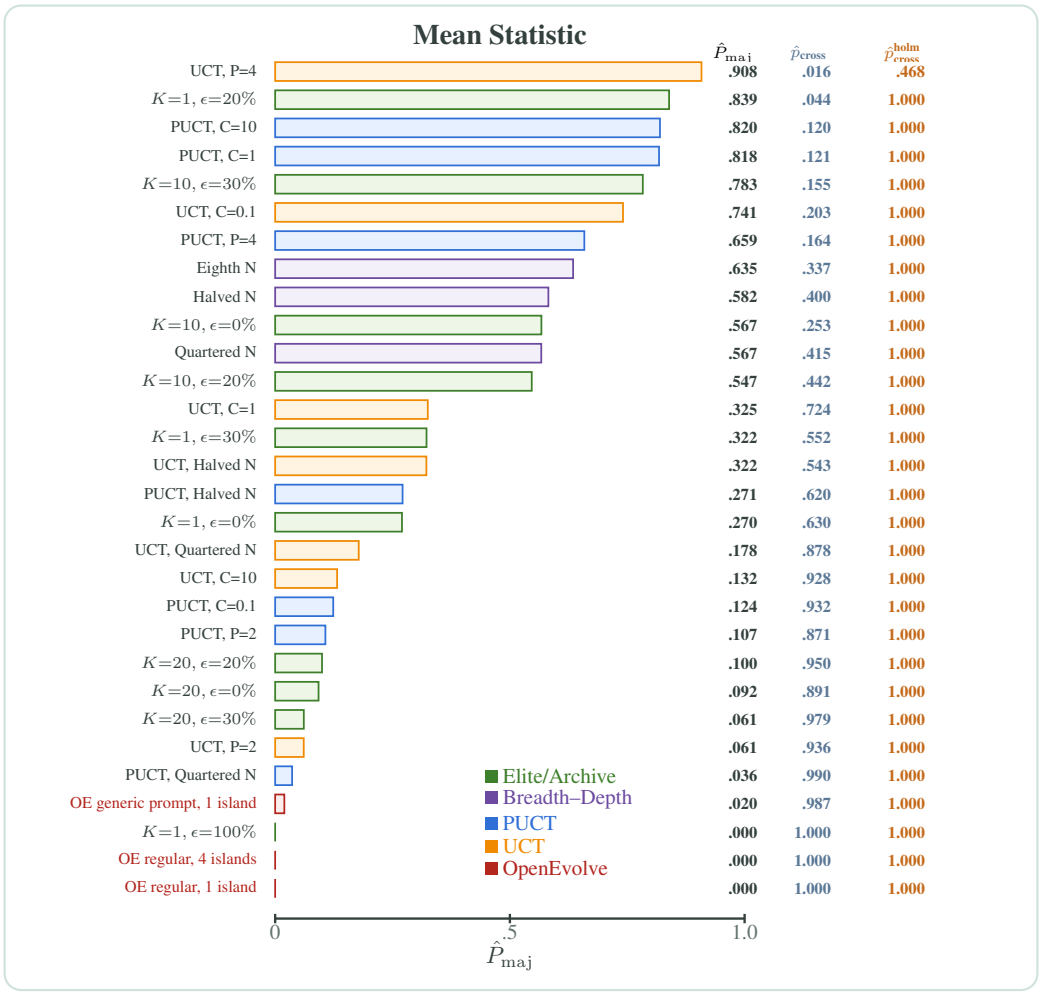


Figure 20: **Full cross-pair majority-win probability leaderboard (mean statistic).** All 30 configurations ranked by battle win rate using the mean score across dimensions.

F Complete Per-Configuration Results

Tables 3–12 organize the results by search-harness category. Table 3 summarizes the full Sequential BoN baseline pool for each model–task pair, reporting the number of runs, mean, median, and maximum. The remaining tables report each five-run configuration's mean, median, and maximum beside their matched-baseline one-sided bootstrap p-values.

Every null distribution uses 100,000 bootstrap samples of five runs. Bold denotes $p < 0.05$, while $< 10^{-5}$ denotes that the configuration outperformed all bootstrap samples for the respective statistic. Unless otherwise noted, Qwen configurations use $N = 16$ and $T = 100$, GPT-OSS-20B uses $N = 8$ and $T = 40$, and GPT-OSS-120B uses $N = 4$ and $T = 40$.

Pair codes are CP (circle packing), HT (Heilbronn triangle), and SAI (second autocorrelation inequality). Model codes are Q3B (Qwen2.5-3B-Instruct), Q4B (Qwen3-4B-Instruct-2507), GPT20B (GPT-OSS-20B), and GPT120B (GPT-OSS-120B).

Table 3: **Sequential BoN baselines.** Summary statistics for the complete matched baseline pools used by the bootstrap tests.

Pair	Runs	Mean	Median	Max
CP/Q3B	100	0.7087	0.7066	0.9488
CP/Q4B	100	0.8502	0.8524	0.9705
CP/GPT20B	30	0.9679	0.9658	0.9984
CP/GPT-120B	30	0.9625	0.9573	0.9898
HT/Q3B	100	0.1066	0.0672	0.4522
HT/Q4B	100	0.3213	0.2990	0.8765
HT/GPT20B	30	0.8300	0.8490	0.9417
HT/GPT-120B	30	0.7738	0.7507	0.9496
SAI/Q3B	100	0.9406	0.9444	0.9695
SAI/Q4B	100	0.9462	0.9452	0.9764
SAI/GPT20B	30	0.9417	0.9444	0.9732
SAI/GPT-120B	30	0.9268	0.9267	0.9515

Table 4: **Sequential BoN + top-K.** Adds a top-K elite archive while sampling exclusively from that archive.

		Observed statistic			One-sided p-value		
Pair	Configuration	Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	K=10	0.7252	0.7004	0.7974	0.28825	0.68088	0.34064
CP/Q3B	K=20	0.6840	0.6997	0.7303	0.79711	0.69830	0.90788
CP/Q3B	K=30	0.6975	0.6896	0.7707	0.65234	0.80736	0.55686
CP/Q4B	K=10	0.8628	0.9032	0.9387	0.37690	0.25001	0.70924
CP/Q4B	K=20	0.8898	0.9387	0.9525	0.14044	0.07406	0.33831
CP/Q4B	K=30	0.9101	0.9124	0.9428	**0.04408**	0.16242	0.52763
CP/GPT20B	K=10	0.9581	0.9589	0.9774	0.92150	0.87938	0.73659
CP/GPT20B	K=20	0.9579	0.9640	0.9716	0.92569	0.73934	0.86832
CP/GPT-120B	K=10	0.9603	0.9581	0.9828	0.63900	0.25300	0.28200
CP/GPT-120B	K=20	0.9492	0.9541	0.9556	0.99700	0.96500	1.00000
HT/Q3B	K=10	0.1598	0.1220	0.3570	0.13651	0.14980	0.22633
HT/Q3B	K=20	0.0802	0.0820	0.1360	0.66310	0.40512	0.74757
HT/Q3B	K=30	0.1622	0.0990	0.4000	0.12800	0.31593	0.18496
HT/Q4B	K=10	0.3268	0.3412	0.6114	0.45090	0.28405	0.375
HT/Q4B	K=20	0.2800	0.3110	0.3840	0.66657	0.44289	0.83360
HT/Q4B	K=30	0.2734	0.2840	0.4080	0.69598	0.55453	0.80732
HT/GPT20B	K=10	0.8722	0.8871	0.9090	0.14883	0.26081	0.51170
HT/GPT20B	K=20	0.8493	0.8618	0.8734	0.35021	0.43606	0.94127
HT/GPT-120B	K=10	0.8067	0.7858	0.9540	0.24600	0.30700	$< 10^{-5}$
HT/GPT-120B	K=20	0.7623	0.7621	0.8929	0.59900	0.43700	0.66800
SAI/Q3B	K=10	0.9425	0.9385	0.9515	0.39766	0.74919	0.65380
SAI/Q3B	K=20	0.9405	0.9402	0.9514	0.52628	0.64846	0.65380
SAI/Q3B	K=30	0.9476	0.9510	0.9521	0.11450	0.10397	0.58486
SAI/Q4B	K=10	0.9523	0.9536	0.9684	0.14249	0.13692	0.14021
SAI/Q4B	K=20	0.9467	0.9468	0.9566	0.47437	0.44256	0.74515
SAI/Q4B	K=30	0.9407	0.9385	0.9548	0.82718	0.86311	0.76110

Continued on next page

Table 4 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
SAI/GPT20B	K=10	0.9388	0.9298	0.9638	0.61273	0.62422	0.73484
SAI/GPT20B	K=20	0.9457	0.9496	0.9717	0.34475	0.43836	0.29137
SAI/GPT20B	K=30	0.9292	0.9374	0.9588	0.89866	0.56195	0.78812
SAI/GPT-120B	K=10	0.9243	0.9246	0.9335	0.68400	0.67800	0.78500
SAI/GPT-120B	K=20	0.9214	0.9310	0.9395	0.85600	0.23600	0.60400

