ALLOCATE, DON'T JUST RANK: REDUCING REDUNDANCY IN TEST-TIME SEARCH FOR SCIENTIFIC DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time search for scientific discovery reuses previously generated programs and mathematical constructions as context for further language-model generation. Yet selecting solutions solely by their individual promise can concentrate a fixed search budget on structurally similar solutions, producing redundant exploration. We formulate solution reuse as batch allocation: deciding which solutions should be explored together to advance the discovery frontier. We introduce LC-DFT, a submodular optimization framework combining distributional frontier-tail quality (DFT) with lineage coverage (LC). DFT scores the best outcomes of previous rollout bundles against historical rewards, providing a dense quality signal even when frontier improvements are rare. LC uses shared search-tree ancestry to discount redundant contributions within the selected batch. The resulting search policy applies with either a fixed or a continually updated language model. With Qwen3-8B, LC-DFT improves mean performance on mathematical construction, scheduling, and denoising tasks relative to the PUCT-based baseline, while DFT achieves the lowest mean GPU-kernel latencies. Search-tree analysis shows reduced within-batch ancestry overlap alongside later breakthroughs from solutions reused across steps. With Qwen3.8-27B, LC-DFT establishes a new state of the art in Circle-32 packing and produces high-performance GPU kernels.

1 INTRODUCTION

Language-model-guided search has begun to produce new programs, algorithms, and mathematical constructions. FunSearch couples language-model proposals with executable evaluation (Romera-Paredes et al., 2024), and AlphaEvolve extends this pattern to broad program evolution (Novikov et al., 2025). TTT-Discover additionally updates the language model on a single target problem while maintaining a buffer of previously generated solutions (Yuksekgonul et al., 2026).

Previously generated solutions can be reused as contexts for further language-model generation. We call the solutions selected for reuse *parents*. For a fixed batch budget, choosing these parents allocates the next batch of search attempts: each selected parent receives a fixed-size rollout bundle. Ranking parents by a singleton score reduces this set decision to independent choices, even though discovery depends on the best outcome of the whole batch.

A parent can generate diverse offspring, but solutions with shared ancestry often inherit similar code, constructions, or reasoning structure. Reusing several such solutions together can concentrate attempts on similar outcomes, even when each has high individual quality. FunSearch preserves diversity through separate program populations (Romera-Paredes et al., 2024), while SimpleTES excludes neighboring historical solutions from a shared prompt (Ye et al., 2026). A solution's contribution to the next batch depends on which other solutions are explored alongside it. This leads to our central question:

Under a fixed search budget, which previously generated solutions should be reused together to advance the discovery frontier?

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.

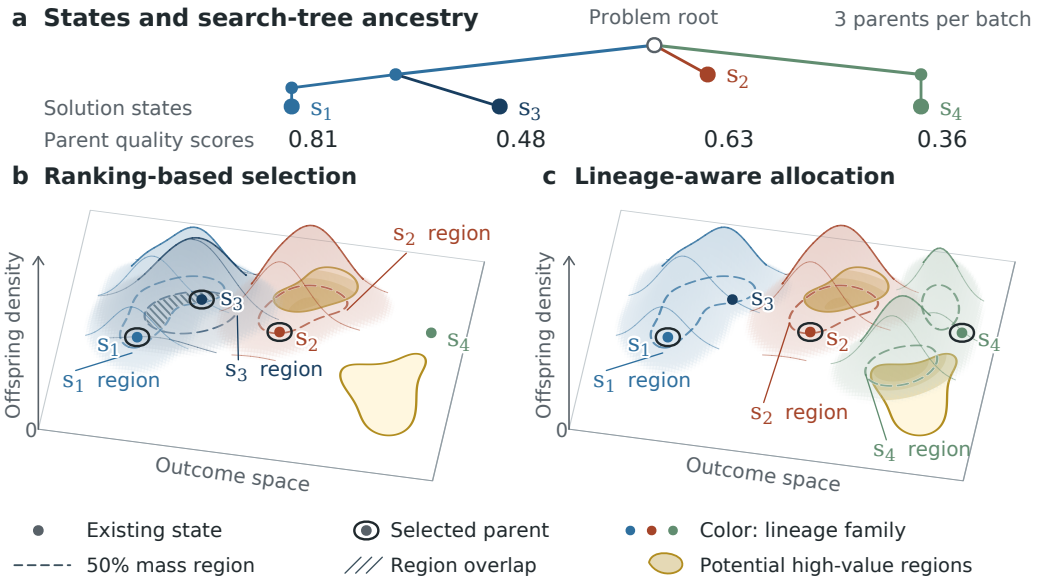


Figure 1: **Parent quality and batch allocation.** (a) Candidate states, quality scores, and ancestry. (b,c) Rings mark selected parents; surfaces show their offspring densities. Dashed contours enclose 50% probability mass, and hatching marks overlap. Replacing s_3 with s_4 reduces overlap among the selected offspring distributions while retaining s_1, s_2. Gold shading shows equal-weight batch density within potential high-value regions. All quantities are illustrative.

We formulate solution reuse as a batch allocation problem. This requires assessing both which solutions warrant further exploration and which should be explored together. *Distributional frontier-tail quality* (DFT) evaluates reuse potential from the best outcome of each previously allocated rollout bundle, rather than the reward of the existing solution. It scores these outcomes relative to historical offspring rewards, accounting for bundle size, so parents receive graded scores even when improvements beyond the current frontier are rare.

Lineage coverage (LC) uses shared ancestry in the search tree to represent structural similarity among candidate solutions. LC-DFT combines DFT quality with lineage coverage in a submodular optimization objective, reducing the marginal value of reusing solutions whose inherited structure is already well represented in the batch (Figure 1). Repeatedly expanding a promising solution across steps can remain valuable; LC addresses the separate question of which solutions should be expanded together.

With Qwen3-8B (Qwen Team, 2025), LC-DFT improves mean performance on mathematical construction, scheduling, and denoising tasks relative to the PUCT-based baseline, while DFT yields the lowest mean GPU-kernel latencies (Table 2). Search-tree analysis shows lower within-batch ancestry overlap alongside productive reuse across steps, including later breakthroughs from repeatedly explored parents (Figure 2). With Qwen3.8-27B (Qwen Team, 2026), LC-DFT establishes a new state of the art in Circle-32 packing and produces high-performance GPU kernels (Table 1).

2 From Frontier Improvement to Batch Allocation

2.1 Parent-conditioned rollout bundles

We study scientific discovery problems specified by a description d, an executable verifier, and a scalar reward. At step t, a buffer $\mathcal{H}_t$ stores previously generated states, each containing a candidate solution and its reuse metadata. Selecting a parent s combines its solution and recorded verifier value with the shared problem description and generation instructions. The language model generates thinking tokens followed by code, which the environment parses and evaluates as an offspring state.

For example, in the Erdős minimum-overlap problem, the selected parent's certificate code conditions the generation of a modified construction, whose upper bound is recomputed by the verifier. Appendix A reproduces a complete prompt.

Table 1: Best solutions found by LC-DFT with Qwen3.8-27B, alongside best-known records and literature results. Parentheses give checkpointed steps for the single run per task.

Domain	Task	Best-known result	Literature results	LC-DFT (27B)
Mathematical construction	**Erdős** ↓ *Upper bound*	**0.38085857** EinsteinArena (n.d.c)	0.380868 SimpleTES (2026)	0.38087562 (20 steps)
	AC1 ↓ *Upper bound*	**1.50274365** EinsteinArena (n.d.a)	1.50287 TTT-Discover (2026)	1.50495165 (25 steps)
	AC2 ↑ *Lower bound*	**0.96359** EinsteinArena (n.d.b)	0.962694 SimpleTES (2026)	0.94973 (25 steps)
	Circle 26 ↑ *Sum of radii*	**2.635983084919** Packomania (2026)	2.635983 AlphaEvolve V2 (2025)	2.635983084918 (20 steps)
	Circle 32 ↑ *Sum of radii*	2.939572771205 Packomania (2026)	2.939572 AlphaEvolve V2 (2025)	**2.939572771206** (20 steps)
GPU-kernel optimization	**TriMul** ↓ *H200, µs*	1064.9 (H100) GPU MODE (2026)	1020.0 (H200) 1122.0 (H100) SimpleTES (2026)	**881.7** (50 steps)
	MLA ↓ *H200, µs*	1787.5 (MI300) GPU MODE (2025)	1876.9⋆ TTT-Discover (2026)	**1191.7** (40 steps)
Heuristic programming	**AHC039** ↑ *Geometry*	**589,491**[C] AtCoder (2026a)	567,062[C] TTT-Discover (2026)	575,374[C] (50 steps)
	AHC058 ↑ *Scheduling*	**851,484,676**[C] AtCoder (2026b)	848,414,228[C] TTT-Discover (2026)	847,260,907[C] (50 steps)
Single-cell denoising	**PBMC** ↑ *Denoising score*	**0.71** SimpleTES (2026)	**0.71** TTT-Discover (2026)	0.70 (25 steps)
	Tabula ↑ *Denoising score*	**0.74** SimpleTES (2026)	0.73 TTT-Discover (2026)	0.73 (25 steps)

Published values retain their original precision; TTT-Discover and SimpleTES entries use gpt-oss-120B. C: AtCoder submission total. GPU timings use H200 unless otherwise labeled. The best-known GPU entries come from the official leaderboards. ⋆ Unmodified public programs re-evaluated on H200. Sources, model details, and broader comparisons: Appendix H.6; run budgets: Appendix H.5.

We write $R(x)$ for the higher-is-better reward of offspring x. The cold-start prior ranks states by their value $V(s)$, which induces the same ordering as R for valid generated states but may use a different numerical scale (Appendix B).

The selector chooses $S_t \subseteq \mathcal{H}_t$ with $|S_t| = K$. Each selected parent receives the same bundle budget n. For $s \in S_t$, denote its conditionally sampled offspring by $X_{s,t}^{(1)}, \ldots, X_{s,t}^{(n)}$. The maximum offspring reward in the bundle is

$$B_{s,t} = \max_{1 \le j \le n} R\Big(X_{s,t}^{(j)}\Big). \tag{1}$$

All probabilities below condition on the information available immediately before selection at step t; the conditioning is suppressed for readability. During rollout generation, the model parameters and selected parent contexts remain fixed. New offspring affect parent selection only in subsequent batches. Model parameters may also be updated between batches.

2.2 From discovery to one-step batch allocation

Let b_0 be the reward of the best-known solution at the start of search. Producing a state s with $R(s) > b_0$ is a *discovery event*, and $R(s) - b_0$ is its *discovery magnitude*. At selection step $t \in \{1, \ldots, T\}$, b_{t-1} is the best reward found before selecting parent set S_t, and b_t is the frontier after its rollout bundles are evaluated. For any parent set S considered at step t, define its *batch discovery gain* as

$$I_t(S) := \left(\max_{s \in S} B_{s,t} - b_{t-1}\right)_+, \qquad b_t = b_{t-1} + I_t(S_t). \tag{2}$$

Across T steps, these gains telescope to $b_T - b_0 = \sum_{t=1}^{T} I_t(S_t)$, the magnitude of the best discovery found.

