FEED: FEEDBACK-GUIDED SELF-TRAINING AND PREFERENCE OPTIMIZATION FOR EFFICIENT TEST-TIME DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly support scientific discovery through test-time learning, but generating candidate solutions is costly, and high-quality examples remain scarce. To improve learning under limited computational budgets, we introduce FEED (Feedback-Enhanced Efficient Discovery), which turns evaluation feedback into a unified objective for quality-weighted self-training and prompt-matched success–failure preference optimization. FEED draws valid programs from current and historical evaluations, assigning each a uniform base weight and additional emphasis according to solution quality. The same weights scale preference supervision against failures generated under the same original prompt, while every selected valid program retains direct supervision, including those without matched failures. Evaluated with Qwen3-8B on six tasks spanning mathematical construction, heuristic design, and function optimization, FEED achieves earlier discovery of high-quality solutions on multiple tasks than strong baselines, including TTT-Discover, under matched one-hour budgets covering generation, evaluation, and learning. For example, on CVRP-ACO, FEED reaches the 10.2 route-cost threshold in all five runs versus none for TTT-Discover, reducing the one-hour restricted mean time to threshold from 60.00 to 17.62 minutes. Further circle-packing evaluations show efficiency gains across Qwen3 model sizes and solution-quality improvements in two other model families. Together, these results demonstrate that FEED's feedback-guided joint optimization of self-training and success–failure preferences enables earlier discovery of high-quality solutions under fixed test-time budgets.

1 INTRODUCTION

Advances in large language models (LLMs) have opened new possibilities for scientific discovery, including the search for mathematical constructions and algorithms (Georgiev et al., 2025). Recent work has improved lower bounds for classical Ramsey numbers (Nagda et al., 2026) and discovered new multiagent learning algorithms (Li et al., 2026). In these studies, executable programs connect model-generated proposals with computational experiments. Systems such as AlphaEvolve and

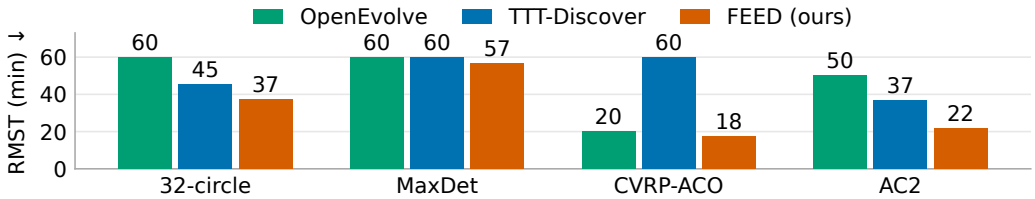

Figure 1: **Time to target quality (lower is better).** Five-seed restricted mean time to threshold (RMST), capped at 60 minutes. We compare FEED with TTT-Discover (Yuksekgonul et al., 2026) and OpenEvolve (Sharma, 2025), an open-source version of AlphaEvolve (Novikov et al., 2025).