Table 5: **Sequential BoN + full history.** Adds full-history exploration while retaining a single elite program.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	epsilon=10%	0.7130	0.6842	0.7685	0.44745	0.83474	0.58263
CP/Q3B	epsilon=100%	0.6210	0.6220	0.6646	0.99709	0.98858	0.99961
CP/Q3B	epsilon=20%	0.7116	0.7394	0.8273	0.46639	0.12700	0.14229
CP/Q3B	epsilon=30%	0.7230	0.7592	0.8727	0.31476	0.05864	**0.04919**
CP/Q3B	epsilon=30%, N=2	0.7099	0.7261	0.7398	0.48928	0.31851	0.77817
CP/Q3B	epsilon=30%, N=4	0.7473	0.7304	0.9108	0.09636	0.28469	**0.04919**
CP/Q3B	epsilon=30%, N=8	0.6865	0.6893	0.7207	0.77441	0.80736	0.93858
CP/Q4B	epsilon=10%	0.8555	0.8708	0.9376	0.45281	0.42621	0.74454
CP/Q4B	epsilon=100%	0.7723	0.7818	0.8719	0.98124	0.90733	0.94882
CP/Q4B	epsilon=20%	0.8853	0.8843	0.9526	0.17137	0.30004	0.13977
CP/Q4B	epsilon=30%	0.8163	0.8327	0.9144	0.82201	0.71718	0.81911
CP/GPT20B	epsilon=100%	0.9363	0.9377	0.9740	0.99999	0.99969	0.78871
CP/GPT20B	epsilon=20%	0.9666	0.9638	0.9919	0.57688	0.79216	0.41107
CP/GPT20B	epsilon=30%	0.9615	0.9644	0.9861	0.82669	0.68277	0.51342
CP/GPT20B	epsilon=30%, N=2	0.9700	0.9710	0.9803	0.38067	0.20907	0.67449
CP/GPT20B	epsilon=30%, N=4	0.9747	0.9722	0.9944	0.16410	0.16236	0.41107
CP/GPT-120B	epsilon=100%	0.9395	0.9377	0.9545	1.00000	1.00000	1.00000
CP/GPT-120B	epsilon=20%	0.9621	0.9645	0.9883	0.51700	0.20200	0.15200
CP/GPT-120B	epsilon=30%	0.9577	0.9555	0.9650	0.78700	0.91100	0.85800
HT/Q3B	epsilon=10%	0.0684	0.0480	0.1460	0.76706	0.74734	0.74757
HT/Q3B	epsilon=100%	0.1136	0.1280	0.2170	0.39538	0.10327	0.55556
HT/Q3B	epsilon=20%	0.2022	0.0930	0.4520	**0.03948**	0.38619	0.09626
HT/Q3B	epsilon=30%	0.1828	0.0690	0.4520	0.07190	0.47918	0.09626
HT/Q4B	epsilon=10%	0.2806	0.2280	0.6390	0.66406	0.74674	0.30433
HT/Q4B	epsilon=100%	0.1110	0.0980	0.1730	0.99868	0.99561	0.99923
HT/Q4B	epsilon=20%	0.3594	0.3030	0.6460	0.31180	0.48019	0.30433
HT/Q4B	epsilon=30%	0.1934	0.1710	0.2710	0.94759	0.90518	0.98706
HT/GPT20B	epsilon=100%	0.7760	0.7829	0.8709	0.89569	0.91397	0.94127
HT/GPT20B	epsilon=20%	0.9215	0.9347	0.9590	**0.00088**	**0.00258**	$< 10^{-5}$
HT/GPT20B	epsilon=30%	0.8597	0.8804	0.9608	0.25169	0.31523	$< 10^{-5}$
HT/GPT-120B	epsilon=100%	0.7001	0.7409	0.8006	0.93700	0.56100	0.92100
HT/GPT-120B	epsilon=20%	0.8333	0.8940	0.9738	0.09800	**0.04800**	$< 10^{-5}$
HT/GPT-120B	epsilon=30%	0.8025	0.8667	0.9570	0.27100	0.10500	$< 10^{-5}$
SAI/Q3B	epsilon=10%	0.9385	0.9361	0.9513	0.65448	0.83689	0.71473
SAI/Q3B	epsilon=100%	0.9369	0.9394	0.9445	0.74589	0.71651	0.96594
SAI/Q3B	epsilon=20%	0.9417	0.9408	0.9498	0.44758	0.59412	0.84587
SAI/Q3B	epsilon=30%	0.9500	0.9515	0.9527	**0.04536**	0.05082	0.44333
SAI/Q4B	epsilon=10%	0.9460	0.9484	0.9696	0.52224	0.40567	0.09589
SAI/Q4B	epsilon=100%	0.9340	0.9360	0.9365	0.97919	0.96258	0.99978
SAI/Q4B	epsilon=20%	0.9444	0.9397	0.9553	0.62436	0.83745	0.76110
SAI/Q4B	epsilon=30%	0.9479	0.9510	0.9551	0.39146	0.33360	0.76110
SAI/GPT20B	epsilon=10%	0.9341	0.9377	0.9528	0.78303	0.56195	0.92358
SAI/GPT20B	epsilon=100%	0.9283	0.9277	0.9564	0.91452	0.62422	0.83268
SAI/GPT20B	epsilon=20%	0.9384	0.9396	0.9564	0.62869	0.56195	0.83268
SAI/GPT20B	epsilon=30%	0.9347	0.9386	0.9546	0.76217	0.56195	0.86877
SAI/GPT-120B	epsilon=100%	0.9218	0.9154	0.9355	0.83700	0.97800	0.78500
SAI/GPT-120B	epsilon=20%	0.9310	0.9328	0.9486	0.20800	0.13400	0.17000
SAI/GPT-120B	epsilon=30%	0.9299	0.9281	0.9390	0.28000	0.35200	0.67500

Table 6: **Epsilon-greedy (top-K + full history).** Combines a top-K elite archive with epsilon-greedy sampling from the full history.

		Observed statistic			One-sided p-value		
Pair	Configuration	Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	K=10, epsilon=10%	0.7538	0.7339	0.8664	0.06534	0.17797	**0.04919**
CP/Q3B	K=10, epsilon=10%, N=2	0.6922	0.6830	0.7308	0.71403	0.84749	0.90013
CP/Q3B	K=10, epsilon=10%, N=4	0.6249	0.6448	0.6892	0.99576	0.97289	0.99655
CP/Q3B	K=10, epsilon=10%, N=8	0.7269	0.7223	0.8057	0.26981	0.31851	0.26652
CP/Q3B	K=10, epsilon=20%	0.6669	0.6822	0.7129	0.91558	0.85919	0.95786
CP/Q3B	K=10, epsilon=30%	0.7343	0.7318	0.7812	0.19188	0.23640	0.44275
CP/Q3B	K=20, epsilon=10%	0.6823	0.6842	0.7016	0.81295	0.83474	0.98853
CP/Q3B	K=20, epsilon=20%	0.6517	0.6858	0.7163	0.96690	0.83474	0.95382
CP/Q3B	K=20, epsilon=30%	0.6805	0.6712	0.7463	0.82760	0.91465	0.74639
CP/Q3B	K=30, epsilon=10%	0.7624	0.7313	0.9398	**0.03862**	0.23640	**0.04919**
CP/Q3B	K=30, epsilon=10%, N=2	0.6068	0.6167	0.6395	0.99938	0.99561	0.99992
CP/Q3B	K=30, epsilon=10%, N=4	0.7137	0.7185	0.8073	0.43800	0.40798	0.26652
CP/Q3B	K=30, epsilon=10%, N=8	0.6891	0.6842	0.7065	0.74799	0.83474	0.96879
CP/Q3B	K=30, epsilon=20%	0.6764	0.6853	0.7810	0.85846	0.83474	0.44275
CP/Q3B	K=30, epsilon=30%	0.6915	0.6723	0.7362	0.72300	0.91465	0.80599
CP/Q4B	K=10, epsilon=10%	0.8937	0.8849	0.9488	0.11659	0.30004	0.47015
CP/Q4B	K=10, epsilon=10%, N=2	0.8178	0.8127	0.8665	0.81215	0.79433	0.95327
CP/Q4B	K=10, epsilon=10%, N=4	0.8343	0.8256	0.9227	0.67360	0.73264	0.77671
CP/Q4B	K=10, epsilon=10%, N=8	0.8277	0.8452	0.8709	0.73327	0.53908	0.95327
CP/Q4B	K=10, epsilon=20%	0.8435	0.8644	0.8684	0.58162	0.44469	0.95327
CP/Q4B	K=10, epsilon=30%	0.8349	0.8417	0.9023	0.66782	0.59354	0.89254
CP/Q4B	K=20, epsilon=10%	0.9047	0.9108	0.9324	0.06266	0.21906	0.74454
CP/Q4B	K=20, epsilon=20%	0.8554	0.8306	0.9387	0.45456	0.71718	0.70924
CP/Q4B	K=20, epsilon=30%	0.8946	0.8728	0.9376	0.11157	0.40724	0.74454
CP/Q4B	K=30, epsilon=10%	0.8914	0.8880	0.9376	0.12996	0.30004	0.74454
CP/Q4B	K=30, epsilon=20%	0.8285	0.8531	0.9191	0.72698	0.50124	0.79139
CP/Q4B	K=30, epsilon=30%	0.8897	0.8956	0.9525	0.14132	0.25001	0.33831
CP/GPT20B	K=10, epsilon=20%	0.9569	0.9547	0.9645	0.94240	0.98057	0.96825
CP/GPT20B	K=10, epsilon=30%	0.9472	0.9543	0.9563	0.99767	0.99127	0.99924
CP/GPT20B	K=20, epsilon=20%	0.9517	0.9526	0.9603	0.98826	0.99127	0.99868
CP/GPT20B	K=20, epsilon=30%	0.9575	0.9566	0.9701	0.93306	0.91425	0.89798
CP/GPT-120B	K=10, epsilon=20%	0.9581	0.9537	0.9765	0.77100	0.96500	0.59900
CP/GPT-120B	K=10, epsilon=30%	0.9596	0.9602	0.9650	0.68800	0.25300	0.85800
CP/GPT-120B	K=20, epsilon=20%	0.9482	0.9525	0.9558	0.99900	0.98000	1.00000
CP/GPT-120B	K=20, epsilon=30%	0.9539	0.9517	0.9725	0.94900	0.99200	0.77900
HT/Q3B	K=10, epsilon=10%	0.0646	0.0800	0.1010	0.79999	0.40512	0.92260
HT/Q3B	K=10, epsilon=20%	0.1114	0.0870	0.2170	0.41120	0.40512	0.55556
HT/Q3B	K=10, epsilon=30%	0.1268	0.0910	0.2550	0.30544	0.38619	0.44074
HT/Q3B	K=20, epsilon=10%	0.1452	0.0760	0.4200	0.19958	0.42335	0.14172
HT/Q3B	K=20, epsilon=20%	0.2366	0.2170	0.4250	**0.01109**	**0.02685**	0.14172
HT/Q3B	K=20, epsilon=20%, N=2	0.1948	0.1500	0.3390	**0.04961**	0.08385	0.30511
HT/Q3B	K=20, epsilon=20%, N=4	0.2174	0.1670	0.4240	**0.02292**	**0.04405**	0.14172
HT/Q3B	K=20, epsilon=20%, N=8	0.1562	0.1060	0.4070	0.15006	0.26568	0.18496
HT/Q3B	K=20, epsilon=30%	0.1078	0.0870	0.2170	0.43784	0.40512	0.55556
HT/Q3B	K=30, epsilon=10%	0.1222	0.0990	0.3390	0.33535	0.31593	0.30511
HT/Q3B	K=30, epsilon=20%	0.1248	0.0740	0.3390	0.31798	0.44176	0.30511
HT/Q3B	K=30, epsilon=30%	0.1126	0.0940	0.2170	0.40253	0.36810	0.55556
HT/Q4B	K=10, epsilon=10%	0.3002	0.2220	0.8160	0.57368	0.76230	**0.04812**
HT/Q4B	K=10, epsilon=10%, N=2	0.4094	0.3200	0.7900	0.15134	0.33432	0.14090
HT/Q4B	K=10, epsilon=10%, N=4	0.2684	0.2510	0.3940	0.71737	0.69831	0.80732
HT/Q4B	K=10, epsilon=10%, N=8	0.4366	0.3520	0.6850	0.09469	0.25189	0.30433
HT/Q4B	K=10, epsilon=20%	0.4376	0.4200	0.5810	0.09286	0.11447	0.50308
HT/Q4B	K=10, epsilon=30%	0.4384	0.4520	0.5850	0.09169	0.07421	0.50308
HT/Q4B	K=20, epsilon=10%	0.3442	0.3420	0.5760	0.37269	0.28405	0.50308
HT/Q4B	K=20, epsilon=20%	0.2968	0.2740	0.5000	0.58955	0.59139	0.55733
HT/Q4B	K=20, epsilon=30%	0.2426	0.2010	0.4380	0.82032	0.84840	0.74780
HT/Q4B	K=30, epsilon=10%	0.3452	0.2710	0.6180	0.36846	0.64660	0.37579
HT/Q4B	K=30, epsilon=20%	0.2946	0.2740	0.6490	0.60011	0.59139	0.30433
HT/Q4B	K=30, epsilon=30%	0.2166	0.2170	0.3490	0.89929	0.79377	0.89351
HT/GPT20B	K=10, epsilon=20%	0.8276	0.8147	0.9080	0.56379	0.78986	0.59982
HT/GPT20B	K=10, epsilon=20%, N=2	0.8500	0.8334	0.9223	0.34271	0.68064	0.41158
HT/GPT20B	K=10, epsilon=20%, N=4	0.8381	0.8648	0.9028	0.46355	0.43606	0.67431
HT/GPT20B	K=10, epsilon=30%	0.8824	0.8861	0.9086	0.08463	0.31523	0.59982
HT/GPT20B	K=20, epsilon=20%	0.7643	0.7957	0.8911	0.93140	0.83698	0.78826
HT/GPT20B	K=20, epsilon=30%	0.8431	0.8483	0.8722	0.41274	0.49768	0.94127
HT/GPT-120B	K=10, epsilon=20%	0.7440	0.7201	0.8199	0.73000	0.68400	0.92100