Each batch produces new solutions and updates the buffer, changing the next allocation problem. Optimizing the full horizon would require predicting this evolving search process. We instead solve the allocation at the current step and repeat after each batch. Given the current buffer and model, the

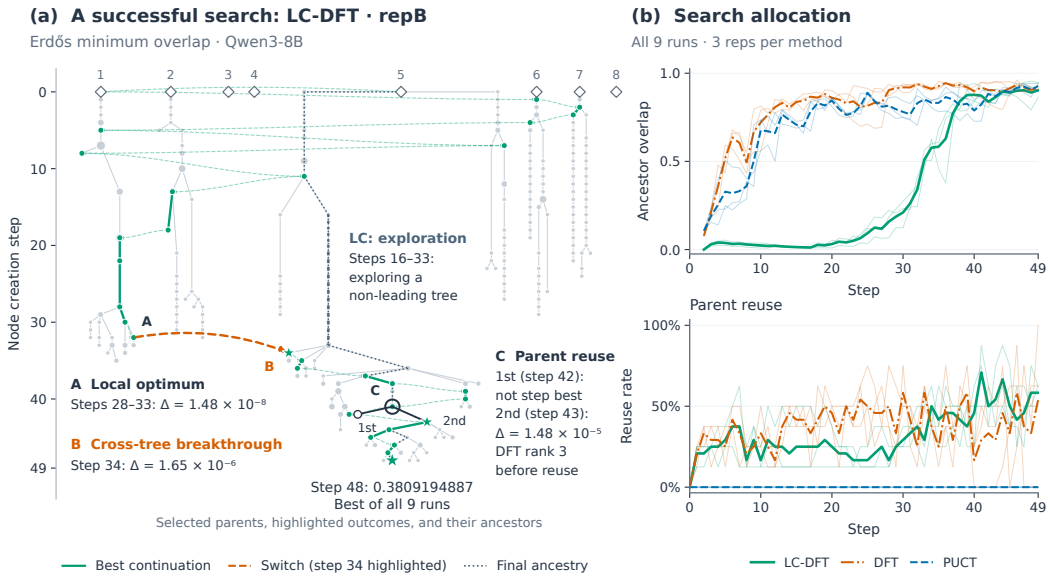


Figure 2: **Branch exploration and parent reuse on Erdős.** (a) LC-DFT repB, showing selected parents and their ancestry; other terminal leaves are omitted. A–C mark a plateau in the leading branch, a breakthrough from another tree, and a later improvement from reusing the same parent. At C, 1st/2nd identify the best offspring from that parent's first and second rollout bundles. Solid green segments follow best continuations; dashed arrows link changes of best representative, and dotted paths trace the final solution's ancestry. Gray-node area reflects how often a parent is selected. (b) Shared ancestry within batches and parent reuse across steps for all nine 8B runs. Thin lines show individual runs; thick lines show means of three runs per method. Step IDs are zero-based.

ideal one-step allocation is

$$S_t^\star \in \arg\max_{\substack{S\subseteq\mathcal{H}_t\\|S|=K}} \mathbb{E}[I_t(S)]. \tag{3}$$

For any reward threshold τ, write $U_t(S;\tau) := \Pr[\max_{s\in S} B_{s,t} > \tau]$ for the probability that the selected batch exceeds τ. For any realized batch, its discovery gain equals the length of the reward interval between the current frontier and the batch maximum:

$$\left(\max_{s\in S} B_{s,t} - b_{t-1}\right)_+ = \int_{b_{t-1}}^{\infty} \mathbf{1}\left\{\max_{s\in S} B_{s,t} > \tau\right\} d\tau.$$

Every threshold below the realized batch maximum contributes to this interval; every higher threshold contributes zero. Taking expectations gives the tail representation

$$\mathbb{E}[I_t(S)] = \int_{b_{t-1}}^{\infty} U_t(S;\tau)\, d\tau. \tag{4}$$

Equation 3 therefore selects the parent set with the largest area under its threshold-hit probability curve above the current frontier. The value $U_t(S; b_{t-1})$ gives only the probability of any strict advance; integrating over higher thresholds additionally accounts for how far the batch is expected to advance.

2.3 Set structure of the ideal utility

For a fixed threshold τ, $U_t(S;\tau)$ is the probability of a union of parent-level threshold-hit events.

Lemma 1 (Submodularity of threshold batch utility)**.** *For fixed t and τ, with $U_t(\varnothing;\tau) = 0$, the set function $U_t(\cdot;\tau)$ is normalized, monotone, and submodular.*

For $s \notin S$, the marginal contribution is

$$U_t(S\cup\{s\};\tau) - U_t(S;\tau) = \Pr\left[B_{s,t} > \tau,\ \max_{u\in S} B_{u,t} \le \tau\right]. \tag{5}$$

The added parent contributes only when it exceeds the threshold and every selected parent fails to do so. Equation 4 is a nonnegative integral of the threshold utilities in Lemma 1, so the ideal expected discovery gain $\mathbb{E}[I_t(S)]$ has the same properties. The set structure thus arises from the discovery objective itself: selecting parents solely by their singleton hit probabilities discards the overlap among their threshold-hit events. Appendix C gives the full event-algebra derivation.

2.4 Distributional frontier-tail surrogate

Although the ideal objective has the desired set structure, evaluating Equation 4 before allocation requires the unobserved upper tail of every candidate's future bundle outcomes. Historical bundle maxima do not directly populate the region above the current frontier b_{t-1}. We therefore replace equal weighting per unit reward above the frontier with a distribution of bundle-scale thresholds constructed from historical rewards.

Let Z_t be a reward sampled from the offspring rewards recorded before step t, with $F_t(y) = \Pr(Z_t < y)$. Draw n independent rewards from this distribution and take their maximum:

$$\widetilde{B}_t = \max_{1 \le j \le n} Z_t^{(j)}, \qquad \Pr(\widetilde{B}_t < y) = F_t(y)^n. \tag{6}$$

The maximum is below y exactly when all n draws are below y. Thus $\widetilde{B}_t$ is a historical comparison maximum on the same best-of-n scale as an actual parent bundle. Let $H_{t,n}$ denote its distribution, and draw $\widetilde{B}_t$ independently of the future parent bundles. The surrogate changes the threshold weighting from

$$\underbrace{\int_{b_{t-1}}^{\infty} U_t(S;\tau)\, d\tau}_{\text{ideal expected discovery gain}} \quad \longrightarrow \quad \underbrace{\overline{U}_t(S) := \int U_t(S;\tau)\, dH_{t,n}(\tau)}_{\text{historical bundle-threshold surrogate}} = \Pr\left[\max_{s \in S} B_{s,t} > \widetilde{B}_t\right],\ \widetilde{B}_t \sim H_{t,n}. \tag{7}$$

The left side weights all reward thresholds above the current frontier by their width. The right side instead averages threshold-hit utility over the historical bundle scale, providing graded comparisons even when no historical bundle exceeds b_{t-1}. By Lemma 1, this nonnegative average is also normalized, monotone, and submodular.

For a singleton parent, the resulting population distributional frontier-tail (DFT) quality is

$$q_t(s) := \overline{U}_t(\{s\}) = \Pr(B_{s,t} > \widetilde{B}_t) = \mathbb{E}_{B_{s,t}}[F_t(B_{s,t})^n]\,. \tag{8}$$

Reducing the distributional set problem to independent singleton selection gives the modular surrogate

$$\overline{U}_t^{\mathrm{mod}}(S) := \sum_{s \in S} q_t(s). \tag{9}$$

For $q_t(s) > 0$, define the distributional overlap

$$o_t(s,S) := \Pr\left[\max_{u \in S} B_{u,t} > \widetilde{B}_t \,\middle|\, B_{s,t} > \widetilde{B}_t\right], \tag{10}$$

and set $o_t(s,S) = 0$ when $q_t(s) = 0$. The exact marginal of the distributional surrogate is then

$$\overline{U}_t(S \cup \{s\}) - \overline{U}_t(S) = q_t(s)\big[1 - o_t(s,S)\big]. \tag{11}$$

Taking the top K parents by $q_t(s)$ maximizes Equation 9 but ignores the set-dependent overlap factor. Section 3.1 constructs DFT quality from bundle outcomes previously observed for each parent. The search tree supplies a structural proxy for $o_t(s,S)$: parents with long shared lineage prefixes carry related intermediate solutions into generation, so their offspring can revisit similar outcomes and failure modes. LC uses this ancestry to discount redundant reuse of structurally similar solutions within the batch.

3 Frontier-Tail Parent Selection

3.1 Distributional frontier-tail quality

Before selection at step t, let $\mathcal{R}_t$ be the multiset of finite offspring rewards from eligible historical parent bundles, and let $\widehat{F}_t$ be its regularized empirical percentile (Appendix B).

For eligible historical bundle i, let B_i be its largest finite offspring reward. Its selection-time utility uses the best-of-n transform in Equation 8:

$$u_{i,t} = \widehat{F}_t(B_i)^n. \tag{12}$$

A completed bundle with no finite offspring reward receives $u_{i,t} = 0$. At each selection step, all historical utilities are recomputed against the current reward distribution.

Let $\mathcal{E}_t(s)$ contain historical bundles whose recorded parent state is exactly s. For cold start, let $\pi_t(s)$ be the descending midrank percentile of the higher-is-better state value $V(s)$ among selectable states in $\mathcal{H}_t$. The DFT reuse score is

$$q_t^{\mathrm{DFT}}(s) = \begin{cases} \dfrac{1}{|\mathcal{E}_t(s)|} \displaystyle\sum_{i \in \mathcal{E}_t(s)} u_{i,t}, & |\mathcal{R}_t| \geq n \text{ and } |\mathcal{E}_t(s)| > 0, \\ c_{\mathrm{prior}}\, \pi_t(s), & \text{otherwise}, \end{cases} \tag{13}$$

with $c_{\mathrm{prior}} = 0.5$. This cold-start scale lets strong untried states compete with parents whose observed bundles have ordinary utility. Once a parent has been expanded, its mean transformed bundle maximum determines subsequent reuse. DFT uses outcomes from that parent alone; lineage enters through batch allocation. Appendix B gives the prior's rank convention, its scale, and the ordering of tied scores.

3.2 Search-tree representation

A discovery problem can begin from multiple seed states. For allocation, we represent those seeds by a shared abstract problem root $r_\star$. For candidate s, let $P^\star(s)$ contain $r_\star$ followed by the generated ancestors of s, excluding s itself. Set $d(r_\star) = 0$; generated descendants have depths $1, 2, \ldots$. If d_s is the generation depth of s and $d(v)$ the depth of feature v, define

$$\widetilde{\phi}_s(v) = \frac{1}{1 + d_s - d(v)}, \qquad \phi_s(v) = \frac{\widetilde{\phi}_s(v)}{\sum_{v' \in P^\star(s)} \widetilde{\phi}_s(v')}. \tag{14}$$

We set $\phi_s(v) = 0$ outside $P^\star(s)$. The feature mass is nonnegative and sums to one. Recent ancestors receive larger weight because they represent more closely shared search history. An ancestor remains part of a state's path even after leaving the selectable buffer. Seed states and their first-generation offspring have only the shared-root feature; deeper states also carry features of their generated ancestors. Coverage thus distinguishes branches developed during search rather than treating distinct initial seeds as already explored branches.