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

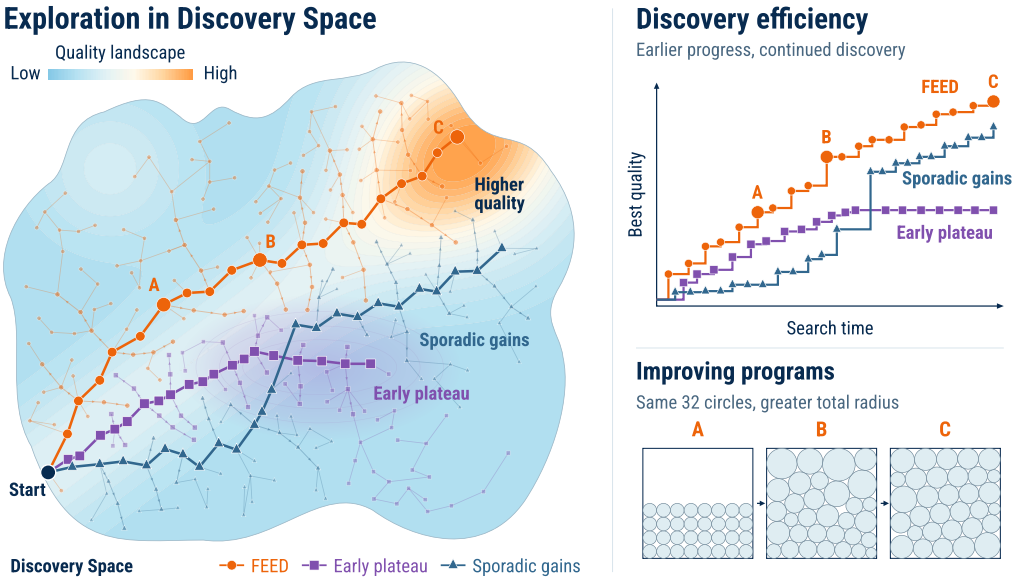

Figure 2: **Discovery efficiency under a limited budget.** Observed search patterns include early plateaus, sporadic gains, and earlier high-quality discovery with FEED. Nodes denote programs. Landscape colors encode quality. Panels A–C show three 32-circle constructions from one FEED lineage. Landscape, paths, and curves schematically summarize observations, not exact search geometry, timing, or scores.

ShinkaEvolve provide frameworks for this approach (Novikov et al., 2025; Lange et al., 2026). They iteratively generate candidate programs and use evaluations of validity and quality to guide subsequent search. Test-time learning takes this process further by using evaluation feedback to update the proposing model during discovery.

Recent methods advance test-time discovery by learning from evaluated candidates. TTT-Discover combines search with an entropic objective that emphasizes high-reward outcomes and has achieved state-of-the-art results on several scientific discovery problems (Yuksekgonul et al., 2026). CALM and EvoTune further explore reinforcement learning and preference learning, respectively, to incorporate evaluation feedback into model updates (Huang et al., 2026; Surina et al., 2025). These methods learn from candidates generated during search, where high-quality solutions are scarce and improvements emerge irregularly. Search can plateau after initial progress or advance through sporadic gains, as depicted in Figure 2. As a result, additional sampling incurs generation and evaluation costs without ensuring corresponding gains in solution quality. This makes learning from existing candidates particularly important. Improving discovery efficiency therefore calls for more effective learning from available evaluation feedback.

Learning effectively from this feedback presents three challenges. First, generating and evaluating candidates is costly, limiting the amount of experience available for model adaptation. Second, high-quality candidates are scarce and improvements emerge irregularly, making it difficult to continually obtain strong positive examples during search. Third, evaluation scores alone provide limited supervision. They quantify candidate performance but do not directly convey how the evaluated programs achieve it. Together, these challenges motivate the question: *how can models extract more useful supervision from available evaluation feedback to improve discovery efficiency?*

To address these challenges, we introduce FEED (Feedback-Enhanced Efficient Discovery), which aims to accelerate discovery by making each round of learning more effective. FEED leverages quality-weighted self-training and prompt-matched success–failure preference learning. To address the scarcity of high-quality candidates, it draws valid programs from both current and historical evaluations and assigns greater learning weight to stronger solutions. A uniform base weight retains direct supervision for every selected valid program, while the quality-dependent component con-

centrates additional emphasis on better examples. To enrich supervision beyond evaluation scores, FEED compares selected valid programs with failures generated under the same original prompt. These comparisons provide preference supervision, scaled by the same quality weights used for self-training. Valid programs without matched failures continue to contribute direct supervision. Together, these mechanisms guide subsequent search toward high-quality solutions through more effective learning from evaluated candidates.

We evaluate FEED using Qwen3-8B (Yang et al., 2025) across six tasks spanning mathematical construction, heuristic design, and function optimization, measuring both solution quality and time to target attainment. Under matched one-hour budgets that include generation, evaluation, and learning, FEED discovers high-quality solutions earlier than TTT-Discover (Yuksekgonul et al., 2026) on multiple tasks. Figure 1 provides an overview of threshold-attainment efficiency on four tasks. On capacitated vehicle routing with an ant-colony solver (CVRP-ACO) (Huang et al., 2026), FEED reduces restricted mean time to threshold from 60.00 to 17.62 minutes relative to TTT-Discover. Beyond earlier target attainment, FEED also approaches strong published reference results within one hour, measured by the mean best-so-far score across five independent seeds. On 32-circle packing, this mean is approximately 0.25% below a conventional-optimization reference (Berthold et al., 2026). On the autoconvolution problem AC1 studied by Matolcsi & Vinuesa (2010), the five-seed mean is 0.54% above their published numerical reference.

To examine the contribution of joint learning, we compare FEED with variants that retain only quality-weighted self-training or only prompt-matched success–failure preference learning. Joint learning reduces time to target attainment across multiple tasks. To assess the contribution of parameter updates, we compare FEED with a fixed-model baseline under the same contextual search rules. FEED improves discovery efficiency on multiple tasks even after accounting for learning costs. We extend the comparison to a different search framework using OpenEvolve (Sharma, 2025), an open-source version of AlphaEvolve (Novikov et al., 2025). FEED again achieves lower time to threshold on multiple tasks. Beyond these method comparisons, additional evaluations on 32-circle show efficiency gains with larger Qwen3 models, with benefits varying across model families (Section 3.3.2).