Continued on next page

Table 6 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
HT/GPT-120B	K=10, epsilon=30%	0.7594	0.7185	0.9333	0.62100	0.68400	0.28600
HT/GPT-120B	K=20, epsilon=20%	0.8084	0.8240	0.9350	0.22700	0.30700	0.28600
HT/GPT-120B	K=20, epsilon=30%	0.7667	0.7617	0.8092	0.55900	0.43700	0.92100
SAI/Q3B	K=10, epsilon=10%	0.9430	0.9457	0.9470	0.36388	0.40912	0.93558
SAI/Q3B	K=10, epsilon=20%	0.9425	0.9405	0.9512	0.39766	0.61204	0.74869
SAI/Q3B	K=10, epsilon=30%	0.9462	0.9499	0.9525	0.17781	0.17893	0.47375
SAI/Q3B	K=20, epsilon=10%	0.9458	0.9499	0.9617	0.19775	0.17893	**0.04840**
SAI/Q3B	K=20, epsilon=10%, N=2	0.9407	0.9497	0.9576	0.51850	0.17893	0.22680
SAI/Q3B	K=20, epsilon=10%, N=4	0.9354	0.9361	0.9463	0.81608	0.83689	0.95114
SAI/Q3B	K=20, epsilon=10%, N=8	0.9442	0.9473	0.9639	0.28502	0.33573	**0.04840**
SAI/Q3B	K=20, epsilon=20%	0.9466	0.9510	0.9518	0.15848	0.10397	0.60877
SAI/Q3B	K=20, epsilon=30%	0.9342	0.9365	0.9475	0.85971	0.83689	0.93004
SAI/Q3B	K=30, epsilon=10%	0.9433	0.9448	0.9511	0.34454	0.44545	0.74869
SAI/Q3B	K=30, epsilon=20%	0.9405	0.9367	0.9522	0.52897	0.82340	0.58486
SAI/Q3B	K=30, epsilon=30%	0.9384	0.9360	0.9567	0.66251	0.83689	0.22680
SAI/Q4B	K=10, epsilon=10%	0.9475	0.9484	0.9533	0.41961	0.40567	0.80509
SAI/Q4B	K=10, epsilon=20%	0.9462	0.9463	0.9580	0.50432	0.44256	0.72794
SAI/Q4B	K=10, epsilon=30%	0.9530	0.9497	0.9702	0.11903	0.38751	0.09589
SAI/Q4B	K=10, epsilon=30%, N=2	0.9480	0.9463	0.9524	0.38582	0.44256	0.85371
SAI/Q4B	K=10, epsilon=30%, N=4	0.9465	0.9447	0.9583	0.48515	0.51771	0.71010
SAI/Q4B	K=10, epsilon=30%, N=8	0.9408	0.9397	0.9481	0.82553	0.83745	0.95491
SAI/Q4B	K=20, epsilon=10%	0.9538	0.9523	0.9648	0.09218	0.18901	0.30253
SAI/Q4B	K=20, epsilon=20%	0.9473	0.9462	0.9571	0.42745	0.46077	0.74515
SAI/Q4B	K=20, epsilon=30%	0.9396	0.9406	0.9565	0.87131	0.74802	0.74515
SAI/Q4B	K=30, epsilon=10%	0.9417	0.9390	0.9559	0.77926	0.83745	0.76110
SAI/Q4B	K=30, epsilon=20%	0.9414	0.9394	0.9555	0.79521	0.83745	0.76110
SAI/Q4B	K=30, epsilon=30%	0.9476	0.9466	0.9517	0.41308	0.44256	0.87386
SAI/GPT20B	K=10, epsilon=10%	0.9276	0.9274	0.9385	0.92757	0.62422	0.97795
SAI/GPT20B	K=10, epsilon=20%	0.9419	0.9336	0.9684	0.49269	0.62422	0.59723
SAI/GPT20B	K=10, epsilon=30%	0.9417	0.9387	0.9606	0.49954	0.56195	0.78812
SAI/GPT20B	K=20, epsilon=10%	0.9486	0.9427	0.9696	0.24124	0.50014	0.51003
SAI/GPT20B	K=20, epsilon=20%	0.9457	0.9488	0.9602	0.34475	0.43836	0.78812
SAI/GPT20B	K=20, epsilon=30%	0.9266	0.9272	0.9485	0.94348	0.68273	0.95708
SAI/GPT20B	K=30, epsilon=10%	0.9393	0.9326	0.9540	0.59249	0.62422	0.86877
SAI/GPT20B	K=30, epsilon=20%	0.9255	0.9290	0.9348	0.95767	0.62422	0.98511
SAI/GPT20B	K=30, epsilon=30%	0.9375	0.9361	0.9619	0.66309	0.62422	0.73484
SAI/GPT-120B	K=10, epsilon=20%	0.9253	0.9268	0.9318	0.62300	0.50200	0.82400
SAI/GPT-120B	K=10, epsilon=30%	0.9330	0.9328	0.9433	0.10500	0.13400	0.17000
SAI/GPT-120B	K=20, epsilon=20%	0.9242	0.9273	0.9340	0.69000	0.44900	0.78500
SAI/GPT-120B	K=20, epsilon=30%	0.9279	0.9268	0.9413	0.42900	0.50200	0.42300