3.3 Lineage-coverage allocation

LC accepts any singleton quality $q_t(s) \in [0, 1]$; LC-DFT sets $q_t(s) = q_t^{\mathrm{DFT}}(s)$. Summing over ancestor features on the paths of buffered states, we use the quality-weighted max-coverage function

$$\mathcal{C}_t(S) = \sum_v \max_{s \in S} q_t(s) \phi_s(v), \tag{15}$$

where the maximum over an empty set is zero. Because the lineage features sum to one, this function is singleton-normalized: $\mathcal{C}_t(\{s\}) = q_t(s)$. For each ancestor feature, the maximum retains only the strongest selected contribution. Adding another parent from an already represented branch therefore produces a smaller coverage gain. Appendix D gives the corresponding greedy update.

For selected set S, write $\Delta_{\mathcal{C}_t}(s \mid S) = \mathcal{C}_t(S \cup \{s\}) - \mathcal{C}_t(S)$. Submodularity and singleton normalization imply $0 \leq \Delta_{\mathcal{C}_t}(s \mid S) \leq q_t(s)$. For $q_t(s) > 0$, define the structural saturation

$$\eta_t(s, S) = 1 - \frac{\Delta_{\mathcal{C}_t}(s \mid S)}{q_t(s)} \tag{16}$$

as the share of singleton coverage already absorbed by S. We set $\eta_t(s, S) = 0$ when $q_t(s) = 0$.

The factorization in Equation 11 motivates discounting a parent's quality when the selected set already covers similar search opportunities. DFT supplies the singleton score, while η_t measures how much

Algorithm 1 Lineage-Coverage Parent Allocation

Require: eligible buffer $\mathcal{H}_t$ with $|\mathcal{H}_t| \geq K$, qualities q_t, coverage $\mathcal{C}_t$, batch size K, correction strength λ_t

```
S ← ∅
for i = 1, ..., K do
    for all s ∈ H_t \ S do
        δ_i(s) ← (1 − λ_t) q_t(s) + λ_t Δ_{C_t}(s | S)
    end for
    s* ← arg max_{s ∈ H_t \ S} δ_i(s)
    S ← S ∪ {s*}
end for
return S
```

of its search-tree coverage is already represented. LC applies this discount with strength $\lambda_t \in [0, 1]$:

$$\begin{aligned}\Delta_t(s \mid S) &= q_t(s)\big[1 - \lambda_t \eta_t(s, S)\big] \\ &= (1 - \lambda_t) q_t(s) + \lambda_t \Delta_{\mathcal{C}_t}(s \mid S). \end{aligned} \tag{17}$$

The parameter λ_t controls how strongly search-tree saturation changes the current batch. At zero, LC recovers top-K selection under q_t; at one, it uses the full structural coverage marginal. The corresponding set objective is

$$J_t(S) = (1 - \lambda_t) \sum_{s \in S} q_t(s) + \lambda_t \mathcal{C}_t(S). \tag{18}$$

With qualities and $\lambda_t \in [0, 1]$ fixed within the batch, J_t is monotone submodular, so greedy allocation attains the standard $(1 - 1/e)$ guarantee under $|S| \leq K$ (Nemhauser et al., 1978). Appendix E states and proves Theorem 2.

Singleton normalization makes the first selected parent independent of λ_t; structural coverage changes the allocation from the second slot onward. We use a predetermined sigmoid schedule for the allocation weight that decreases over the run, emphasizing tree coverage early and DFT quality after direct parent-reuse evidence has accumulated. This weight changes only greedy allocation, not DFT quality. LC scores and allocates over the full eligible buffer. Appendix F gives its exact definition.

4 Experiments

4.1 Experimental Setup

We evaluate ten search tasks: five mathematical construction problems, two GPU-kernel optimization problems, two AtCoder heuristic-programming problems, and single-cell RNA-seq denoising. We use Qwen3-8B to compare parent-selection methods and Qwen3.8-27B to obtain the strongest verified LC-DFT solutions. For denoising, search uses Pancreas, and selected programs are evaluated on held-out PBMC and Tabula data.

The 8B comparison includes three selectors. PUCT follows the TTT-Discover reuse rule and provides the baseline. DFT selects the top K parents by distributional frontier-tail score. LC-DFT uses the same quality score and hyperparameters but selects parents jointly through lineage coverage. The DFT–LC-DFT comparison tests joint allocation with the quality estimator held fixed. Each method generates its own search and training trajectory.

Our 8B runs use 50 steps, selecting $K = 8$ parents with $n = 64$ rollouts each, for 25,600 attempts. Table 2 reports our mean run-best scores over three runs, except circle packing (one run); reported TTT-Discover baselines are identified in the table. For 27B, we report the best verified solution from one run per task, with the consumed budget beside each score in Table 1. Final best scores retain each task's native metric and direction, while best-so-far curves describe progress throughout search. We follow TTT-Discover's task configurations and adapt training and generation budgets to available resources; Appendices H.2–H.5 give the detailed settings and evaluation protocols.

Table 2: Qwen3-8B method comparison. Our results are means over three runs, except circle packing (one run).

Task	TTT-Discover / PUCT	DFT	LC-DFT
Erdős ↓ *Upper bound*	0.38096841*, †	0.38094456	**0.38092384**
AC1 ↓ *Upper bound*	1.50525	1.50488485	**1.50475789**
AC2 ↑ *Lower bound*	0.9472	0.94699	**0.94885**
Circle 26 ↑ *Sum of radii*	2.635983084907*	2.635983084915	**2.635983084916**
Circle 32 ↑ *Sum of radii*	2.939572771200*	2.939572771200	**2.939572771203**
TriMul ↓ *H200, µs*	6554.2*	**5367.0**	6667.8
MLA ↓ *H200, µs*	3543.5*	**3524.8**	3573.4
AHC058 ↑ *Scheduling*	761,532,973*	758,789,165	**766,321,006**
Denoising ↑ *PBMC*	0.69*	0.69	**0.70**
Denoising ↑ *Tabula*	0.48*, ‡	**0.72**	**0.72**

∗ Our reproduction of PUCT; unmarked baselines are reported by Yuksekgonul et al. (2026). Generation budgets are listed in Appendix Table 6; repetition counts and score aggregation appear in Appendix H.3. AHC uses public-case means multiplied by 150. † The reported Erdős bound is 0.380932 (program estimate); our scores use recomputed C5. ‡ One Tabula timeout contributes zero. Bold indicates the best result among comparable entries in each row, including ties.

4.2 Main Results

With Qwen3-8B, LC-DFT obtains better mean bounds than the TTT-Discover/PUCT baseline on Erdős, AC1, and AC2, and higher mean scores on AHC058 and both denoising evaluations (Table 2). Comparing it with DFT shows where joint allocation adds value: LC-DFT achieves better mean bounds on the same three mathematical tasks and a higher mean AHC058 score, with smaller gains in the single-run circle-packing results. It also achieves the best PBMC mean and ties with DFT on Tabula at the reported precision. On GPU-kernel optimization, DFT yields the lowest mean run-best latency on both TriMul and MLA, outperforming PUCT and LC-DFT. Reuse quality and batch allocation thus make different contributions across tasks. Appendix Figure 3 shows progress throughout the search budget.

With Qwen3.8-27B, LC-DFT establishes a new state of the art in Circle-32 packing and produces high-performance GPU kernels (Table 1). Its Circle-32 construction reaches a sum of radii of 2.939572771206, exceeding Packomania's published record of 2.939572771205 (Specht, 2026). On MLA, latency falls from 1876.9 to $1191.7\,\mu$s, a 36.5% reduction relative to the fastest released TTT-Discover program re-evaluated under the same H200 protocol. The TriMul kernel reaches $881.7\,\mu$s, the lowest reported H200 time among the listed results.

4.3 Search Dynamics

LC-DFT reduces shared ancestry within batches while retaining productive reuse across steps. Across 54 Qwen3-8B runs on eight tasks, its mean ancestor overlap is lower than DFT's on every task and lower than PUCT's on all six tasks with recorded PUCT trees (Appendix Table 10). This structural change accompanies the mean-score gains on mathematical construction and AHC058 in Table 2, whereas quality-only DFT remains stronger on GPU kernels: LC consistently reduces shared ancestry, while the benefit of joint allocation varies across tasks. On Erdős, mean overlap falls from 0.807 for DFT to 0.329 for LC-DFT, while previously selected parents still receive 32.5% of LC-DFT selections, compared with 37.6% for DFT and zero for PUCT. Across the six tasks with three runs each for DFT and LC-DFT, all 18 runs of each method produce a new global best from a previously selected parent. Lower within-batch ancestry overlap therefore coexists with repeated investment that yields new discoveries.

Figure 2 illustrates both routes to later breakthroughs in Erdős LC-DFT repB, the strongest of the nine Qwen3-8B runs on this task. While the leading branch nearly stalls during steps 28–33, continued allocation to another initial tree yields a breakthrough at step 34, lowering the global bound by 1.65×10^{-6}. Later, a parent's first bundle at step 42 improves its solution without producing that step's global best; its resulting DFT score ranks third. Reusing that parent at step 43 lowers the global bound by 1.48×10^{-5}, and the winning offspring lies on the final solution's ancestor chain. Across the three LC-DFT runs, low ancestor overlap early in search gives way to greater concentration later,

consistent with the decreasing allocation weight. Together, these observations show how exploration across branches and sustained reuse of promising solutions can contribute within the same search.

5 Related Work

Parent selection in test-time discovery. TTT-Discover uses a PUCT-based reuse rule (Yuksekgonul et al., 2026). CCTS learns contrastive parent weights from a semantic concept hierarchy extracted from programs (Leleu et al., 2026), while ShinkaEvolve balances exploration and exploitation in parent sampling and rejects low-novelty proposals (Lange et al., 2026). SimpleTES selects several historical solutions as context for one prompt using graph-based PUCT, excluding one-hop neighbors to reduce redundancy (Ye et al., 2026). All candidates in its local sampling batch share that prompt. LC-DFT instead assigns a separate rollout bundle to each selected parent and jointly selects the parent set, accounting for structural similarity among the solutions explored together. Appendix K discusses broader test-time learning and search approaches.

Batch Bayesian optimization. Batch Bayesian optimization models interactions among simultaneously evaluated points (Azimi et al., 2010). Set-valued acquisitions optimize best-of-batch outcomes under a joint surrogate posterior; Wilson et al. (2018) analyze their greedy construction and differentiable relaxations. Our current-frontier utility $U_t(S; b_{t-1})$ and expected discovery gain $\mathbb{E}[I_t(S)]$ have the batch probability-of-improvement and expected-improvement forms, respectively. Here, however, each parent receives n language-model rollouts rather than one point evaluation. DFT scores observed bundle outcomes rather than fitting a joint predictive posterior, and lineage coverage represents structural similarity through genealogy rather than an input-space or posterior kernel.