Our contributions are threefold:

- **Unified self-training and preference learning.** We introduce FEED, a test-time program discovery method that leverages quality-weighted self-training with prompt-matched success–failure preference learning. Both objectives share the same quality weights, while all selected valid programs receive direct supervision, including those without matched failures.
- **Discovery efficiency under matched budgets.** We demonstrate that FEED improves discovery efficiency over TTT-Discover across multiple tasks, as measured by solution quality and target attainment, under matched one-hour budgets that account for both search and learning costs.
- **Ablation analysis of learning benefits.** Through ablations, we show that joint training outperforms either objective alone on several tasks. We further demonstrate that parameter updates can provide gains that outweigh their time cost relative to fixed-model search under the same contextual search rules. Additional evaluations examine how these benefits vary across tasks, model sizes, and model families.

Due to space limits, more related work is provided in Appendix A.

2 FEED: Efficient Test-Time Discovery

2.1 Preliminaries

For a fixed task x, a language model $\pi_\theta(y \mid p)$ with parameters θ generates a program y from a prompt p containing the task and, optionally, previously evaluated programs and feedback. A fixed evaluator determines validity $v_x(y) \in \{0, 1\}$ (one for valid programs) and assigns valid programs a finite quality $q_x(y)$, with higher values indicating better solutions.

Our objective is to discover high-quality solutions earlier under fixed computational resources. Let B be the elapsed-time budget, including generation, evaluation, and learning, and let τ_i be the evaluation completion time of newly generated candidate y_i. The best verified quality within this budget is $Q_x(B) = \max_{i:\, \tau_i \le B,\, v_x(y_i)=1} q_x(y_i)$. We assess discovery efficiency through $Q_x(B)$ at a

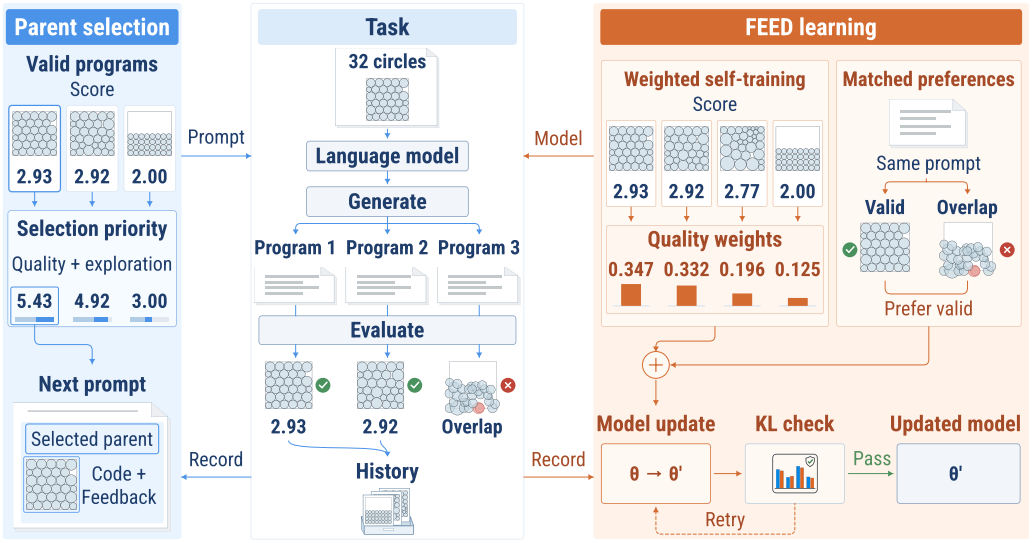

Figure 3: **FEED's joint learning within contextual search.** Quality-weighted self-training and prompt-matched success–failure preference learning share the same program weights. Unpaired valid programs retain direct supervision. Updated parameters guide subsequent candidate generation.

fixed budget and the time to reach a target quality $q^\star$, defined as $T_x(q^\star) = \inf\{B \geq 0 : Q_x(B) \geq q^\star\}$. Pre-supplied seed programs are excluded, and $Q_x(B)$ is undefined until a valid new program is evaluated. The infimum considers only budgets with defined $Q_x(B)$, with $T_x(q^\star) = \infty$ if the target is never reached.