Table 7: **Epsilon-greedy + single child ($N = 1$).** Uses the epsilon-greedy harness with one generated child per update step.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	K=30, epsilon=10%	0.6667	0.6936	0.7417	0.91621	0.76343	0.76270
CP/Q4B	K=10, epsilon=10%	0.8240	0.7835	0.9463	0.76467	0.87456	0.52763
CP/GPT20B	K=1, epsilon=30%	0.9612	0.9644	0.9665	0.83787	0.68277	0.96825
HT/Q3B	K=20, epsilon=20%	0.2090	0.2550	0.3330	**0.03107**	**0.01162**	0.30511
HT/Q4B	K=10, epsilon=10%	0.2328	0.2070	0.4280	0.85351	0.82232	0.74780
HT/GPT20B	K=10, epsilon=20%	0.9175	0.9530	0.9673	**0.00213**	$< 10^{-5}$	$< 10^{-5}$
SAI/Q3B	K=20, epsilon=10%	0.9297	0.9337	0.9495	0.96352	0.86292	0.84587
SAI/Q4B	K=10, epsilon=30%	0.9452	0.9473	0.9567	0.57139	0.44256	0.74515
SAI/GPT20B	K=20, epsilon=0%	0.9446	0.9485	0.9646	0.38498	0.43836	0.73484

Table 8: **Basic OpenEvolve (one island).** Canonical one-island OpenEvolve runs.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	K=30, epsilon=10%	0.4423	0.4246	0.5632	1.00000	1.00000	1.00000

Continued on next page

Table 8 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q4B	K=10, epsilon=10%	0.7030	0.7028	0.8029	0.99995	0.99991	0.99692
CP/GPT20B	K=1, epsilon=30%	0.9438	0.9551	0.9689	0.99943	0.91425	0.94119
CP/GPT-120B	K=1, epsilon=0%	0.9532	0.9581	0.9648	0.96800	0.25300	0.85800
HT/Q3B	K=20, epsilon=20%	0.0036	0.0000	0.0168	1.00000	1.00000	0.99993
HT/Q4B	K=10, epsilon=10%	0.5890	0.6770	0.7810	**0.00214**	**0.00352**	0.14090
HT/GPT20B	K=10, epsilon=20%	0.6522	0.6572	0.8304	0.99976	0.99964	0.98996
HT/GPT-120B	K=1, epsilon=0%	0.7535	0.6967	0.9134	0.66200	0.90600	0.50800
SAI/Q3B	K=20, epsilon=10%	0.9135	0.9265	0.9318	1.00000	0.96381	0.99898
SAI/Q4B	K=10, epsilon=30%	0.9121	0.9156	0.9364	1.00000	0.99976	0.99982
SAI/GPT20B	K=20, epsilon=0%	0.9257	0.9256	0.9411	0.95488	0.73931	0.96924
SAI/GPT-120B	K=1, epsilon=0%	0.9183	0.9139	0.9330	0.96300	0.99100	0.78500

Table 9: **OpenEvolve + multiple islands.** Extends OpenEvolve with multiple interacting islands.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	4 islands, K=30, epsilon=10%	0.6114	0.6167	0.6392	0.99888	0.99561	0.99992
CP/Q4B	4 islands, K=10, epsilon=10%	0.7398	0.7440	0.8146	0.99866	0.99141	0.99579
CP/GPT20B	3 islands, K=1, epsilon=30%	0.9534	0.9509	0.9709	0.97995	0.99127	0.86832
CP/GPT-120B	4 islands, K=1, epsilon=0%	0.9587	0.9578	0.9693	0.73600	0.31200	0.81900
HT/Q3B	4 islands, K=20, epsilon=20%	0.0000	0.0000	0.0002	1.00000	1.00000	1.00000
HT/Q4B	4 islands, K=10, epsilon=10%	0.1164	0.0360	0.3840	0.99823	0.99977	0.83360
HT/GPT20B	3 islands, K=10, epsilon=20%	0.8916	0.9239	0.9567	**0.04449**	**0.00855**	$< 10^{-5}$
HT/GPT-120B	4 islands, K=1, epsilon=0%	0.6943	0.6669	0.8829	0.95100	0.98200	0.66800
SAI/Q3B	4 islands, K=20, epsilon=10%	0.8943	0.8897	0.9135	1.00000	1.00000	1.00000
SAI/Q4B	4 islands, K=10, epsilon=30%	0.9127	0.9129	0.9192	1.00000	1.00000	1.00000
SAI/GPT20B	4 islands, K=20, epsilon=0%	0.9378	0.9377	0.9424	0.65028	0.56195	0.96924
SAI/GPT-120B	4 islands, K=1, epsilon=0%	0.9183	0.9137	0.9270	0.96300	0.99100	0.95600

Table 10: **UCT.** Uses UCT parent selection, including breadth–depth and parent-count ablations.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	P=1, C=0.1	0.7410	0.7230	0.7850	0.13680	0.31851	0.44275
CP/Q3B	P=1, C=0.1, N=2	0.6726	0.6610	0.7980	0.88342	0.94107	0.30482
CP/Q3B	P=1, C=0.1, N=4	0.6240	0.6330	0.7120	0.99619	0.98521	0.95786
CP/Q3B	P=1, C=0.1, N=8	0.6838	0.7000	0.7950	0.79952	0.69830	0.34064
CP/Q3B	P=1, C=1	0.6876	0.6960	0.7760	0.76323	0.76343	0.47341
CP/Q3B	P=1, C=1, N=2	0.6266	0.6780	0.7160	0.99516	0.87168	0.95382
CP/Q3B	P=1, C=1, N=4	0.6478	0.6390	0.7250	0.97487	0.97756	0.92146
CP/Q3B	P=1, C=1, N=8	0.6032	0.6170	0.6910	0.99966	0.99561	0.99655
CP/Q3B	P=1, C=10	0.6788	0.7020	0.7420	0.84125	0.62712	0.76270
CP/Q3B	P=2, C=0.1, N=4	0.5320	0.5620	0.7480	1.00000	0.99884	0.74639
CP/Q3B	P=4, C=0.1, N=4	0.6296	0.7050	0.7280	0.99391	0.59158	0.91471
CP/Q3B	P=4, C=0.1, N=8	0.7160	0.7150	0.7480	0.40574	0.44443	0.74639
CP/Q4B	P=1, C=0.1	0.8160	0.8030	0.8920	0.82404	0.80883	0.90064
CP/Q4B	P=1, C=1	0.8128	0.7820	0.9310	0.84487	0.90733	0.77671
CP/Q4B	P=1, C=10	0.9076	0.9300	0.9430	0.05196	0.11381	0.52763
CP/Q4B	P=1, C=10, N=2	0.7852	0.7630	0.8450	0.95840	0.95608	0.97678
CP/Q4B	P=1, C=10, N=4	0.7522	0.7490	0.8090	0.99589	0.98200	0.99692
CP/Q4B	P=1, C=10, N=8	0.8784	0.9320	0.9380	0.22664	0.10266	0.70924
CP/Q4B	P=2, C=10, N=4	0.8376	0.8480	0.9040	0.64174	0.53908	0.89254
CP/Q4B	P=4, C=10, N=4	0.8250	0.8280	0.9200	0.75643	0.73264	0.79139
CP/Q4B	P=4, C=10, N=8	0.9058	0.9390	0.9400	0.05846	0.06536	0.57909
CP/GPT20B	P=1, C=0.1	0.9702	0.9640	0.9930	0.36936	0.73934	0.41107
CP/GPT20B	P=1, C=0.1, N=2	0.9696	0.9650	0.9930	0.40283	0.49901	0.41107
CP/GPT20B	P=1, C=0.1, N=4	0.9448	0.9500	0.9610	0.99904	0.99728	0.99868
CP/GPT20B	P=1, C=1	0.9566	0.9550	0.9640	0.94627	0.94302	0.99303
CP/GPT20B	P=1, C=10	0.9460	0.9450	0.9690	0.99850	0.99728	0.92242
CP/GPT20B	P=2, C=0.1, N=2	0.9576	0.9640	0.9700	0.93082	0.73934	0.89798
CP/GPT20B	P=4, C=0.1	0.9740	0.9730	0.9860	0.18798	0.16236	0.51342