Genealogical diversity and set-aware allocation. Search history informs both similarity and allocation: genealogical distance provides a domain-agnostic diversity measure (Gabor & Belzner, 2017), while Monte Carlo Elites uses offspring survival to allocate exploration among elites or archive cells (Sfikas et al., 2021). For joint selection, submodular objectives capture diminishing returns (Nemhauser et al., 1978; Krause & Golovin, 2014); curriculum-guided hindsight experience replay combines modular proximity and submodular diversity under a scheduled tradeoff (Fang et al., 2019). LC-DFT brings these ideas to parent reuse, using shared search-tree ancestry to discount the contribution of solutions whose structure is already represented in the selected batch.

6 Limitations

DFT relies on bundles previously generated from the same parent, leaving untried states to be ranked by the state-value prior. Its score evaluates outcomes relative to historical offspring rewards rather than the magnitude of improvement beyond the current frontier. It can therefore favor parents with consistently strong bundles over those with lower typical quality but rarer, larger breakthroughs. When model parameters change during search, older bundles can also become less representative of future outcomes.

Shared ancestry measures structural similarity: nearby branches can yield different proposals, while distant branches can converge to similar algorithms. LC also optimizes one batch at a time, without explicitly valuing how the offspring generated now will create new search opportunities in later steps.

7 Conclusion

LC-DFT formulates the reuse of previously generated solutions as a submodular batch-allocation problem. DFT evaluates which solutions warrant further attempts, while LC accounts for shared ancestry when deciding which solutions to expand together. The Qwen3-8B mean results favor LC-DFT on mathematical construction and AHC058, while DFT yields the lowest mean GPU-kernel latencies. With Qwen3.8-27B, LC-DFT establishes a new state of the art in Circle-32 packing and produces high-performance GPU kernels. Across the analyzed search trees, LC-DFT reduces shared ancestry within batches while retaining parent reuse that produces new best solutions. The resulting distinction is central to allocation: repeated investment in a promising solution can be productive even when expanding many similar solutions together is redundant.

AI USE STATEMENT

We used generative AI tools for literature search, manuscript drafting and language editing, checking mathematical derivations and proofs, and writing implementation code. These tools also supported discussions of the method and experimental design. The authors take responsibility for the final manuscript, code, and reported results.

REPRODUCIBILITY STATEMENT

Appendix B gives the DFT estimator details. Appendices D–F provide the coverage update, proof, and weight schedule. Appendix H.1 specifies the tasks and reward mappings; Appendices H.2–H.4 give computational resources, training settings, run budgets, and evaluation procedures. Appendix A includes a complete parent-conditioned prompt.

REFERENCES

AtCoder. AtCoder Heuristic Contest 039: Submission 79052825. Official submission record, 2026a. URL https://atcoder.jp/contests/ahc039/submissions/79052825. Score 589491; reference updated September 25, 2026.

AtCoder. AtCoder Heuristic Contest 058: Submission 79233573. Official submission record, 2026b. URL https://atcoder.jp/contests/ahc058/submissions/79233573. Score 851484676; reference updated September 25, 2026.

Javad Azimi, Alan Fern, and Xiaoli Z. Fern. Batch bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, volume 23, 2010.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient monte-carlo bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21524–21538, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf.

Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 335–336. ACM, 1998.

EinsteinArena. First autocorrelation inequality (upper bound): Leaderboard. Online leaderboard, n.d.a. URL https://einsteinarena.com/problems/first-autocorrelation-inequality. Accessed September 22, 2026.

EinsteinArena. Second autocorrelation inequality (lower bound): Leaderboard. Online leaderboard, n.d.b. URL https://einsteinarena.com/problems/second-autocorrelation-inequality. Accessed September 22, 2026.

EinsteinArena. Erdős minimum overlap (upper bound): Leaderboard. Online leaderboard, n.d.c. URL https://einsteinarena.com/problems/erdos-min-overlap. Accessed September 22, 2026.

Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Thomas Gabor and Lenz Belzner. Genealogical distance as a diversity estimate in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1572–1577. Association for Computing Machinery, 2017. doi: 10.1145/3067695.3082529.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025. URL https://arxiv.org/abs/2511.02864.

Javier Gonzalez, Zhenwen Dai, Philipp Hennig, and Neil D. Lawrence. Batch bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pp. 648–657, 2016.

Halil Alperen Gozeten, Muhammed Emrullah Ildiz, Xuechen Zhang, Mahdi Soltanolkotabi, Marco Mondelli, and Samet Oymak. Test-time training provably improves transformers as in-context learners. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 20266–20295, 2025.

GPU MODE. AMD MLA Decode leaderboard. Online programming competition, 2025. URL https://www.gpumode.com/leaderboard/463. Accessed September 25, 2026; MI300 first-place submission 31138, amd-mla-submission.py.

GPU MODE. TriMul leaderboard. Online programming competition, 2026. URL https://www.gpumode.com/leaderboard/496. Accessed September 25, 2026; H100 first-place submission 781368.

Akshat Gupta, Jermaine Lei, Alexander Lu, Gopala Anumanchipalli, and Leshem Choshen. Automated discovery has no universally superior harness. *arXiv preprint arXiv:2607.18235*, 2026.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.507.

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

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

Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli (eds.), *Tractability: Practical Approaches to Hard Problems*, pp. 71–104. Cambridge University Press, 2014.

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

Timothee Leleu, Sudeera Gunathilaka, Federico Ghimenti, and Surya Ganguli. Contrastive concept-tree search for LLM-assisted algorithm discovery. *arXiv preprint arXiv:2602.03132*, 2026.

Pan Li. Dictionaries, not darwin: Set-level selection beats LLM evolution in scientific equation discovery. *arXiv preprint arXiv:2607.04108*, 2026.

Anji Liu, Jianshu Chen, Mingze Yu, Yu Zhai, Xuewen Zhou, and Ji Liu. Watch the unobserved: A simple approach to parallelizing monte carlo tree search. In *International Conference on Learning Representations*, 2020.

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, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. EvoX: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026. URL https://arxiv.org/abs/2602.23413v1.

Clara Meister, Martina Forster, and Ryan Cotterell. Determinantal beam search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 6551–6562. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.512.

Elvis Nava, Mojmir Mutny, and Andreas Krause. Diversified sampling for batched bayesian optimization with determinantal point processes. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 7031–7054, 2022.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

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. *arXiv preprint arXiv:2506.13131*, 2025.

Rafael Oliveira, Louis C. Tiao, and Fabio T. Ramos. Batch bayesian optimisation via density-ratio estimation with guarantees. In *Advances in Neural Information Processing Systems*, volume 35, pp. 29816–29829, 2022. doi: 10.52202/068431-2162.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.8-27B. Official model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B. Accessed September 26, 2026.

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.

Sakana AI. Sakana AI agent wins AtCoder Heuristic Contest (first AI to place 1st), 2026. URL https://sakana.ai/ahc058/. Official report, January 5, 2026.

Konstantinos Sfikas, Antonios Liapis, and Georgios N. Yannakakis. Monte carlo elites: Quality-diversity selection as a multi-armed bandit problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 180–188. Association for Computing Machinery, 2021. doi: 10.1145/3449639.3459321.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Eckard Specht. The best known packings of variable-sized circles in a square with maximized sum of radii. Packomania, 2026. URL https://www.packomania.com/csqv/csqv.html. Updated September 21, 2026; accessed September 22, 2026.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9229–9248, 2020.

Haorui Wang, Parshin Shojaee, Kazem Meidani, Kunyang Sun, José Miguel Hernández-Lobato, Teresa Head-Gordon, Jiajun He, Chandan K. Reddy, Chao Zhang, and Yuanqi Du. Towards diverse scientific hypothesis search with large language models. *arXiv preprint arXiv:2606.10587*, 2026.

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. *arXiv preprint arXiv:2511.23473*, 2025.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1954–1963, 2015.

Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*, 2026. URL https://arxiv.org/abs/2602.04837.

James T. Wilson, Frank Hutter, and Marc Peter Deisenroth. Maximizing acquisition functions for bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Haotian Ye, Haowei Lin, Jingyi Tang, Yizhen Luo, Rahul Thapa, Caiyin Yang, Chang Su, Rui Yang, Ruihua Liu, Rundao Li, Zeyu Li, Pengwei Sun, Chong Gao, Dachao Ding, Guangrong He, Miaolei Zhang, Lina Sun, Wenyang Wang, Yuchen Zhong, Zhuohao Shen, Puheng Li, Pan Lu, Bianxiao Cui, Di He, Jianzhu Ma, Junfeng Li, Hexi Baoyin, Yejin Choi, Stefano Ermon, Xiaowen Chu, Tongyang Li, Yuzhi Xu, and James Zou. Structured scaling of AI discovery across diverse scientific domains, 2026. URL https://arxiv.org/abs/2604.19341v2. Version 2, July 27, 2026.

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. *arXiv preprint arXiv:2601.16175*, 2026. URL https://arxiv.org/abs/2601.16175v2.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 131459–131483, 2025. doi: 10.52202/085713-4376. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/be690ea16f005c174f6c4102a5970e67-Abstract-Conference.html.

A Complete Parent-Conditioned Prompt

Listing 1 gives a complete Erdős user prompt for one buffered parent state. There is no separate system message. The shared prompt specifies the step-function construction, validity conditions, evaluator, resource limits, and output format. Its parent-specific portion supplies the selected certificate's accepted code, score history, previous evaluator output, and access to the construction through `initial_h_values`. The language model generates reasoning followed by Python code that constructs or modifies the certificate. Executing the code produces an offspring state; the verifier recomputes its upper bound, which is converted to a higher-is-better reward.