Method overview. FEED leverages quality-weighted self-training and prompt-matched success–failure preference learning within contextual search (Figure 3). Each round generates and evaluates candidate programs using previously evaluated programs as context. FEED selects valid programs from current and historical evaluations for self-training and pairs them with available failures generated under the same original prompt for preference learning. Shared quality weights connect the two objectives, and the resulting parameter updates guide subsequent generation. Appendix C.2 provides the parent-selection rule and full search-and-learning loop.

2.2 Contextual Search and Evaluation

FEED acquires learning experience through contextual search, using previously evaluated programs and their feedback to guide candidate generation, as in FunSearch and AlphaEvolve (Romera-Paredes et al., 2024; Novikov et al., 2025). At round t, with current model parameters θ_t, it selects a parent program from the search archive before round t, $\mathcal{H}_t^-$, using a PUCT-inspired rule, drawing on prior-guided tree search (Silver et al., 2017) and its use in TTT-Discover (Yuksekgonul et al., 2026). The task description, selected program, and its evaluation feedback form the generation prompts $p_{t,i}$, from which candidates are sampled as $y_{t,i} \sim \pi_{\theta_t}(\cdot \mid p_{t,i})$. Evaluating these candidates produces records $\mathcal{E}_t$ that inform subsequent search and model updates. For learning, FEED draws on these current records and a historical pool of eligible valid programs, maintained separately from the parent-search archive, to construct the joint supervision described below.

2.3 Joint Self-Training and Preference Optimization

Quality-weighted self-training. For self-training, FEED learns from valid programs in both current evaluations and prior rounds. It selects programs according to their evaluation quality from the current records $\mathcal{E}_t$ and the historical learning pool $\mathcal{P}_t$. Let G_t index the $m = |G_t| > 0$ selected programs. Each selected program y_i serves as a target completion for its original generation prompt

p_i. Self-training then optimizes the likelihood of these completions, with quality-dependent weights defined below.

For each selected program, we compute the mean-token log likelihood $\ell_\theta(y_i \mid p_i) = |M_i|^{-1} \sum_{k \in M_i} \log \pi_\theta(y_{ik} \mid p_i, y_{i,<k})$, where M_i is the nonempty set of scored completion positions, y_{ik} is the token at position k, and $y_{i,<k}$ denotes the preceding tokens.

These selected programs provide valid implementations to learn from, but their solution quality varies. FEED therefore retains direct supervision from every selected program while assigning greater weight to stronger solutions. Specifically, half of the total training weight is distributed uniformly across the selected programs, and the remaining half is allocated according to evaluation quality. Let $s_x(y_i)$ be the raw evaluation score and $d_x \in \{1, -1\}$ indicate maximization or minimization, respectively, so that $q_x(y_i) = d_x s_x(y_i)$. To obtain positive values for the logarithmic transform, define $c_i = \max(s_x(y_i) + o_x, \epsilon_s)$ using a task-fixed offset o_x and floor $\epsilon_s > 0$. The supervision weights are

$$w_i = \underbrace{\frac{1}{2m}}_{\text{uniform support}} + \underbrace{\frac{1}{2} \frac{\exp z_i}{\sum_{j \in G_t} \exp z_j}}_{\text{quality weighting}}, \qquad z_i = \frac{d_x \log c_i}{\log(1+\delta)}, \tag{1}$$

where $\delta > 0$ controls sensitivity to relative differences in the shifted scores. The weights sum to one and satisfy $w_i \geq 1/(2m)$. Applying these weights to the program log likelihoods gives the self-training loss

$$\mathcal{L}_{\text{pos}}(\theta) = -\sum_{i \in G_t} w_i \ell_\theta(y_i \mid p_i). \tag{2}$$

Prompt-matched preference optimization. Self-training provides direct supervision from valid programs. FEED further leverages failed attempts by learning to prefer valid programs over failures generated under the same original prompt. Matching prompts keeps the task and parent context fixed within each comparison. For each selected valid program $i \in G_t$, let $\mathcal{N}_i$ index programs from the current evaluations $\mathcal{E}_t$ that were generated under the same prompt, $p_j = p_i$, and confirmed to fail evaluation. Valid programs with low scores are not treated as failures. We use the initial model π_{ref} as a fixed reference throughout search, with ℓ_{ref} denoting its mean-token log likelihood defined analogously to ℓ_θ. For each pair, we define a DPO-style reference-relative margin (Rafailov et al., 2023) that measures the model's log-likelihood preference for the valid program relative to this fixed reference:

$$D_{ij} = \big[\ell_\theta(y_i \mid p_i) - \ell_\theta(y_j \mid p_i)\big] - \big[\ell_{\text{ref}}(y_i \mid p_i) - \ell_{\text{ref}}(y_j \mid p_i)\big]. \tag{3}$$