Continued on next page

Table 10 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/GPT20B	P=4, C=0.1, N=2	0.9698	0.9720	0.9790	0.39106	0.16236	0.73659
CP/GPT-120B	P=1, C=0.1, N=4	0.9652	0.9560	0.9870	0.19000	0.86400	0.11700
CP/GPT-120B	P=2, C=0.1, N=2	0.9556	0.9560	0.9640	0.84300	0.86400	0.88000
CP/GPT-120B	P=4, C=0.1, N=2	0.9690	0.9640	0.9890	0.06900	0.20900	0.11700
HT/Q3B	P=1, C=0.1	0.1408	0.0880	0.4530	0.22304	0.40512	$< 10^{-5}$
HT/Q3B	P=1, C=1	0.0758	0.0300	0.2170	0.70197	0.91657	0.55556
HT/Q3B	P=1, C=10	0.1556	0.1680	0.3390	0.15217	**0.04405**	0.30511
HT/Q3B	P=1, C=10, N=2	0.1476	0.1260	0.4450	0.18767	0.10327	0.09626
HT/Q3B	P=1, C=10, N=4	0.0884	0.0540	0.1920	0.59141	0.66224	0.58066
HT/Q3B	P=1, C=10, N=8	0.1240	0.0710	0.4300	0.32310	0.44176	0.09626
HT/Q3B	P=2, C=10, N=4	0.1412	0.1430	0.2240	0.22071	0.09362	0.47177
HT/Q3B	P=4, C=10, N=4	0.0580	0.0160	0.2310	0.85343	0.98186	0.47177
HT/Q3B	P=4, C=10, N=8	0.1578	0.0820	0.4520	0.14372	0.40512	0.09626
HT/Q4B	P=1, C=0.1	0.2328	0.2530	0.4930	0.85351	0.69831	0.55733
HT/Q4B	P=1, C=1	0.4104	0.2970	0.7390	0.14890	0.49807	0.14090
HT/Q4B	P=1, C=1, N=2	0.2068	0.1700	0.3880	0.92248	0.91492	0.83360
HT/Q4B	P=1, C=1, N=4	0.2642	0.2490	0.4520	0.73473	0.73081	0.71236
HT/Q4B	P=1, C=1, N=8	0.2192	0.2620	0.3160	0.89298	0.66464	0.95845
HT/Q4B	P=1, C=10	0.2494	0.2090	0.3790	0.79521	0.80802	0.86636
HT/Q4B	P=2, C=1, N=4	0.5394	0.5660	0.8230	**0.00913**	**0.01798**	**0.04812**
HT/Q4B	P=4, C=1, N=4	0.1666	0.1050	0.4520	0.97927	0.98868	0.71236
HT/Q4B	P=4, C=1, N=8	0.3014	0.2590	0.6810	0.56770	0.66464	0.30433
HT/GPT20B	P=1, C=0.1	0.8878	0.9220	0.9650	0.05911	**0.00855**	$< 10^{-5}$
HT/GPT20B	P=1, C=1	0.9044	0.9170	0.9420	**0.01281**	**0.01980**	$< 10^{-5}$
HT/GPT20B	P=1, C=1, N=2	0.9010	0.9150	0.9510	**0.01876**	**0.01980**	$< 10^{-5}$
HT/GPT20B	P=1, C=1, N=4	0.8092	0.8470	0.9820	0.71614	0.49768	$< 10^{-5}$
HT/GPT20B	P=1, C=10	0.8174	0.8190	0.9610	0.65252	0.78986	$< 10^{-5}$
HT/GPT20B	P=2, C=1, N=2	0.8844	0.8940	0.9370	0.07407	0.12120	0.29467
HT/GPT20B	P=4, C=1	0.9238	0.9280	0.9640	**0.00051**	**0.00855**	$< 10^{-5}$
HT/GPT20B	P=4, C=1, N=2	0.8586	0.8730	0.8910	0.26189	0.37360	0.78826
HT/GPT-120B	P=1, C=1, N=4	0.8678	0.8380	0.9430	**0.01700**	0.15300	0.36300
HT/GPT-120B	P=2, C=1, N=2	0.7874	0.8480	0.8860	0.34900	0.12600	0.66500
HT/GPT-120B	P=4, C=1, N=2	0.8586	0.8330	0.9460	**0.03500**	0.18100	0.19700
SAI/Q3B	P=1, C=0.1	0.9340	0.9370	0.9510	0.86687	0.80903	0.76539
SAI/Q3B	P=1, C=1	0.9440	0.9430	0.9540	0.29986	0.52074	0.34117
SAI/Q3B	P=1, C=1, N=2	0.9378	0.9340	0.9610	0.69619	0.85026	0.14083
SAI/Q3B	P=1, C=1, N=4	0.9380	0.9390	0.9480	0.68483	0.73268	0.90266
SAI/Q3B	P=1, C=1, N=8	0.9390	0.9410	0.9460	0.62576	0.57561	0.95114
SAI/Q3B	P=1, C=10	0.9394	0.9450	0.9520	0.60007	0.44545	0.58486
SAI/Q3B	P=2, C=1, N=4	0.9270	0.9150	0.9620	0.98706	0.99726	**0.04840**
SAI/Q3B	P=4, C=1, N=4	0.9266	0.9400	0.9470	0.98909	0.66541	0.93558
SAI/Q4B	P=1, C=0.1	0.9476	0.9430	0.9650	0.41034	0.61084	0.30253
SAI/Q4B	P=1, C=0.1, N=2	0.9230	0.9280	0.9670	0.99992	0.99338	0.18164
SAI/Q4B	P=1, C=0.1, N=4	0.9442	0.9390	0.9660	0.63883	0.83745	0.26412
SAI/Q4B	P=1, C=0.1, N=8	0.9444	0.9390	0.9600	0.62564	0.83745	0.58001
SAI/Q4B	P=1, C=1	0.9396	0.9390	0.9450	0.86991	0.83745	0.96881
SAI/Q4B	P=1, C=10	0.9416	0.9380	0.9550	0.78632	0.90631	0.76110
SAI/Q4B	P=2, C=0.1, N=4	0.8278	0.9420	0.9440	1.00000	0.66493	0.97445
SAI/Q4B	P=4, C=0.1, N=4	0.9300	0.9390	0.9460	0.99624	0.83745	0.96254
SAI/GPT20B	P=1, C=0.1	0.9570	0.9570	0.9660	0.05656	0.16345	0.67160
SAI/GPT20B	P=1, C=0.1, N=2	0.9406	0.9430	0.9590	0.54298	0.50014	0.78812
SAI/GPT20B	P=1, C=0.1, N=4	0.9432	0.9430	0.9730	0.44120	0.50014	0.29137
SAI/GPT20B	P=1, C=1	0.9444	0.9480	0.9610	0.39458	0.50014	0.78812
SAI/GPT20B	P=1, C=10	0.9320	0.9340	0.9450	0.83972	0.62422	0.96924
SAI/GPT20B	P=2, C=0.1, N=2	0.9374	0.9400	0.9590	0.66544	0.56195	0.78812
SAI/GPT20B	P=4, C=0.1, N=2	0.9474	0.9640	0.9710	0.27980	0.08733	0.40929
SAI/GPT-120B	P=1, C=0.1, N=4	0.9316	0.9270	0.9590	0.19800	0.38500	0.13700
SAI/GPT-120B	P=2, C=0.1, N=2	0.9276	0.9160	0.9690	0.44000	0.97700	$< 10^{-5}$
SAI/GPT-120B	P=4, C=0.1, N=2	0.9314	0.9300	0.9480	0.20400	0.22300	0.25700

Table 11: **PUCT.** Uses PUCT parent selection with one parent program ($P = 1$).

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	P=1, C=0.1	0.6568	0.6540	0.8020	0.95431	0.95594	0.30482