Listing 1: Complete parent-conditioned user prompt for one Erdős state.

```
You are an expert in harmonic analysis, numerical optimization, and mathematical discovery.
Your task is to find an improved upper bound for the Erdos minimum overlap problem constant C5.


## Problem

Find a step function h: [0, 2] -> [0, 1] that **minimizes** the overlap integral:

$$C_5 = \max_k \int h(x)(1 - h(x+k)) dx$$

**Constraints**:
1. h(x) in [0, 1] for all x
2. integral h(x) dx = 1

**Discretization**: Represent h as n_points samples over [0, 2].
With dx = 2.0 / n_points:
- 0 <= h[i] <= 1 for all i
- sum(h) * dx = 1 (equivalently: sum(h) == n_points / 2 exactly)

The evaluation computes: C5 = max(np.correlate(h, 1-h, mode="full") * dx)

Smaller sequences with less than 1k samples are preferred - they are faster to optimize and
    evaluate.

**Lower C5 values are better** - they provide tighter upper bounds on the Erdos constant.

## Budget & Resources
- **Time budget**: 1000s for your code to run
- **CPUs**: 2 available

## Rules
- Define `run(seed=42, budget_s=1000, **kwargs)` that returns `(h_values, c5_bound, n_points)`
- Use scipy, numpy, cvxpy[CBC,CVXOPT,GLOP,GLPK,GUROBI,MOSEK,PDLP,SCIP,XPRESS,ECOS], math
- Make all helper functions top level, no closures or lambdas
- No filesystem or network IO
- `evaluate_erdos_solution()` and `initial_h_values` (an initial construction, if available)
    are pre-imported
- Your function must complete within budget_s seconds and return the best solution found

**Lower is better**. Current record: C5 <= 0.38092. Our goal is to find a construction that
    shows C5 <= 0.38080.

You are iteratively optimizing C5 bound.
Here is the last code we ran:
def run(seed=42, budget_s=1000, **kwargs):
    return initial_h_values, 0.38112, len(initial_h_values)

Here is the C5 bound before and after running the code above (lower is better): 0.382050 ->
    0.381120
Target: 0.3808. Current gap: 0.000320. Further improvements will also be generously rewarded.

--- Previous Program Output ---
candidate satisfied integral constraint exactly
--- End Output ---

You may want to start your search from the current construction, which you can access through
    the `initial_h_values` global variable (n=4 samples).
You are encouraged to explore solutions that use other starting points to prevent getting
    stuck in a local optimum.

Reason about how you could further improve this construction.
Ideally, try to do something different than the above algorithm. Could be using different
    algorithmic ideas, adjusting your heuristics, adjusting / sweeping your hyperparemeters,
    etc.
Unless you make a meaningful improvement, you will not be rewarded.
```

B DFT Estimator Details

Eligible bundle history. DFT uses bundles finalized before step t, each allocated n rollout attempts. Bundles of a different size are excluded. If some attempts encounter parsing or system failures, completed evaluations unaffected by those failures still contribute to the history; the exponent remains the allocated budget n. A bundle with no such completed evaluation is excluded. Completed attempts rejected for incorrect or invalid outputs count toward the bundle budget but contribute no reward to $\mathcal{R}_t$. A retained bundle with no finite offspring reward receives utility zero. All finite rewards from retained evaluations enter the pooled history. The rank scale becomes available once $|\mathcal{R}_t| \geq n$; our experiments use $n = 64$.

Selection-time transformation. With $N_t = |\mathcal{R}_t|$, the regularized empirical percentile is

$$\widehat{F}_t(y) = \frac{|\{z \in \mathcal{R}_t : z < y\}| + \frac{1}{2}|\{z \in \mathcal{R}_t : z = y\}|}{N_t + 1}. \tag{19}$$

At each selection step, historical bundle utilities are recomputed using the current pooled rewards. For a bundle with finite outcomes, its maximum B_i enters Equation 12, while all its finite offspring rewards contribute to $\mathcal{R}_t$. Rewards equal to B_i contribute half weight to the numerator of $\widehat{F}_t(B_i)$ before taking the n-th power. Thus the bundle being scored also contributes to the historical distribution against which it is compared.

State-value prior. Let N_t^V be the number of selectable buffered states with finite higher-is-better values. Sort those states by $V(s)$ in descending order and let $r_t(s)$ be the corresponding midrank, with the best rank equal to one. Then

$$\pi_t(s) = \frac{N_t^V - r_t(s) + 1}{N_t^V}. \tag{20}$$

Tied states share their average rank. A state with a non-finite value receives prior zero. The prior is computed over the full selectable buffer.

For a valid generated state, $V(s)$ is the native verifier score with its sign chosen so that larger is better. The reward $R(s)$ may additionally apply a task-specific monotone rescaling, such as an inverse runtime or a fixed score divisor. The two quantities agree in ordering, not necessarily in numerical value. Initial states use the task-provided seed value and the prior until they have been expanded and the historical reward scale is available. Table 5 gives the task-specific mappings.

The cold-start branch multiplies $\pi_t(s)$ by $c_{\text{prior}} = 0.5$. For n independent draws from a continuous distribution F_t, the transformed maximum $F_t(B)^n$ is uniform on $[0, 1]$ and has mean 0.5. This motivates the prior's scale; the regularized finite-history scores follow Equation 19.

Exact evidence and ordering. Observed reuse uses only bundles generated from the same parent state. DFT orders states by decreasing $q_t^{\text{DFT}}(s)$, then by decreasing $\pi_t(s)$, finite $V(s)$, and generation depth. Remaining ties follow current buffer order, then state identifier. Direct DFT takes the first K states. LC uses this ordering to break equal allocation marginals.

C Event Derivation of the Batch Marginal

At a fixed step t, all candidate bundle outcomes are defined under the joint distribution determined by the current buffer and model. We take $I_t(\varnothing) = 0$ and assume $\mathbb{E}[I_t(S)] < \infty$ for the finite parent sets considered here. For the theoretical utility, a bundle with no finite offspring reward has $B_{s,t} = -\infty$, contributing no discovery gain or hit at a finite threshold.

Fix a deterministic threshold τ and define the strict hit event

$$A_{s,t}(\tau) = \{B_{s,t} > \tau\}. \tag{21}$$

The fixed-threshold utility is the probability of a union:

$$U_t(S; \tau) = \Pr\left[\bigcup_{u \in S} A_{u,t}(\tau)\right]. \tag{22}$$

For $s \notin S$, the event newly added by s is its hit event intersected with the complement of every hit event already in the batch. Hence

$$U_t(S \cup \{s\}; \tau) - U_t(S; \tau) = \Pr\left[A_{s,t}(\tau) \cap \left(\bigcup_{u \in S} A_{u,t}(\tau)\right)^c\right] = \Pr\left[A_{s,t}(\tau) \cap \bigcap_{u \in S} A_{u,t}(\tau)^c\right]. \tag{23}$$

Proof of Lemma 1. For $A \subseteq S$ and $s \notin S$,

$$A_{s,t}(\tau) \cap \bigcap_{u \in S} A_{u,t}(\tau)^c \subseteq A_{s,t}(\tau) \cap \bigcap_{u \in A} A_{u,t}(\tau)^c.$$

Taking probabilities and applying Equation 23 gives

$$U_t(A \cup \{s\}; \tau) - U_t(A; \tau) \geq U_t(S \cup \{s\}; \tau) - U_t(S; \tau),$$

which proves diminishing returns. The empty union has probability zero, giving normalization. Adding parents enlarges the union, giving monotonicity. Integrating these relations over $\tau \geq b_{t-1}$ proves the same properties for the finite expected discovery gain in Equation 4. □

When $\Pr[A_{s,t}(\tau)] > 0$, factoring the joint event by conditional probability gives

$$U_t(S \cup \{s\}; \tau) - U_t(S; \tau) = \Pr[A_{s,t}(\tau)] \Pr\left[\bigcap_{u \in S} A_{u,t}(\tau)^c \,\middle|\, A_{s,t}(\tau)\right]. \tag{24}$$

When $\Pr[A_{s,t}(\tau)] = 0$, the marginal is zero and no conditional probability is evaluated. At $\tau = b_{t-1}$, the strict event gives the current-frontier quantity $U_t(S; b_{t-1})$. Replacing the deterministic threshold with the independent historical comparison maximum $\widetilde{B}_t$ and averaging over that threshold gives

$$\begin{aligned} &\overline{U}_t(S \cup \{s\}) - \overline{U}_t(S) \\ &\quad = \Pr\left[B_{s,t} > \widetilde{B}_t,\ \max_{u \in S} B_{u,t} \leq \widetilde{B}_t\right] \\ &\quad = q_t(s)\big[1 - o_t(s, S)\big], \end{aligned} \tag{25}$$

recovering Equation 11. The first factor is the population DFT singleton quality; the second is the conditional probability that the other selected parents do not exceed the comparison maximum. When $q_t(s) = 0$, the marginal is zero and the conditional probability need not be evaluated.

D MAX-COVERAGE ALLOCATION

Feature weights obey $\phi_s(v) \geq 0$ and $\sum_v \phi_s(v) = 1$, and singleton qualities satisfy $q_t(s) \in [0, 1]$.

Boundary example. An initial state has feature path $(r_\star)$, hence unit mass on the shared root. Its first-generation offspring also have only the shared-root feature because the candidate itself is excluded. This intentional coarsening prevents distinct seed identifiers from creating artificial coverage before the run has generated branch structure. Under max coverage with $\lambda_t = 1$, a weaker root-only candidate has zero coverage marginal after a stronger root-only candidate is selected; if all remaining candidates are root-only, their equal allocation marginals are resolved by DFT quality. For a second-generation state with path (seed, a, s), the allocation path is $(r_\star, a)$; the raw harmonic weights are $(1/3, 1/2)$, which normalize to $(2/5, 3/5)$. Deeper paths add increasingly specific generated ancestors while retaining some shared-root mass.

Equation 15 retains the strongest selected contribution on every lineage feature. With

$$m_v(S) = \max_{u \in S} q_t(u)\phi_u(v), \tag{26}$$

the marginal is

$$\Delta_{\mathcal{C}_t}(s \mid S) = \sum_v \max\{0, q_t(s)\phi_s(v) - m_v(S)\}\,. \tag{27}$$

After selecting s, the sufficient update is $m_v \leftarrow \max\{m_v, q_t(s)\phi_s(v)\}$. Each $m_v(S)$ can only increase as the selected set grows, so every residual term has diminishing returns. Feature normalization gives $\mathcal{C}_t(\{s\}) = q_t(s)$.

First-slot identity. Max coverage satisfies

$$\Delta_{\mathcal{C}_t}(s \mid \varnothing) = q_t(s). \tag{28}$$

Consequently, the LC marginal in Equation 17 equals $q_t(s)$ for the first slot under every λ_t.

E LC Objective and Greedy Guarantee

Theorem 2 (LC objective)**.** *For fixed nonnegative qualities, $\lambda_t \in [0,1]$, and a normalized monotone submodular coverage function $\mathcal{C}_t$, the objective J_t is normalized, monotone, and submodular. Under $|S| \le K$, greedy allocation attains the standard $(1-1/e)$ approximation guarantee for J_t.*

Submodular objective. The singleton-quality term

$$M_t(S) = \sum_{s \in S} q_t(s) \tag{29}$$

is normalized, monotone, and modular because $q_t(s) \ge 0$. By assumption, $\mathcal{C}_t$ is normalized, monotone, and submodular. Equation 18 is a nonnegative linear combination of these functions for $\lambda_t \in [0,1]$, so J_t is normalized, monotone, and submodular. The cardinality-constrained greedy result of Nemhauser et al. (1978) gives

$$J_t(S_{\text{greedy}}) \ge \left(1 - \frac{1}{e}\right) \max_{|S| \le K} J_t(S). \tag{30}$$

The max-coverage function in Appendix D satisfies the premise directly: each feature contributes the maximum of fixed nonnegative singleton weights, and nonnegative sums over features preserve monotonicity and submodularity.

F Allocation Weight Schedule

Full-buffer allocation. Direct DFT and LC-DFT both consider every selectable state in the buffer. The buffer retains at most the two highest-value offspring of each direct parent, subject to an overall capacity of 1000 states. Historical bundle evidence and ancestor paths are retained separately when states leave the selectable buffer. LC-DFT applies Algorithm 1 to their DFT scores.