Joint objective. FEED jointly learns from valid programs and their matched failure comparisons. For each pair, the preference loss $\log(1 + \exp(-\kappa D_{ij}))$, with $\kappa > 0$, decreases as the reference-relative margin increases. This objective therefore encourages a stronger preference for the valid program relative to the initial model. With objective coefficients $\alpha, \beta \geq 0$, the joint loss is

$$\mathcal{L}(\theta) = \alpha \mathcal{L}_{\text{pos}}(\theta) + \beta \sum_{i \in G_t : \mathcal{N}_i \neq \varnothing} \frac{w_i}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \log(1 + \exp(-\kappa D_{ij})). \tag{4}$$

Each valid program's quality weight w_i scales both its self-training loss and its average preference loss over matched failures. Programs without matched failures contribute only to self-training.

Controlled updates. Updates from the joint objective are accepted only if their measured KL divergence from the pre-update policy is finite and at most $\varepsilon > 0$ (Schulman et al., 2015). Otherwise, FEED retries or retains the current model. Appendix C.1 details the acceptance check, retry procedure, and distinction from the fixed preference reference.

3 Experiment

We evaluate whether FEED's joint learning from evaluation feedback leads to earlier discovery of high-quality solutions. Comparisons use matched model and resource settings and account for both search and learning costs.

3.1 Experimental Setup

Tasks and coverage. We evaluate six tasks spanning mathematical construction, heuristic design, and function optimization. Construction tasks include 32-circle packing (Berthold et al., 2026) and maximal-determinant matrix construction (MaxDet) (Orrick et al., 2003). Heuristic-design tasks comprise online bin packing (OBP) (Johnson et al., 1974) and capacitated vehicle routing with a fixed ant-colony solver (CVRP-ACO) (Dantzig & Ramser, 1959; Huang et al., 2026). Function optimization is evaluated on two autoconvolution problems, AC1 and AC2 (Schinzel & Schmidt, 2002; Martin & O'Bryant, 2009). In each task, executable evaluation determines candidate validity and solution quality.

Compared methods and evaluation. On these tasks, we compare FEED with TTT-Discover (Yuksekgonul et al., 2026). The main comparison and component ablations use Qwen3-8B (Yang et al., 2025) in a shared execution framework, with five seeds per task and a one-hour evaluation horizon for each run. Elapsed time includes initialization, generation, evaluation, learning, and update retries. Table 10 reports the time breakdown, and Appendix C.3 provides the detailed configurations.

To measure how reliably and quickly each method reaches a target quality, we report Hits and restricted mean time to threshold (RMST) at a representative score threshold for each task. Hits counts the runs that reach the threshold or better within the budget. RMST averages the time to first attainment across all five seeds, assigning the full budget to runs that do not reach the target. Specifically, $R(B) = \frac{1}{5}\sum_{i=1}^{5} \min(T_i, B)$, where T_i is the first-hit time, taken as ∞ for non-hits, and $B = 60$ minutes for the main comparison. Lower RMST indicates less time spent without a solution meeting the target. Appendix D presents RMST curves across a range of score thresholds.

3.2 Main Results

Figure 4 shows how the mean best-so-far solution quality evolves over the one-hour search, while Table 1 reports Hits and RMST at the task-specific thresholds.