Continued on next page

Table 11 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	P=1, C=1	0.7098	0.6850	0.9110	0.49005	0.83474	**0.04919**
CP/Q3B	P=1, C=1, N=2	0.5992	0.6050	0.7160	0.99979	0.99803	0.95382
CP/Q3B	P=1, C=1, N=4	0.6266	0.6440	0.7340	0.99516	0.97289	0.84270
CP/Q3B	P=1, C=1, N=8	0.7626	0.7380	0.8740	**0.03793**	0.12700	**0.04919**
CP/Q3B	P=1, C=10	0.7378	0.7310	0.8160	0.16189	0.23640	0.22610
CP/Q3B	P=1, C=10, N=2	0.5504	0.5830	0.6870	1.00000	0.99884	0.99751
CP/Q3B	P=1, C=10, N=4	0.6248	0.6840	0.7030	0.99586	0.83474	0.98546
CP/Q3B	P=1, C=10, N=8	0.6650	0.6760	0.7480	0.92427	0.88334	0.74639
CP/Q3B	P=1, C=100	0.6886	0.7110	0.7270	0.75310	0.44443	0.92146
CP/Q4B	P=1, C=0.1	0.8696	0.8540	0.9510	0.30819	0.48266	0.37270
CP/Q4B	P=1, C=1	0.8376	0.8200	0.9420	0.64174	0.76412	0.55379
CP/Q4B	P=1, C=10	0.8758	0.8750	0.9410	0.24919	0.38896	0.57909
CP/Q4B	P=1, C=10, N=2	0.7670	0.7730	0.8560	0.98711	0.93390	0.96492
CP/Q4B	P=1, C=10, N=4	0.7548	0.7470	0.8240	0.99499	0.98200	0.99417
CP/Q4B	P=1, C=10, N=8	0.7866	0.7670	0.9410	0.95519	0.94164	0.57909
CP/GPT20B	P=1, C=0.1	0.9464	0.9630	0.9660	0.99832	0.83853	0.96825
CP/GPT20B	P=1, C=1	0.9658	0.9640	0.9980	0.62339	0.73934	0.15662
CP/GPT20B	P=1, C=10	0.9714	0.9660	0.9840	0.30619	0.49901	0.51342
CP/GPT20B	P=1, C=10, N=2	0.9684	0.9640	0.9950	0.47275	0.73934	0.41107
CP/GPT20B	P=1, C=10, N=4	0.9782	0.9780	0.9920	0.06822	0.08764	0.41107
CP/GPT-120B	P=1, C=10, N=4	0.9656	0.9630	0.9870	0.17700	0.20900	0.11700
HT/Q3B	P=1, C=0.1	0.1034	0.0420	0.3390	0.47112	0.82297	0.30511
HT/Q3B	P=1, C=1	0.2224	0.1130	0.4390	**0.01923**	0.20401	0.09626
HT/Q3B	P=1, C=1, N=2	0.0844	0.0640	0.2000	0.62629	0.55420	0.55556
HT/Q3B	P=1, C=1, N=4	0.0810	0.0580	0.2170	0.65577	0.60874	0.55556
HT/Q3B	P=1, C=1, N=8	0.0834	0.1160	0.1630	0.63452	0.18953	0.62742
HT/Q3B	P=1, C=10	0.1028	0.0920	0.2330	0.47551	0.38619	0.47177
HT/Q4B	P=1, C=0.1	0.2102	0.1820	0.4520	0.91501	0.89516	0.71236
HT/Q4B	P=1, C=1	0.3318	0.2980	0.5080	0.42711	0.49807	0.55733
HT/Q4B	P=1, C=1, N=2	0.2268	0.2560	0.4660	0.87116	0.66464	0.65256
HT/Q4B	P=1, C=1, N=4	0.2552	0.2910	0.4450	0.77199	0.51661	0.73052
HT/Q4B	P=1, C=1, N=8	0.2922	0.2570	0.6430	0.61081	0.66464	0.30433
HT/Q4B	P=1, C=10	0.3062	0.3750	0.4620	0.54582	0.22132	0.65256
HT/GPT20B	P=1, C=0.1	0.8398	0.8580	0.8970	0.44654	0.43606	0.67431
HT/GPT20B	P=1, C=1	0.8986	0.8970	0.9790	**0.02391**	0.05778	$< 10^{-5}$
HT/GPT20B	P=1, C=1, N=2	0.8764	0.8920	0.9640	0.12006	0.12120	$< 10^{-5}$
HT/GPT20B	P=1, C=1, N=4	0.9316	0.9490	0.9730	**0.00004**	$< 10^{-5}$	$< 10^{-5}$
HT/GPT20B	P=1, C=10	0.8062	0.8460	0.9180	0.73715	0.49768	0.51170
HT/GPT-120B	P=1, C=1, N=4	0.8388	0.8780	0.9690	0.08700	0.05300	$< 10^{-5}$
SAI/Q3B	P=1, C=0.1	0.9346	0.9460	0.9520	0.84563	0.40912	0.58486
SAI/Q3B	P=1, C=1	0.9448	0.9450	0.9580	0.25160	0.44545	0.22680
SAI/Q3B	P=1, C=10	0.9514	0.9510	0.9600	**0.02104**	0.10397	0.18462
SAI/Q3B	P=1, C=10, N=2	0.9196	0.9190	0.9390	0.99965	0.99189	0.99342
SAI/Q3B	P=1, C=10, N=4	0.9190	0.9140	0.9520	0.99972	0.99895	0.58486
SAI/Q3B	P=1, C=10, N=8	0.9222	0.9320	0.9430	0.99844	0.89762	0.97267
SAI/Q4B	P=1, C=0.1	0.9434	0.9410	0.9630	0.68864	0.74802	0.37556
SAI/Q4B	P=1, C=1	0.9436	0.9470	0.9490	0.67668	0.44256	0.94565
SAI/Q4B	P=1, C=10	0.9516	0.9470	0.9680	0.17475	0.44256	0.14021
SAI/Q4B	P=1, C=10, N=2	0.9394	0.9440	0.9650	0.87651	0.53645	0.30253
SAI/Q4B	P=1, C=10, N=4	0.9334	0.9400	0.9520	0.98363	0.80900	0.86387
SAI/Q4B	P=1, C=10, N=8	0.9438	0.9430	0.9600	0.66462	0.61084	0.58001
SAI/GPT20B	P=1, C=0.1	0.9508	0.9610	0.9720	0.17744	0.12218	0.29137
SAI/GPT20B	P=1, C=0.1, N=2	0.9530	0.9510	0.9700	0.12568	0.37733	0.40929
SAI/GPT20B	P=1, C=0.1, N=4	0.9390	0.9510	0.9700	0.60347	0.37733	0.40929
SAI/GPT20B	P=1, C=1	0.9488	0.9630	0.9660	0.23407	0.08733	0.67160
SAI/GPT20B	P=1, C=10	0.9376	0.9360	0.9710	0.65755	0.62422	0.40929
SAI/GPT-120B	P=1, C=0.1, N=4	0.9274	0.9310	0.9520	0.46000	0.17900	0.13700

Table 12: **PUCT + multiple parents (TTT-Discover).** Extends PUCT to multiple parent programs ($P > 1$), corresponding to the TTT-Discover endpoint.

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	P=2, C=10, N=4	0.6116	0.6330	0.6720	0.99885	0.98521	0.99935
CP/Q3B	P=4, C=10, N=4	0.6708	0.6900	0.8470	0.89449	0.80736	0.09572

Continued on next page

Table 12 (continued)

Pair	Configuration	Observed statistic			One-sided p-value		
		Mean	Median	Max	p_{mean}	p_{median}	$p_{\max}$
CP/Q3B	P=4, C=10, N=8	0.7276	0.7320	0.8120	0.26111	0.22147	0.26652
CP/Q4B	P=2, C=10, N=4	0.8248	0.8060	0.9390	0.75796	0.80883	0.69032
CP/Q4B	P=4, C=10, N=4	0.7424	0.7540	0.8100	0.99827	0.98200	0.99692
CP/Q4B	P=4, C=10, N=8	0.8820	0.8790	0.9490	0.19675	0.33376	0.47015
CP/GPT20B	P=2, C=10, N=2	0.9726	0.9700	0.9970	0.24804	0.26098	0.15662
CP/GPT20B	P=4, C=10	0.9792	0.9950	0.9980	0.05085	**0.00870**	0.15662
CP/GPT20B	P=4, C=10, N=2	0.9750	0.9800	0.9970	0.15236	0.05836	0.15662
CP/GPT-120B	P=2, C=10, N=2	0.9554	0.9570	0.9580	0.85500	0.51500	0.88000
CP/GPT-120B	P=4, C=10, N=2	0.9554	0.9570	0.9640	0.85500	0.51500	0.88000
HT/Q3B	P=2, C=1, N=4	0.0702	0.0700	0.1490	0.75197	0.47918	0.72934
HT/Q3B	P=4, C=1, N=4	0.1188	0.0530	0.3390	0.35886	0.67919	0.30511
HT/Q3B	P=4, C=1, N=8	0.0816	0.0430	0.1880	0.65085	0.82297	0.58066
HT/Q4B	P=2, C=1, N=4	0.2158	0.1520	0.4340	0.90142	0.95627	0.74780
HT/Q4B	P=4, C=1, N=4	0.3206	0.2970	0.6550	0.47980	0.49807	0.30433
HT/Q4B	P=4, C=1, N=8	0.4342	0.4170	0.6590	0.09900	0.11447	0.30433
HT/GPT20B	P=2, C=1, N=2	0.8298	0.8590	0.9530	0.54280	0.43606	$< 10^{-5}$
HT/GPT20B	P=4, C=1	0.9268	0.9410	0.9970	**0.00027**	**0.00258**	$< 10^{-5}$
HT/GPT20B	P=4, C=1, N=2	0.7952	0.7850	0.8440	0.80663	0.87796	0.97799
HT/GPT-120B	P=2, C=1, N=2	0.8862	0.8950	0.9570	**0.00500**	0.05300	$< 10^{-5}$
HT/GPT-120B	P=4, C=1, N=2	0.7728	0.8120	0.8680	0.46200	0.31800	0.75000
SAI/Q3B	P=2, C=10, N=4	0.9408	0.9460	0.9620	0.50924	0.40912	**0.04840**
SAI/Q3B	P=4, C=10, N=4	0.9342	0.9360	0.9480	0.85971	0.83689	0.90266
SAI/Q4B	P=2, C=10, N=4	0.9508	0.9490	0.9660	0.21374	0.38751	0.26412
SAI/Q4B	P=4, C=10, N=4	0.9408	0.9420	0.9630	0.82390	0.66493	0.37556
SAI/GPT20B	P=2, C=0.1, N=2	0.9514	0.9570	0.9660	0.16287	0.16345	0.67160
SAI/GPT20B	P=4, C=0.1, N=2	0.9392	0.9420	0.9550	0.59551	0.50014	0.86877
SAI/GPT-120B	P=2, C=0.1, N=2	0.9050	0.9150	0.9160	1.00000	0.99100	1.00000
SAI/GPT-120B	P=4, C=0.1, N=2	0.9354	0.9270	0.9630	0.06900	0.38500	0.13700

G Full Budget-Matched Allocation Results

Table 13 reports the full budget-matched allocation sweep corresponding to main-text Table 1. The main table summarizes the three baselines and representative single-, two-, and three-stage schedules using their cross-pair average scores. The appendix restores all schedules and the twelve pair-level mean-score columns, using the same unweighted average statistic.

The statistical comparison in this table is a simulation-based vector battle against a matched Sequential BoN reference. In each resampling iteration and for each task–model pair, we draw one score from the candidate policy's simulated outcome distribution. For pruning and ensembling policies, this score is obtained by replaying the policy against the available run pool; for the Sequential BoN reference row, it is obtained by sampling five Sequential BoN baseline runs with replacement and taking their maximum. We then independently sample five Sequential BoN baseline runs with replacement for the same task–model pair and again take the maximum. The policy and sampled baseline are compared pairwise: the policy wins the pair if its score is larger, the baseline wins if its score is larger, and exact equality is counted as a tie. Across all available task–model pairs, the iteration is decided by majority vote: the policy wins the vector battle if it wins more pairs than the baseline, the baseline wins if it wins more pairs, and the iteration is tied otherwise.

Repeating this procedure over many resampling iterations gives the win, loss, and tie rates reported for each strategy. The pair-level entries in the table are the mean simulated policy scores for each task–model pair, and the Avg. column is their unweighted average across the twelve pairs. To assess whether a strategy wins decisive vector battles more often than expected by chance, we use a one-sided binomial tail test after discarding vector-level ties. The null hypothesis is that, conditional on a decisive vector battle, the policy and baseline are equally likely to win. Thus, if a strategy wins W decisive battles and the sampled baseline wins L, the p-value is the upper-tail probability of observing at least W wins under a $\mathrm{Binomial}(W + L, 0.5)$ distribution, with the analogous tail probability used when summarizing baseline wins. The Sequential BoN reference row therefore serves as a baseline-vs-baseline check: its nearly symmetric win rates and non-significant p-value indicate no evidence that either sampled side systematically wins.