Sigmoid schedule. For a run of $T \ge 2$ steps indexed by $t \in \{1, \dots, T\}$, define $p_t = (t-1)/(T-1)$ and $\sigma(x) = (1+e^{-x})^{-1}$. The main configuration uses

$$\lambda_t = \frac{\sigma(6) - \sigma(12p_t - 6)}{\sigma(6) - \sigma(-6)}. \tag{31}$$

The weight decreases from one at the first step to zero at the last.

G PUCT Baseline

The baseline follows the UCT/PUCT search tradition (Kocsis & Szepesvári, 2006; Silver et al., 2018). We use the TTT-Discover parent-reuse score (Yuksekgonul et al., 2026) on the higher-is-better state-value scale V. For state s, let $Q_t^V(s)$ be the best offspring state value observed from s, using $V(s)$ until an offspring value is available. Let $N_t(s)$ count rollout attempts from s or its descendants, and let T_t count all rollout attempts. Attempts without an offspring state value increment these counts

Table 3: Task objectives and evaluation procedures, following TTT-Discover (Yuksekgonul et al., 2026).

Task	Evaluation
Erdős	Recompute the normalized step-function cross-correlation bound from the emitted certificate; minimize the verified bound.
AC1 / AC2	Validate the emitted nonnegative step sequence and deterministically recompute the AC1 upper-bound or AC2 lower-bound certificate.
Circle 26 / 32	Recompute the sum of radii after unit-square boundary and pairwise non-overlap checks at 10^{-12} tolerance.
TriMul / MLA	GPU Mode benchmarks on one H200 with reference correctness checks. TriMul reports the geometric mean of seven case means. MLA reports the arithmetic mean at prefill length 6144, using 4096 only for warm-up. Table 4 gives input shapes, numerical tolerances, and timing rules.
AHC039 / AHC058	Official testers on 150 public inputs; the search score is the mean absolute score across cases.
Denoising	OpenProblems v1.0.0 split with seed 42. Minimize Pancreas MSE subject to $0.97 \le (0.257575 - p)/(0.257575 - 0.031739) \le 1$, where p is the Poisson metric. Evaluate the lowest-MSE valid program on held-out PBMC and Tabula Muris Senis Lung data. The reported OpenProblems score is the mean of the MSE and Poisson scores, each linearly normalized to 0 for no denoising and 1 for perfect reconstruction using that test dataset's reference values, without clipping.

Table 4: Kernel evaluation on H200. Correctness is checked against the reference implementation before timing. TriMul reports the geometric mean of seven per-case arithmetic means; MLA reports the arithmetic mean for the single timed case at prefill length 6144. Here B, N, and C denote TriMul batch, sequence, and channel dimensions; $h = 128$ throughout.

Task	Numeric type	Timed benchmark cases	Correctness	Timing rule
TriMul	FP32 inputs, masks, and weights	$(B, N, C;$ mask, distribution): $(2, 256, 128;$ no, $\mathcal{N})$, $(1, 768, 128;$ no, Cauchy), $(2, 256, 384;$ yes, $\mathcal{N})$, $(1, 512, 128;$ no, $\mathcal{N})$, $(1, 1024, 128;$ no, Cauchy), $(1, 768, 384;$ yes, $\mathcal{N})$, $(1, 1024, 384;$ no, $\mathcal{N})$	18-case test suite and per-benchmark check; rtol $= 2 \times 10^{-2}$, atol $= 2 \times 10^{-2}$	Suite warm-up; 3–100 repeats per case; stop at SEM/mean $< 10^{-3}$, 10 s accumulated measured time, or 120 s wall time
MLA Decode	BF16 cache, weights	$(B,$ prefill, dim, $d_q) = (128, 6144, 7168, 1536)$; prefill 4096 is warm-up only. Decode length 1, 128 heads, 128-value and 64-RoPE dimensions	4-case test suite; timed-case check before timing and after each repeat; rtol $= 2 \times 10^{-2}$, atol $= 8 \times 10^{-3}$	0.2 s warm-up; 3–100 repeats; stop at SEM/mean $< 10^{-2}$ or 30 s accumulated measured time

but leave $Q_t^V(s)$ unchanged. If $r_t^V(s) \in \{1, \dots, N_t^V\}$ is the ordinal rank of $V(s)$, best first, the normalized linear rank prior is

$$P_t(s) = \frac{N_t^V - r_t^V(s) + 1}{N_t^V(N_t^V + 1)/2}. \tag{32}$$

Let D_t^V be the span of state values among non-initial states when they are available, or among all buffered states otherwise, with a floor of 10^{-6}. The score is

$$\mathrm{PUCT}_t(s) = Q_t^V(s) + c_{\mathrm{puct}} D_t^V P_t(s) \frac{\sqrt{1 + T_t}}{1 + N_t(s)}, \tag{33}$$

with $c_{\mathrm{puct}} = 1$. The value estimate and exploration bonus are both measured on the state-value scale V.

The rank prior assigns distinct ordinal ranks to equal values. Candidates are ordered by decreasing $\mathrm{PUCT}_t(s)$, then decreasing $V(s)$, with remaining ties following current buffer order. Parent selection follows the released TTT-Discover baseline procedure.

H Tasks and Experimental Settings

H.1 Task Definitions

H.2 Training and Computational Resources

Our computational setup uses one node with eight NVIDIA H200 GPUs and 224 CPUs for generation and training with either Qwen3-8B or Qwen3.8-27B. AC1 and AC2, Erdős, and 27B circle packing use additional CPU evaluation nodes with 256 GiB of memory each. GPU-kernel evaluation uses one H200 per candidate.

Table 5: State values V and offspring rewards R, both oriented so that larger is better. Here r is the native verifier score and $\bar{s}$ is the mean absolute score over public AHC inputs. The mappings preserve the task's ordering but can differ in scale.

Task	Native objective	State value V	Offspring reward R
Erdős / AC1	minimize bound r	$-r$	$1/(10^{-8}+r)$
AC2	maximize bound r	r	r
Circle 26 / 32	maximize radii sum r	r	r
TriMul	minimize runtime r	$-r$	$1500/r$
MLA Decode	minimize runtime r	$-r$	$5000/r$
AHC039	maximize mean score $\bar{s}$	$\bar{s}$ for accepted states	$\bar{s}/1500$
AHC058	maximize mean score $\bar{s}$	$\bar{s}$ for accepted states	$\bar{s}/(1500 \cdot 2000)$
Denoising	minimize Pancreas MSE r	$-r$	$1/r, r > 0$

Invalid programs do not become selectable states. Their failed attempts count toward the bundle budget but contribute no reward to the DFT reference distribution. Denoising requires strictly positive finite MSE; nonpositive values are excluded rather than treated as valid rewards. Parsing or system failures exclude the affected attempts, not completed evaluations elsewhere in the bundle; Appendix B gives the history rules.

Table 6: Task-specific settings of the Qwen3-8B runs used for the reported results. Length limits are measured in tokens.

Task	Steps	Attempts	Context limit	Prompt + thinking limit	Response limit	Learning rate	KL coefficient
Erdős' minimum overlap	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
Autocorrelation inequality (AC1)	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
Autocorrelation inequality (AC2)	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
Circle packing ($n = 26$)	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
Circle packing ($n = 32$)	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
TriMul	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
MLA Decode	50	25,600	32,768	26,000	32,768	4×10^{-5}	0.10
AHC039 (Geometry)	50	25,600	32,768	22,000	32,768	4×10^{-5}	0.01
AHC058 (Scheduling)	50	25,600	32,768	25,000	32,768	2×10^{-5}	0.01
Denoising (Pancreas)	50	25,600	32,768	20,000	24,000	4×10^{-5}	0.10

Table 2 reports means only for settings with three completed runs, except circle packing, which reports one run per method. Each step uses $K = 8$ parents and $n = 64$ rollouts per parent, for 512 attempts. Context limits include the prompt and response; response limits include generated thinking and code. The prompt-plus-thinking limit reserves context for code completion. Token limits correspond to the runs contributing the reported scores.

Test-time updates use LoRA rank 32, scale 16, and zero dropout on the attention projections and feed-forward linear layers. Weight decay is zero, and the Adam coefficients are $(0.9, 0.95)$. One model update follows each completed selection batch. Table 6 lists the task-specific learning rates, KL coefficients, and budgets of the runs used for the reported Qwen3-8B results. The AHC learning rates and KL coefficients follow the released task configurations.

Within each task, our methods share the same search and training pipeline, including buffer management, training settings, and the task evaluator.

H.3 Search Budgets and Repeated Runs

The context limit covers the prompt and generated response. Within it, a separate prompt-plus-thinking limit reserves room to finish the code; the response limit caps the total generated thinking and code tokens. A response limit equal to the context limit therefore allows generation only within the space remaining after the prompt. All reported Qwen3-8B runs use 50 steps and $25{,}600$ attempts; their generation limits are task-specific (Table 6). The reported methods share these token limits within each task. Appendix H.5 gives the budgets of the 27B results.

Repeated runs. For each Qwen3-8B task and selector, we average the best verifier score from three complete runs; circle packing reports one run per method. Denoising reports the mean held-out score of the three programs selected during Pancreas search.

Published and reproduced baselines. Table 2 uses our PUCT mean where three runs are complete and our single-run PUCT result for circle packing; remaining populated baseline cells retain the Qwen3-8B values from TTT-Discover Tables 2–3 (Yuksekgonul et al., 2026). For circle packing, we compare one run each for PUCT, DFT, and LC-DFT, all using the same feasibility repair. No result

Table 7: H200 runtimes of the three released TTT-Discover MLA programs under our evaluation protocol. Each program is evaluated once without source modifications, using repeated calls to estimate its mean runtime.

Released program	Runtime (μs) ↓	Correctness
`mla_code_1.py`	2976.789	Pass
`mla_code_2.py`	**1876.903**	Pass
`mla_code_3.py`	1901.617	Pass

from a different model fills a missing 8B cell. The reported mathematics runs and our runs use 50 steps of 512 attempts, but a published best result is not a three-run mean. Bold in this table compares our three-run means, or the single-run repaired PUCT, DFT, and LC-DFT sums for circle packing, with ties determined at the displayed precision and the two denoising datasets judged separately.

The original Qwen3-8B Erdős value, 0.380932, is a program-reported bound accepted under the released tolerance check; our values are recomputed C5 verifier scores. Circle-packing DFT used the earlier CPU backend, while PUCT and LC-DFT used the current backend.

The Qwen3-8B AHC058 scores in Table 2 are public-case means multiplied by 150, not AtCoder submission totals. Only complete three-run sets contribute to these means.

H.4 Evaluation Protocols

Kernel evaluation uses one H200, with numerical checks and timing procedures given in Table 4. AHC search evaluates programs on 150 public inputs using the official tester. Denoising uses OpenProblems v1.0.0 data, the released Poisson constraint, and strictly positive finite MSE values; Table 3 specifies the data split and score.