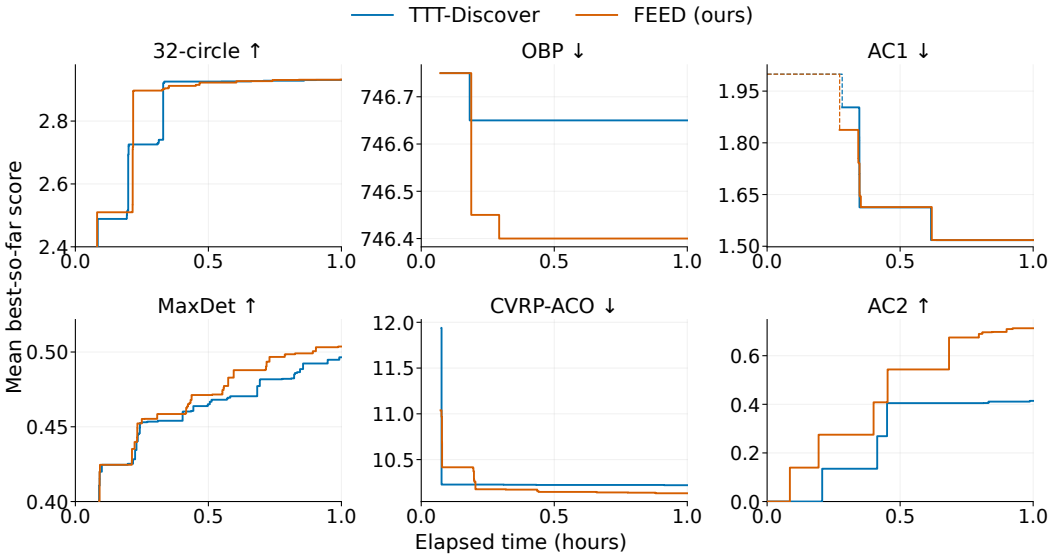

Figure 4: **Discovery trajectories within one hour.** Columns correspond to construction, heuristic design, and function optimization. Curves show five-seed mean best-so-far scores. Time includes learning. (**↓ indicates lower is better, and ↑ indicates higher is better**). Curves start once all five runs have valid results, except that initial missing results are represented by schematic dashes in AC1 and per-run zeros in AC2.

Geometric and matrix construction: 32-circle and MaxDet. On 32-circle packing, the goal is to maximize the sum of radii of 32 non-overlapping circles inside a unit square. FEED reaches high-

quality constructions earlier, although TTT-Discover later catches up and the mean trajectories end at similar scores (Figure 4). At the 2.93 threshold, FEED succeeds in all five runs versus four for TTT-Discover, reducing RMST from 45.40 to 37.38 minutes (Table 1). At one hour, FEED's five-seed mean best-so-far score is 2.93218, close to the published conventional-optimization reference of 2.93957 (Berthold et al., 2026). On MaxDet, which maximizes the determinant of a matrix with entries in $\{-1, +1\}$, FEED maintains a higher mean score through much of the latter half of the hour and finishes at 0.5037 versus 0.4965. It reaches the 0.52 threshold in one of five runs, while TTT-Discover records no hits.

Table 1: **Score-threshold attainment within one hour.** Hits reports successful runs out of five. RMST is measured in minutes (lower is better). The RMST ratio is $R_{\mathrm{TTT}}/R_{\mathrm{FEED}}$. Values above 1 favor FEED.

Task	Threshold	TTT-Discover		FEED (ours)		RMST
		Hits	RMST ↓	Hits	RMST ↓	ratio
32-circle ↑	2.93	4/5	45.40	**5/5**	**37.38**	1.21×
MaxDet ↑	0.52	0/5	60.00	**1/5**	**56.62**	1.06×
OBP ↓	746.5	1/5	**50.19**	1/5	50.26	1.00×
CVRP-ACO ↓	10.2	0/5	60.00	**5/5**	**17.62**	3.41×
AC1 ↓	1.55	5/5	20.85	5/5	**20.81**	1.00×
AC2 ↑	0.6	3/5	36.86	**5/5**	**21.79**	1.69×

Heuristic design: OBP and CVRP-ACO. On OBP, which assigns arriving items to bins while minimizing bin usage, FEED's mean bin count decreases further early in the search and remains lower thereafter, ending at 746.40 versus 746.65 for TTT-Discover (Figure 4). On CVRP-ACO, candidate programs provide heuristic information to a fixed ant-colony solver seeking short, capacity-feasible vehicle routes. Both methods improve rapidly at first, but TTT-Discover's mean trajectory then largely plateaus while FEED continues to reduce route cost. This continued improvement translates into earlier and more consistent target attainment. FEED reaches the 10.2 threshold in all five runs, whereas none of the five TTT-Discover runs reaches it within one hour. RMST decreases from 60.00 to 17.62 minutes, a $3.41\times$ reduction (Table 1).

Function optimization: AC1 and AC2. These tasks search for function constructions whose autoconvolution properties improve bounds in mathematical inequalities. On AC1, where lower scores are better, both methods improve rapidly along similar trajectories and reach the 1.55 threshold in all five runs, with RMSTs of approximately 21 minutes (Figure 4 and Table 1). FEED's five-seed mean best-so-far score reaches 1.51793 within one hour, close to the published numerical reference (Matolcsi & Vinuesa, 2010) of 1.50973 reported by Yuksekgonul et al. (2026). On AC2, where higher scores are better, FEED continues to improve and establishes a larger advantage over TTT-Discover during the latter half of the hour. Its mean best-so-far score reaches 0.7129 at one hour, compared with 0.4142 for TTT-Discover. FEED reaches the 0.6 threshold in all five runs, compared with three for TTT-Discover, while RMST decreases from 36.86 to 21.79 minutes (Table 1).