Table 13: **Budget-matched evaluation.** Entries report mean final scores over $100,000$ empirical resampling trials under a five-full-run-equivalent budget. All score values are reported as percentages, obtained by multiplying the original values by 100. Winrate is the number of the twelve model–problem pairs on which a strategy has a higher reported score than the Sequential BoN reference; ties do not count as wins. Higher is better; bold indicates the best value in each score column and the highest winrate.

Strategy / schedule	CP Q3B	CP Q4B	CP GPT 20B	CP GPT 120B	HT Q3B	HT Q4B	HT GPT 20B	HT GPT 120B	SA Q3B	SA Q4B	SA GPT 20B	SA GPT 120B	Winrate (/12)
Sequential BoN reference, 5 full	78.42	93.60	98.63	97.56	24.50	**56.05**	91.29	89.80	95.38	96.09	**96.59**	94.27	N/A
Single-harness baseline, 5 full	74.46	89.61	97.77	96.90	25.39	43.90	91.69	89.81	95.06	95.47	95.65	94.20	3/12
Unpruned harness portfolio, 5 full	77.33	93.13	98.37	97.37	29.11	51.08	94.01	91.89	95.36	95.97	96.15	94.73	4/12
Single-stage pruning at 25%													
25%: 17→1	77.74	94.78	98.68	97.39	35.82	45.48	93.92	92.57	95.44	96.30	96.03	94.27	7/12
25%: 8→4	78.40	94.02	98.50	97.54	32.03	52.32	94.65	92.91	95.48	96.16	96.28	94.76	7/12
25%: 14→2	78.68	**94.79**	98.72	97.60	**35.93**	48.59	94.60	93.55	95.49	**96.32**	96.26	94.29	**10/12**
Single-stage pruning at 50%													
50%: 9→1	78.13	94.07	98.50	97.58	32.97	50.58	94.25	92.73	95.45	96.17	96.18	94.53	8/12
50%: 6→4	77.75	93.56	98.49	97.48	30.55	52.85	94.46	92.61	95.44	96.08	96.21	94.71	5/12
50%: 8→2	78.55	94.06	98.59	97.59	32.69	52.29	94.65	93.07	95.48	96.18	96.25	94.50	9/12
Two-stage pruning at 25%/50%													
7→5→4	78.22	93.85	98.49	97.54	31.31	53.10	94.55	92.56	95.47	96.14	96.27	**94.78**	7/12
8→6→3	78.76	94.09	98.59	97.62	32.51	53.27	94.79	93.00	95.49	96.20	96.31	94.62	9/12
9→5→3	78.99	94.26	98.64	97.65	33.19	52.92	94.91	93.29	95.50	96.22	96.32	94.61	**10/12**
10→4→3	79.01	94.41	98.66	97.64	33.77	52.00	95.02	93.60	95.51	96.25	96.31	94.49	**10/12**
10→6→2	**79.24**	94.39	98.67	97.69	34.03	52.87	94.90	93.40	95.50	96.26	96.32	94.62	**10/12**
10→8→1	78.52	94.23	98.55	97.63	33.48	51.35	94.44	92.96	95.47	96.21	96.23	94.63	9/12
12→6→1	78.79	94.48	98.66	97.69	34.73	51.29	94.82	93.44	95.49	96.25	96.23	94.62	**10/12**
14→4→1	78.56	94.67	98.72	97.69	35.58	50.17	95.05	**93.78**	**95.51**	96.28	96.22	94.38	**10/12**
Three-stage pruning at 25%/50%/75%													
7→6→4→3	78.29	93.89	98.54	97.56	31.64	53.73	94.69	92.83	95.48	96.16	96.26	94.67	7/12
8→5→4→3	78.66	94.08	98.57	97.61	32.37	53.51	94.79	92.96	95.49	96.19	96.29	94.71	9/12
8→6→5→1	78.43	94.04	98.50	97.57	32.64	54.21	94.63	92.70	95.47	96.18	96.21	94.71	9/12
10→5→3→2	79.23	94.42	98.68	97.69	33.99	53.09	95.07	93.50	95.51	96.26	96.31	94.58	**10/12**
10→6→3→1	79.08	94.39	98.63	97.69	34.02	53.62	94.96	93.25	95.49	96.25	96.28	94.66	9/12
12→5→2→1	79.17	94.58	**98.73**	**97.72**	35.07	52.42	**95.11**	93.62	95.51	96.28	96.26	94.51	**10/12**

Table 14: **Budget-matched evaluation for PUCT.** Entries report mean final scores over $100,000$ empirical resampling trials under a five-full-run-equivalent budget. All score values are reported as percentages, obtained by multiplying the original values by 100. Winrate is the number of the twelve model–problem pairs on which a strategy has a higher reported score than the Sequential BoN reference; ties do not count as wins. Higher is better; bold indicates the best value in each score column and the highest winrate.

Strategy / schedule	CP Q2.5	CP Q3	CP GPT 20B	CP GPT 120B	HT Q2.5	HT Q3	HT GPT 20B	HT GPT 120B	SA Q2.5	SA Q3	SA GPT 20B	SA GPT 120B	Winrate (/12)
Sequential BoN reference, 5 full	78.37	**93.58**	98.62	97.55	24.35	**55.90**	91.31	89.75	95.39	96.09	96.60	94.27	N/A
Single-harness baseline, 5 full	77.27	91.16	99.08	96.98	25.17	52.49	94.01	93.11	95.19	96.05	96.65	94.36	6/12
Unpruned harness portfolio, 5 full	79.01	91.68	99.08	96.93	24.00	49.36	95.56	93.30	95.41	95.97	96.62	94.85	7/12
Single-stage pruning at 25%													
25%: 17→1	80.09	92.98	98.91	96.94	**29.03**	49.43	94.64	93.60	95.63	96.28	96.61	94.28	**9/12**
25%: 8→4	80.33	92.61	99.27	97.33	25.53	53.34	96.40	94.75	95.48	96.20	96.74	94.88	**9/12**
25%: 14→2	**81.47**	92.92	99.10	97.51	27.96	51.99	94.86	94.61	95.57	96.23	96.79	94.22	8/12
Single-stage pruning at 50%													
50%: 9→1	80.10	93.34	98.97	97.43	25.38	52.42	95.94	94.99	95.65	96.14	96.71	93.82	8/12
50%: 6→4	80.02	92.40	99.18	97.07	24.02	50.72	96.08	93.96	95.52	96.09	96.73	94.89	7/12
50%: 8→2	80.99	93.23	99.23	97.30	25.07	52.52	96.54	94.65	**95.68**	96.21	96.76	94.05	8/12
Two-stage pruning at 25%/50%													
7→5→4	80.37	92.53	99.21	97.20	24.99	52.27	96.33	94.40	95.56	96.18	96.79	**94.94**	**9/12**
8→6→3	81.21	93.08	99.29	97.33	25.77	53.18	**96.67**	94.72	95.65	96.24	96.79	94.46	**9/12**
9→5→3	81.12	93.09	99.22	97.42	26.36	54.23	96.31	95.00	95.51	96.21	96.79	94.42	**9/12**
10→4→3	80.88	92.83	99.24	97.51	25.83	54.08	96.05	95.13	95.47	96.22	**96.80**	94.41	**9/12**
10→6→2	81.45	93.07	99.21	97.52	25.64	53.97	96.50	95.24	95.58	96.23	**96.81**	94.10	8/12
10→8→1	80.38	93.21	99.04	97.50	24.98	52.64	96.19	95.20	95.67	96.19	96.74	93.89	8/12
12→6→1	80.45	93.21	99.20	97.70	26.47	54.40	96.44	**95.65**	95.59	96.25	96.76	94.04	**9/12**
14→4→1	80.80	93.21	99.26	**97.80**	27.73	54.93	95.74	95.60	95.57	96.28	96.78	94.15	**9/12**
Three-stage pruning at 25%/50%/75%													
7→6→4→3	80.52	92.77	99.19	97.21	25.26	52.23	96.40	94.39	95.60	96.19	**96.80**	94.81	**9/12**
8→5→4→3	80.84	92.81	**99.31**	97.31	26.07	53.48	96.64	94.73	95.58	96.24	96.78	94.81	**9/12**
8→6→5→1	80.63	93.04	99.10	97.33	26.31	53.73	**96.67**	94.73	95.63	96.25	96.70	94.44	**9/12**
10→5→3→2	81.27	92.99	99.24	97.52	26.10	54.64	96.51	95.26	95.52	96.24	**96.80**	94.34	**9/12**
10→6→3→1	81.18	93.06	99.15	97.51	26.12	54.58	96.54	95.25	95.58	96.26	96.76	94.16	8/12
12→5→2→1	80.72	93.22	99.26	97.69	26.85	55.12	96.37	95.57	95.55	**96.28**	96.76	94.12	**9/12**

Table 15: **Budget-matched evaluation for UCT.** Entries report mean final scores over $100,000$ empirical resampling trials under a five-full-run-equivalent budget. All score values are reported as percentages, obtained by multiplying the original values by 100. Winrate is the number of the twelve model–problem pairs on which a strategy has a higher reported score than the Sequential BoN reference; ties do not count as wins. Higher is better; bold indicates the best value in each score column and the highest winrate.