H.4.1 MLA Reference Evaluation on H200

We evaluate the three released TTT-Discover MLA programs without modifying their source code and use the lowest runtime in the literature-results column of Table 1. Although TTT-Discover reports its final MLA results on MI300X, its search obtains correctness and runtime rewards on H200 (Yuksekgonul et al., 2026, Section 4.2 and Appendix C.1). The released prompt explicitly specifies NVIDIA H200, as does the prompt used in our experiments. We compare the released programs and our own under the same H200 evaluation settings, using the same evaluator, input configuration, numerical tolerances, and timing procedure.

All three programs pass the correctness checks; their runtimes are reported in Table 7. Measurements use one H200 with PyTorch 2.13.0+cu129, CUDA 12.9, and Triton 3.7.1. Each value averages repeated calls at batch size 128 and prefill length 6144, following Table 4; prefill length 4096 is used only for warm-up. The timed interval includes the complete program call and final GPU synchronization, excluding input generation, reference checks, and initial compilation. The lowest mean, $1876.903\,\mu$s, is rounded to $1876.9\,\mu$s in the main table.

The leading submission on the AMD MI300X MLA leaderboard (GPU MODE, 2025) uses an AMD-specific HIP implementation and cannot be evaluated unchanged on H200. Porting it would alter the submitted implementation. We therefore use the unmodified TTT-Discover programs as the H200 reference, while the MI300X leaderboard results describe performance on a different target device.

H.4.2 Circle-Packing Feasibility Repair

The online verifier allowed a geometric tolerance of 10^{-12} in the reported 8B runs and 10^{-14} in the current 27B circle-packing runs. For each best solution from our experiments, selected by its original score, we keep the centers fixed and reduce each radius by the maximum of zero, its boundary violation, and half its largest incident overlap. Reductions are computed at high precision and rounded upward to multiples of 10^{-30}; the resulting constructions are then verified using exact rational inequalities. Tables 2 and 1 report the strictly feasible radius sums rounded to 12 decimal

places, matching Packomania's display precision. This postprocessing does not change the online rewards or search trajectories.

H.4.3 Denoising Held-Out Evaluation and Compatibility Repairs

The reported denoising means use three runs per method with a $20{,}000$-token prompt-plus-thinking limit, a $24{,}000$-token response limit, and a $32{,}768$-token context limit, as listed in Table 6. Each run comprises 50 steps and $25{,}600$ attempts. Each run's program was selected on Pancreas and then evaluated on held-out PBMC and Tabula. The held-out means include two post-search compatibility repairs: PUCT repB caps the graph neighbor count at the test dataset's cell count minus two, and DFT repC omits an inverse transform on data already represented in the original gene space. The reported values use the corresponding repaired-program evaluations; the original programs are retained in the archive. The PUCT Tabula mean includes two completed evaluations and one 1200-second timeout assigned zero. These means describe the stated generation budget, rather than the later $26{,}000$-token prompt-plus-thinking and $32{,}768$-token response configuration.

H.5 27B Results and Selection Scope

Table 1 reports the best verifier-accepted generated solution from one Qwen3.8-27B LC-DFT run per task. Selection covers the first 20 steps for Erdős, Circle 26, and Circle 32 ($10{,}240$ attempts each), 25 for AC1, AC2, and Pancreas ($12{,}800$ each), 40 for MLA ($20{,}480$), and 50 for TriMul and each AHC task ($25{,}600$ each). Each step allocates eight bundles of 64 attempts. The ten runs use a $65{,}536$-token context and response limit, with a $40{,}000$-token prompt-plus-thinking limit.

Each selected candidate's original score was matched to its completed verification record, including correctness and a finite raw score; circle-packing scores are subsequently repaired as described above. Initial solutions and candidates beyond the reported steps are outside this selection scope. The two AHC values in Table 1 are recorded AtCoder submission totals for the unchanged programs selected by public-test performance. Held-out denoising evaluates the unchanged program selected by Pancreas MSE within the 25-step checkpoint on frozen PBMC and Tabula splits. Its OpenProblems-aligned scores are 0.7010600374 and 0.7270120112, displayed at the references' two-decimal precision in Table 1.

H.6 Literature Results and Best-Known Records

Table 1 presents representative literature results alongside best-known records. The literature-results column summarizes relevant task-level comparisons; Tables 8 and 9 retain the broader method comparisons. TTT-Discover entries in that column use gpt-oss-120B, including the released MLA programs. SimpleTES also uses gpt-oss-120B and supplies the Erdős bound 0.380868 and AC2 bound 0.962694 (Ye et al., 2026, Table 1 and Supplementary Table 17). These values were checked against its July 27, 2026 version on September 25; its main comparison table uses reference results as of April 22. AlphaEvolve V2 uses the Gemini-2.0 family, and its six-decimal circle values retain the precision reported in TTT-Discover's comparison table. Table 8 retains the TTT-Discover Erdős value 0.380876 as a program-reported estimate, not as a recomputed C5 verifier score. These published results retain their original search budgets and initialization choices.

The best-known column retains source records checked on September 22, 2026, with the AHC and GPU leaderboard records updated on September 25. The EinsteinArena leaderboards supply 0.38085857 for Erdős (EinsteinArena, n.d.c), 1.50274365 for AC1 (EinsteinArena, n.d.a), and 0.96359 for AC2 (EinsteinArena, n.d.b). Circle-packing records are from Packomania's variable-radius, maximum-sum problem (Specht, 2026): 2.635983084919 and 2.939572771205. The high-precision Circle comparison uses these records and our strictly feasible constructions; six-decimal literature entries are not ranked by appending zeros. Equality at a common display precision is treated as a tie.

SimpleTES v2 (July 27, 2026), Supplementary Table 13, supplies the H200 TriMul reference (Ye et al., 2026): $1.020 \pm 0.001\,\mathrm{ms}$, or $1020.0 \pm 1.0\,\mu\mathrm{s}$. Its Supplementary Table 12 reports $1.122 \pm 0.008\,\mathrm{ms}$ on H100 with Triton 3.6.0. These are paper-reported measurements; the deviations summarize repeated timing trials, not independent search runs. Our $881.7\,\mu\mathrm{s}$ result uses the H200 protocol in Table 4. The MLA literature reference uses our September 23 H200 re-evaluation in Appendix H.4.1.

Table 8: Published AI discovery results on mathematical construction tasks. Values and model descriptions follow Yuksekgonul et al. (2026, Tables 2–3) and Ye et al. (2026, Table 1 and Supplementary Table 17), at their reported precision. Current best-known records and our 27B constructions remain in Table 1.

Method	Model	Erdős ↓	AC1 ↓	AC2 ↑
AlphaEvolve	Gemini-2.0 Pro/Flash	0.380924	1.50530	0.8962
AlphaEvolve V2	Gemini-2.0 Pro/Flash	0.380924	1.50317	0.9610
ThetaEvolve	R1-Qwen3-8B	–	1.50681	0.9468
ThetaEvolve (warm start)	R1-Qwen3-8B	–	1.50314	–
TTT-Discover	Qwen3-8B	0.380932†	1.50525	0.9472
TTT-Discover	gpt-oss-120B	0.380876†	1.50287	0.9591
SimpleTES	gpt-oss-120B	0.380868	1.503871	0.962694

Method	Model	Circle 26 ↑	Circle 32 ↑
AlphaEvolve	Gemini-2.0 Pro/Flash	2.635862	2.937944
AlphaEvolve V2	Gemini-2.0 Pro/Flash	2.635983	2.939572
ShinkaEvolve	Ensemble	2.635982	–
ThetaEvolve	R1-Qwen3-8B	2.635983	–
TTT-Discover	Qwen3-8B	2.635983	2.939572

Methods: AlphaEvolve (Novikov et al., 2025), AlphaEvolve V2 (Georgiev et al., 2025), ThetaEvolve (Wang et al., 2025), ShinkaEvolve (Lange et al., 2026), TTT-Discover (Yuksekgonul et al., 2026), and SimpleTES (Ye et al., 2026). The warm-start ThetaEvolve result refines the AlphaEvolve V2 construction with AC1 bound 1.50317; it is distinct from the from-scratch result. † Published program estimate. – means not reported in the cited comparison. Circle values retain six decimal places and are not padded for comparison with 12-decimal constructions.

The TriMul entry in the best-known column uses the official H100 leaderboard's first-place submission 781368, checked on September 25, 2026: $1064.9228290162392\,\mu$s, displayed as $1064.9\,\mu$s (GPU MODE, 2026). The literature-results column reports both SimpleTES measurements: $1020.0\,\mu$s on H200 and $1122.0\,\mu$s on H100 (Ye et al., 2026, Supplementary Tables 12–13). The MLA entry uses the official MI300 leaderboard's first-place submission 31138, checked on the same date: $1787.4566666666667\,\mu$s, displayed as $1787.5\,\mu$s (GPU MODE, 2025). This is the official aggregate score, rather than an average of TTT-Discover's three historical MI300X measurements. Devices are labeled in Table 1.

The AHC references, updated on September 25, 2026, are the AtCoder submission totals of $589{,}491$ for AHC039 from submission 79052825 (AtCoder, 2026a) and $851{,}484{,}676$ for AHC058 from submission 79233573 (AtCoder, 2026b). These are post-contest submission records, rather than the original contest winners' scores. Our AHC entries in Table 1 report $575{,}374$ for AHC039 (submission 79475779) and $847{,}260{,}907$ for AHC058 (submission 79493481), recorded from the official submission pages.

For held-out denoising, SimpleTES v2 Supplementary Table 15 reports PBMC 0.71 and Tabula 0.74, after search on Pancreas. TTT-Discover reports 0.71 and 0.73 with gpt-oss-120B, so SimpleTES provides a tied best PBMC reference and the higher Tabula reference at the reported precision. These held-out scores are kept separate from online Pancreas results.

I Best-So-Far Search Trajectories

Figure 3 shows progress over the search budget for the Qwen3-8B comparisons in Table 2.

J Search-Tree Analysis

Table 10 summarizes 54 Qwen3-8B runs across eight tasks. Figure 2 shows the nine Erdős runs: three repetitions each of PUCT, DFT, and LC-DFT. Each run has 50 steps and eight selected-parent slots per step. Step identifiers in the figure are zero-based, $t = 0, \ldots, 49$. These diagnostics use recorded parent selections and historical ancestry, including ancestors no longer in the selectable buffer.