Together, these results show that FEED discovers high-quality solutions earlier on multiple tasks and approaches published reference constructions on 32-circle and AC1 within one hour. We next examine how joint supervision and parameter updates contribute to these gains, and whether the benefits extend to larger models and other model families.

3.3 Ablation and Generalization Studies

3.3.1 Learning Objectives and Search Comparisons

To examine the contribution of joint learning, we compare FEED with variants that retain only one learning objective (Table 2). We further evaluate whether FEED's efficiency gains extend to comparisons with search-only methods (Table 3). Both comparisons use a one-hour budget that includes learning costs where applicable.

Table 2: **Ablations of learning objectives under a one-hour budget.** Thresholds and metrics follow Table 1. Bold marks the highest nonzero Hits and lowest RMST per row, including ties.

Task	Threshold	Positive-only		Preference-only		FEED (ours)	
		Hits	RMST ↓	Hits	RMST ↓	Hits	RMST ↓
32-circle ↑	2.93	4/5	**34.10**	1/5	56.47	**5/5**	37.38
MaxDet ↑	0.52	0/5	60.00	0/5	60.00	**1/5**	**56.62**
OBP ↓	746.5	**3/5**	**40.57**	0/5	60.00	1/5	50.26
CVRP-ACO ↓	10.2	2/5	45.07	3/5	42.02	**5/5**	**17.62**
AC1 ↓	1.55	**5/5**	21.79	**5/5**	24.47	**5/5**	**20.81**
AC2 ↑	0.6	4/5	32.86	4/5	29.41	**5/5**	**21.79**

Learning objectives. Positive-only retains quality-weighted self-training, while Preference-only retains prompt-matched success–failure preference learning. They use $(\alpha, \beta) = (1, 0)$ and $(0, 1)$ in Equation 4, respectively, while FEED uses $(1, 1)$. Both variants retain FEED's sample-selection rules and update-control procedure. All other settings remain unchanged. The retained objective is not rescaled, and updates with no learning signal are skipped. FEED achieves lower RMST than both single-objective variants on multiple tasks (Table 2). On CVRP-ACO, FEED reaches the threshold in all five runs, compared with two for Positive-only and three for Preference-only. Its RMST is 17.62 minutes, compared with 45.07 and 42.02 minutes, respectively. These results support the value of jointly leveraging quality-weighted self-training and prompt-matched success–failure preference learning for efficient discovery.

Table 3: **Search comparisons within one hour.** PUCT keeps the model fixed under FEED's contextual search rules. Hits reports successful runs out of five. RMST is in minutes. Thresholds follow Table 1. Timing details appear in Appendix C.5. Bold marks the highest nonzero Hits and lowest RMST per row, including ties.

Task	Threshold	PUCT		OpenEvolve		FEED (ours)	
		Hits	RMST ↓	Hits	RMST ↓	Hits	RMST ↓
32-circle ↑	2.93	2/5	47.57	0/5	60.00	**5/5**	**37.38**
MaxDet ↑	0.52	0/5	60.00	0/5	60.00	**1/5**	**56.62**
OBP ↓	746.5	**2/5**	53.42	**2/5**	**46.45**	1/5	50.26
CVRP-ACO ↓	10.2	3/5	37.29	**5/5**	20.16	**5/5**	**17.62**
AC1 ↓	1.55	**5/5**	**18.05**	0/5	60.00	**5/5**	20.81
AC2 ↑	0.6	4/5	34.12	2/5	50.38	**5/5**	**21.79**

Comparison with search-only methods. To evaluate FEED's efficiency relative to search without parameter updates, we compare it with PUCT and OpenEvolve (Sharma, 2025) (Table 3). The PUCT baseline uses the same parent-selection, prompt-construction, and archive-update rules as FEED, while keeping the initial model fixed. OpenEvolve provides a comparison with a different search framework. FEED achieves lower RMST than each search-only method on multiple tasks, even after accounting for learning costs. On 32-circle, FEED reaches the threshold in all five runs, versus two for PUCT and none for OpenEvolve, with RMSTs of 37.38, 47.57, and 60.00 minutes, respectively. On AC2, FEED's RMST is 21.79 minutes, compared with 34.12 for PUCT and 50.38 for OpenEvolve. Appendix C.3 details the OpenEvolve configuration.