Strategy / schedule	CP Q2.5	CP Q3	CP GPT 20B	CP GPT 120B	HT Q2.5	HT Q3	HT GPT 20B	HT GPT 120B	SA Q2.5	SA Q3	SA GPT 20B	SA GPT 120B	Winrate (/12)
Sequential BoN reference, 5 full	**78.45**	**93.56**	**98.62**	97.58	24.36	56.19	91.34	89.87	95.38	**96.10**	**96.60**	94.28	N/A
Single-harness baseline, 5 full	74.55	90.27	97.69	97.99	33.24	52.44	94.90	92.47	95.27	95.63	96.16	**95.86**	5/12
Unpruned harness portfolio, 5 full	74.45	92.56	97.91	97.98	29.08	56.97	94.78	92.53	95.23	95.42	96.18	95.32	6/12
Single-stage pruning at 25%													
25%: 17→1	76.53	91.77	98.08	96.93	**39.76**	37.85	95.04	89.49	95.20	95.75	95.81	93.44	2/12
25%: 8→4	75.62	92.74	98.00	97.96	33.93	53.57	95.31	91.96	**95.49**	95.61	96.13	95.78	6/12
25%: 14→2	76.42	92.50	98.18	97.56	38.58	41.21	95.18	90.98	95.38	95.68	95.85	94.41	4/12
Single-stage pruning at 50%													
50%: 9→1	75.14	92.05	98.08	98.18	34.01	44.01	94.72	90.38	95.19	95.49	96.11	95.43	5/12
50%: 6→4	74.87	92.81	98.08	98.00	31.18	57.48	95.02	**92.96**	95.35	95.48	96.13	95.48	6/12
50%: 8→2	75.21	92.57	98.27	98.12	33.99	48.20	95.06	92.18	95.29	95.51	96.12	95.39	5/12
Two-stage pruning at 25%/50%													
7→5→4	75.32	92.84	98.05	98.04	32.60	56.53	95.22	92.44	95.42	95.58	96.20	95.67	**7/12**
8→6→3	75.56	92.88	98.19	98.10	33.81	52.11	95.25	92.39	95.42	95.57	96.17	95.60	6/12
9→5→3	75.80	92.85	98.04	98.07	35.32	50.41	95.25	91.67	95.42	95.55	96.23	95.69	6/12
10→4→3	76.00	92.54	98.03	98.01	35.59	48.29	95.35	91.49	95.45	95.59	96.17	95.62	6/12
10→6→2	75.91	92.46	98.10	98.22	35.73	47.58	95.23	90.70	95.33	95.62	96.27	95.59	5/12
10→8→1	75.51	91.98	98.10	98.25	34.29	44.21	94.85	90.07	95.23	95.55	96.17	95.56	5/12
12→6→1	76.04	92.26	98.00	**98.32**	36.03	44.63	95.22	88.91	95.32	95.68	96.27	95.67	4/12
14→4→1	76.16	92.32	98.10	98.11	37.90	41.43	**95.42**	89.09	95.38	95.70	96.09	95.11	4/12
Three-stage pruning at 25%/50%/75%													
7→6→4→3	75.33	92.82	98.15	98.06	32.67	57.63	95.13	92.87	95.41	95.54	96.23	95.64	**7/12**
8→5→4→3	75.61	92.83	98.09	98.08	33.82	55.66	95.31	92.21	95.47	95.66	96.20	95.78	6/12
8→6→5→1	75.69	92.52	98.19	98.10	33.38	**58.55**	94.89	91.51	95.37	95.60	96.20	95.69	6/12
10→5→3→2	76.08	92.62	98.08	98.15	35.40	49.24	95.31	90.83	95.41	95.63	96.26	95.74	6/12
10→6→3→1	76.00	92.41	98.13	98.23	35.04	49.93	95.10	89.79	95.36	95.61	96.29	95.70	4/12
12→5→2→1	76.29	92.52	98.14	98.21	36.92	46.10	95.33	89.02	95.36	95.71	96.24	95.56	4/12

Table 16: **Budget-matched evaluation for TopK archive.** Entries report mean final scores over $100,000$ empirical resampling trials under a five-full-run-equivalent budget. All score values are reported as percentages, obtained by multiplying the original values by 100. Winrate is the number of the twelve model–problem pairs on which a strategy has a higher reported score than the Sequential BoN reference; ties do not count as wins. Higher is better; bold indicates the best value in each score column and the highest winrate.

Strategy / schedule	CP Q2.5	CP Q3	CP GPT 20B	CP GPT 120B	HT Q2.5	HT Q3	HT GPT 20B	HT GPT 120B	SA Q2.5	SA Q3	SA GPT 20B	SA GPT 120B	Winrate (/12)
Sequential BoN reference, 5 full	78.46	93.59	98.63	97.55	24.34	**56.18**	91.33	89.85	95.38	96.08	**96.59**	94.28	N/A
Single-harness baseline, 5 full	77.29	91.34	97.91	97.24	31.82	47.30	91.47	91.77	95.37	95.81	95.86	94.24	3/12
Unpruned harness portfolio, 5 full	77.83	93.77	97.91	97.27	31.26	50.05	92.60	90.84	95.40	96.06	95.96	94.44	6/12
Single-stage pruning at 25%													
25%: 17→1	76.31	95.17	**98.67**	97.40	34.47	45.93	91.09	91.64	95.43	96.37	95.72	94.28	6/12
25%: 8→4	78.31	94.67	98.13	97.38	33.34	52.08	92.44	92.16	95.50	96.24	96.04	94.38	7/12
25%: 14→2	78.36	**95.19**	98.50	97.41	36.11	48.53	92.26	92.92	95.47	**96.41**	96.01	94.24	6/12
Single-stage pruning at 50%													
50%: 9→1	77.99	94.63	98.09	97.43	34.63	52.30	92.25	92.15	95.40	96.25	95.89	94.37	7/12
50%: 6→4	77.67	94.21	98.04	97.36	32.63	52.18	92.97	91.58	95.44	96.18	96.02	94.37	7/12
50%: 8→2	78.51	94.66	98.09	97.47	34.50	53.55	**93.10**	92.38	95.47	96.27	95.99	94.35	8/12
Two-stage pruning at 25%/50%													
7→5→4	78.07	94.48	98.14	97.39	32.95	53.07	92.69	91.69	95.47	96.22	96.04	94.46	7/12
8→6→3	78.71	94.71	98.22	97.48	34.07	54.37	92.96	92.18	95.49	96.29	96.06	94.41	8/12
9→5→3	78.95	94.84	98.28	97.53	34.62	53.85	92.87	92.65	95.52	96.31	96.06	94.40	8/12
10→4→3	79.21	94.94	98.32	97.48	34.82	52.34	92.67	92.99	**95.54**	96.34	96.05	94.33	8/12
10→6→2	**79.26**	94.93	98.34	97.59	35.40	54.55	92.81	93.00	95.51	96.35	96.02	94.44	**9/12**
10→8→1	78.32	94.80	98.22	97.46	34.74	52.98	92.24	92.45	95.44	96.29	95.88	**94.47**	7/12
12→6→1	78.72	94.97	98.34	97.56	35.71	52.38	92.36	93.23	95.50	96.34	95.85	94.41	**9/12**
14→4→1	78.70	95.11	98.49	97.40	**36.43**	50.16	92.42	**93.33**	95.52	96.35	95.78	94.25	7/12
Three-stage pruning at 25%/50%/75%													
7→6→4→3	78.26	94.51	98.17	97.44	33.29	53.48	93.09	91.75	95.49	96.24	96.00	94.37	7/12
8→5→4→3	78.53	94.68	98.22	97.47	33.82	53.57	92.76	92.17	95.51	96.27	96.01	94.39	8/12
8→6→5→1	78.28	94.65	98.21	97.48	34.19	53.71	92.81	91.87	95.46	96.24	95.90	94.38	7/12
10→5→3→2	79.22	94.94	98.36	97.58	35.36	53.81	92.89	93.01	**95.54**	96.34	96.01	94.38	**9/12**
10→6→3→1	78.87	94.94	98.34	**97.60**	35.40	54.69	92.89	92.65	95.48	96.33	95.94	**94.47**	**9/12**
12→5→2→1	79.25	95.06	98.39	97.59	35.88	53.13	92.58	93.28	95.51	96.35	95.87	94.39	**9/12**

Table 17: **Pairwise strategy win counts.** Entry (i, j) is the number of the 12 model–problem pairs on which strategy i has a higher reported mean final score than strategy j. Counts use the two-decimal scores in Table 1; ties count for neither strategy. Diagonal entries are omitted. The highlighted 12→5→2→1 schedule is the only strategy that wins more pairs than it loses against every other strategy.

ID	Strategy / schedule	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Sequential BoN reference, 5 full	–	9	8	4	2	4	3	2	2	2	3	2	2
2	Single-harness baseline, 5 full	3	–	0	0	0	0	0	0	0	0	0	0	0
3	Unpruned harness portfolio, 5 full	4	12	–	4	2	2	1	1	1	2	1	1	1
4	25%: 17→1	7	12	8	–	0	4	4	4	4	3	4	4	3
5	25%: 14→2	10	12	10	12	–	10	9	5	6	5	9	5	3
6	50%: 9→1	8	12	10	8	2	–	3	0	0	2	4	0	1
7	50%: 8→2	9	12	11	8	3	9	–	0	2	3	9	0	0
8	10→6→2	10	12	11	8	7	12	12	–	7	4	10	7	4
9	12→6→1	10	12	11	8	5	12	10	3	–	4	10	4	1
10	14→4→1	10	12	10	9	6	10	9	7	7	–	10	7	3
11	8→6→5→1	9	12	11	8	3	7	2	2	2	2	–	2	2
12	10→6→3→1	9	12	11	8	6	12	12	3	5	4	10	–	3
13	**12→5→2→1**	10	12	11	9	8	11	12	8	11	7	10	9	–