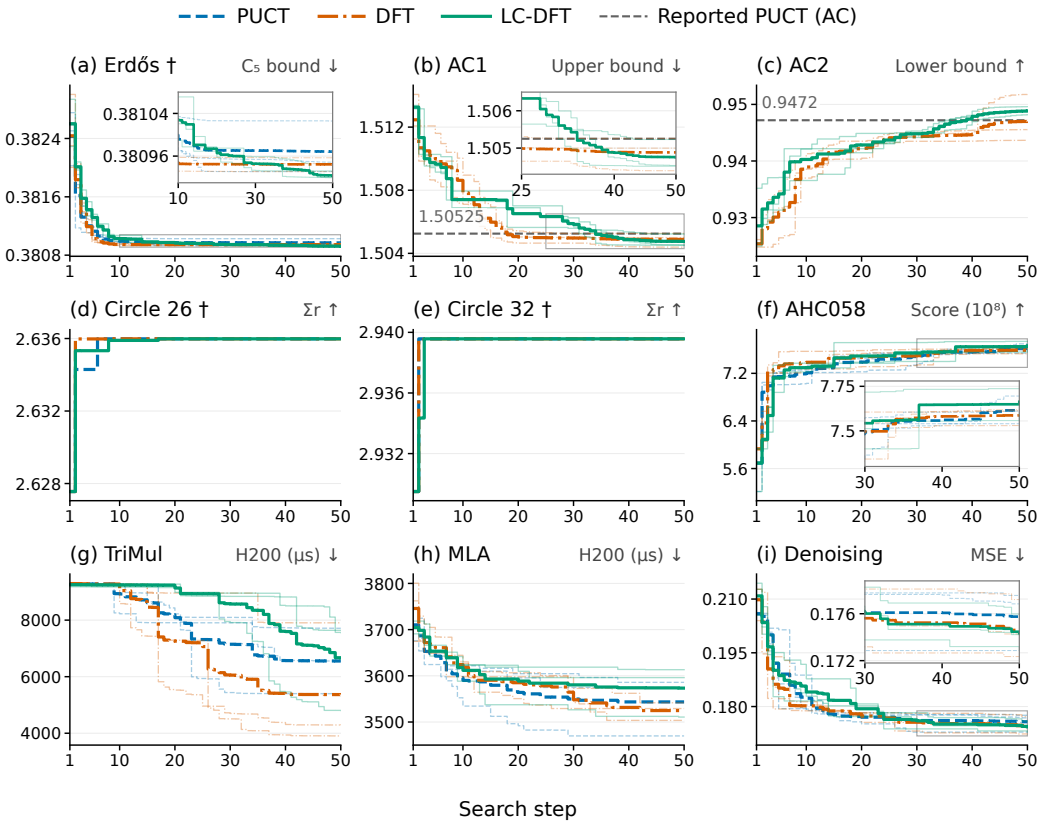


Figure 3: **Best-so-far search trajectories.** Thin lines show individual runs; thick lines show their pointwise means. Insets show final-stage detail. Gray dashed lines in AC1/AC2 are the PUCT best values reported by TTT-Discover, rather than local trajectories or replicate means. A dagger marks differing execution protocols within a task; external references may also use different protocols. Circle curves show recorded verifier scores before the precision repairs used in Table 2. AHC058 reports public-case means multiplied by 150, with the axis in units of 10^8. TriMul and MLA report H200 latency in microseconds; denoising shows the online Pancreas MSE.

Table 9: Published AI results on GPU-kernel optimization, heuristic programming, and denoising. TriMul reports H200 runtimes (Ye et al., 2026, Supplementary Table 13); AHC reports AtCoder totals (Yuksekgonul et al., 2026, Table 6); denoising reports scores on PBMC and Tabula (Ye et al., 2026, Supplementary Table 15).

Task	Method	Result
TriMul (H200, μs) ↓	TTT-Discover	1064.0
	Aster	1101.0
	K-Search	1075.0
	SimpleTES	1020.0
AHC039 ↑	ALE-Agent	550,647
	ShinkaEvolve	558,026
	TTT-Discover	567,062
AHC058 ↑	ALE-Agent	848,373,282
	TTT-Discover	848,414,228
PBMC ↑	TTT-Discover	0.71
	SimpleTES	0.71
Tabula ↑	TTT-Discover	0.73
	SimpleTES	0.74

TriMul values are the H200 measurements reported by SimpleTES, converted from milliseconds to microseconds. MLA re-evaluations are reported separately in Table 7. For AHC, ALE-Agent uses Gemini-2.5 Pro for AHC039 (Imajuku et al., 2025) and Gemini-3 Pro Preview / GPT-5.2 for AHC058 (Sakana AI, 2026); ShinkaEvolve (Lange et al., 2026) uses an ensemble and has no reported AHC058 value; TTT-Discover uses gpt-oss-120B. Denoising entries average normalized MSE and Poisson scores after search on Pancreas.

Table 10: Search-tree allocation and parent reuse for Qwen3-8B. Overlap is the mean ancestor Jaccard overlap, with the range of run-level values in brackets where multiple runs are available. Reuse and successful reuse are equal-weight means of run-level percentages. Successful reuse is conditioned on selecting a previously selected parent; – denotes no such selections.

Task	Method	Runs	Ancestor overlap	Reuse (%)	Successful reuse (%)
Erdős	PUCT	3	0.729 [0.720, 0.735]	0.0	–
	DFT	3	0.807 [0.799, 0.816]	37.6	5.62
	LC-DFT	3	0.329 [0.316, 0.337]	32.5	5.91
AC1	DFT	3	0.801 [0.784, 0.825]	45.0	3.30
	LC-DFT	3	0.313 [0.297, 0.323]	44.2	2.35
AC2	DFT	3	0.806 [0.787, 0.818]	45.7	2.43
	LC-DFT	3	0.298 [0.270, 0.323]	39.4	3.78
Circle 26	PUCT	1	0.585	0.0	–
	DFT	1	0.719	25.5	0.00
	LC-DFT	1	0.267	50.3	1.00
Circle 32	PUCT	1	0.651	0.0	–
	DFT	1	0.738	14.8	1.69
	LC-DFT	1	0.232	32.5	1.54
TriMul	PUCT	3	0.656 [0.606, 0.697]	2.9	0.00
	DFT	3	0.734 [0.708, 0.766]	47.1	5.70
	LC-DFT	3	0.375 [0.366, 0.382]	39.8	6.92
MLA	PUCT	3	0.448 [0.350, 0.557]	1.0	3.70
	DFT	3	0.594 [0.533, 0.628]	69.0	1.32
	LC-DFT	3	0.231 [0.148, 0.331]	57.6	1.75
AHC058	PUCT	3	0.643 [0.631, 0.650]	0.0	–
	DFT	3	0.737 [0.699, 0.774]	80.3	2.92
	LC-DFT	3	0.254 [0.196, 0.309]	52.0	3.36

Ancestor overlap. Let $A(s)$ be the set of generated strict ancestors of parent s, excluding s itself, initial seed states, and the abstract problem root. For two selected parents with known ancestry and a

nonempty ancestor union, their Jaccard overlap is

$$\mathrm{Jac}(s, s') = \frac{|A(s) \cap A(s')|}{|A(s) \cup A(s')|}. \tag{34}$$

For the selected slots $(s_{t,1}, \ldots, s_{t,K_t})$, the per-step overlap is the arithmetic mean of $\mathrm{Jac}(s_{t,i}, s_{t,j})$ over valid pairs $i < j$. Pairs with unknown ancestry or an empty union are excluded; if no valid pairs remain, the step's overlap is undefined rather than assigned zero. Slots are not deduplicated before averaging. Lower overlap indicates that the selected parents share less generated ancestry. This is an unweighted structural diagnostic, distinct from the quality- and depth-weighted LC allocation objective in Equation 18.

Parent reuse. Let $\mathcal{U}_{<t} = \bigcup_{u<t}\{s_{u,1}, \ldots, s_{u,K_u}\}$ contain parents selected in earlier steps. The cross-step reuse fraction is

$$\mathrm{Reuse}_t = \frac{1}{K_t} \sum_{i=1}^{K_t} \mathbf{1}\{s_{t,i} \in \mathcal{U}_{<t}\}. \tag{35}$$

Repeated slots within the current step do not by themselves establish earlier reuse. This statistic counts direct selections of the same parent state, rather than the descendant rollout attempts included in PUCT visit counts. It requires complete selection history through the reported step.

For both diagnostics, thin curves show individual runs and the method curve gives the equal-weight mean across the three repetitions at each step, using only defined values. Undefined steps remain gaps, and curves are not smoothed. Ancestor overlap measures where allocation concentrates in the genealogy; parent reuse measures repeated investment in exactly the same state.

Successful reuse. A reused selection is successful when its bundle contains a best offspring of that step and this offspring strictly improves the global best before the step. Tied step winners share credit when they improve the previous record; merely matching that record is not a success. The successful-reuse rate divides successful reused selections by all reused selections in the run. A run with no reused selections has no defined rate.

Run-level aggregation. For Table 10, ancestor overlap pools all valid within-step pairs in each run; the reuse fraction pools selected slots across steps. We then average each run-level statistic with equal weight across executions. The overlap intervals give the minimum and maximum run-level means, not confidence intervals. Across the six tasks with repeated DFT and LC-DFT runs, every LC-DFT overlap value is below every DFT value for the same task. All 18 DFT runs and all 18 LC-DFT runs in these comparisons contain at least one successful reused selection.

Tree illustration. In Figure 2, a tied best value retains the earlier representative. Dashed links connect successive best representatives and need not be parent–offspring edges; dotted links identify the final solution's ancestry. The winning offspring at step 34 is not on that final ancestor chain, whereas the offspring at step 43 from the repeated parent is. The two events illustrate cross-branch exploration and productive parent reuse within the same search.

K Extended Related Work

Search budgets and model adaptation. Search budgets can be distributed among competing procedures or among solutions within an evolving search. A budget-matched, search-only study uses partial-run scores to allocate the remaining budget among search procedures (Gupta et al., 2026). Model adaptation offers a complementary way to improve subsequent proposals: test-time training updates parameters using inference-time information (Sun et al., 2020; Gozeten et al., 2025), test-time reinforcement learning uses unlabeled reasoning instances (Zuo et al., 2025), and ThetaEvolve incorporates test-time learning into evolutionary program search (Wang et al., 2025). Parent allocation determines which solutions receive further attempts, whether the generator remains fixed or is updated between batches.

Selection in reasoning and evolutionary search. The objects selected range from intermediate reasoning states to complete solutions and populations. Tree of Thoughts, RAP, and LATS explore intermediate reasoning states through tree search (Yao et al., 2023; Hao et al., 2023; Zhou et al., 2024). WU-UCT coordinates parallel simulations through statistics for pending outcomes (Liu et al., 2020), while determinantal beam search selects interacting subsets during decoding (Meister et al., 2021). At the population level, EvoX adapts evolution strategies (Liu et al., 2026), Group-Evolving Agents shares experience across evolving agents (Weng et al., 2026), and EvoDiverse maintains tempered populations of scientific hypotheses (Wang et al., 2026). PTB-Search instead selects extracted equation terms after proposal generation (Li, 2026). Our selection precedes generation: it chooses complete solutions to provide contexts for the next batch of rollout bundles.

Parallel acquisitions and subset selection. Batch acquisition methods account for interactions among jointly evaluated candidates. BoTorch supports Monte Carlo acquisition optimization through sample-average approximation (Balandat et al., 2020), while Batch BORE uses density-ratio estimation (Oliveira et al., 2022). Local penalization and DPP batching discourage redundant evaluations through geometry or posterior kernels (Gonzalez et al., 2016; Nava et al., 2022). Related selection methods combine uncertainty filtering with submodular diversity (Wei et al., 2015) or balance relevance and redundancy through MMR (Carbonell & Goldstein, 1998). These approaches assess candidates in relation to the selected set, matching the allocation perspective developed here for solution reuse.