3.3.2 Model Size and Family

Model size. To examine discovery efficiency across model sizes, we compare FEED with TTT-Discover and OpenEvolve using Qwen3-8B, Qwen3-14B, and Qwen3-32B (Yang et al., 2025) on 32-circle over a 180-minute horizon. FEED reaches the 2.93 threshold in all five runs at each scale, reducing RMST relative to TTT-Discover by 34.7%, 9.8%, and 17.6%, respectively (Table 4). OpenEvolve reaches the threshold in one, zero, and two runs at the three respective scales, with higher RMST than FEED throughout. Across the evaluated configurations, discovery time does not decrease monotonically with model size. FEED reaches the target fastest with 8B, while 32B im-

proves over 14B but remains slower than 8B. This pattern highlights the importance of considering search and learning costs alongside model capability when assessing discovery efficiency. Experimental configurations are provided in Appendix C.6.

Table 4: **32-circle attainment at 2.93 within a 180-minute horizon.** Hits counts runs out of five. RMST is in minutes (lower is better). Bold marks the best result within each model, including ties.

Model	OpenEvolve		TTT-Discover		FEED (ours)	
	Hits	RMST ↓	Hits	RMST ↓	Hits	RMST ↓
Qwen3-8B	1/5	158.37	**5/5**	57.23	**5/5**	**37.38**
Qwen3-14B	0/5	180.00	4/5	102.57	**5/5**	**92.51**
Qwen3-32B	2/5	161.16	4/5	85.12	**5/5**	**70.15**

Model family. Beyond model scale, we examine whether FEED's benefits extend to other model families using Ministral-3-8B-Reasoning-2512 (Liu et al., 2026a) and OLMo-3-7B-Think (Team Olmo, 2025). On 32-circle, FEED achieves higher mean best-so-far scores across five seeds at 180 minutes than TTT-Discover, with scores of 2.7148 versus 2.4196 with Ministral, and 2.8119 versus 2.7702 with OLMo (Figure 5). OpenEvolve achieves mean scores of 1.4537 with Ministral and 2.6667 with OLMo at 180 minutes. These results show that FEED's solution-quality gains extend beyond the Qwen3 family.

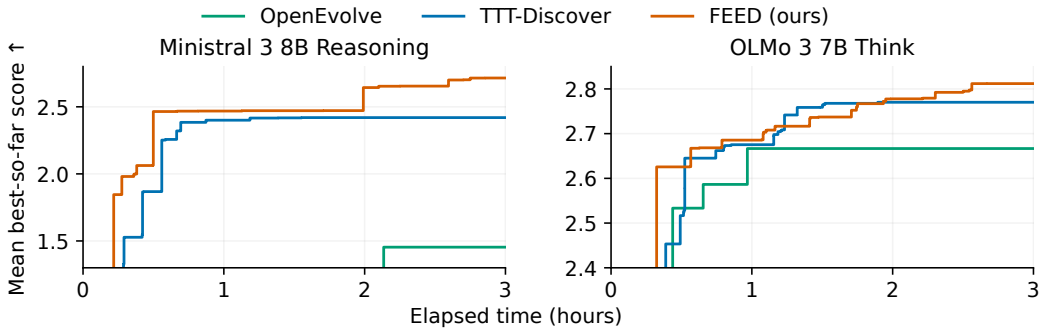

Figure 5: **32-circle on two additional model families.** Curves show mean best-so-far scores over five seeds (higher is better), with independent vertical scales. Each run contributes zero before its first valid solution. Portions below the displayed ranges are omitted. OpenEvolve on Ministral has valid solutions in three runs. Timing follows Appendix C.6.

4 Conclusion and Future Work

We introduced FEED, which leverages quality-weighted self-training and prompt-matched success–failure preference learning to improve test-time discovery efficiency. Under matched one-hour budgets including search and learning costs, FEED discovers high-quality solutions earlier than TTT-Discover on multiple tasks. Ablations support the contributions of joint supervision and parameter updates, while additional evaluations show efficiency gains across Qwen3 model sizes and quality gains with other model families. Future work will explore how to adapt update timing, learning effort, and the balance between the two objectives to search progress.

AI use statement

In this work, we used generative AI tools for manuscript polishing, literature retrieval, and assistance with adapting our code to the computing environment used for the experiments. The authors independently reviewed and verified all AI-assisted content. We take full responsibility for the final content of this paper.

Ethics statement

This work studies automated program discovery on mathematical construction and combinatorial optimization tasks and does not involve human-subject experiments or personal data. The authors are responsible for the appropriate use and attribution of the models, benchmarks, and software underlying this work.

Reproducibility statement

Section 2 specifies FEED's learning objectives and update procedure. Section 3 defines the comparison settings, score-threshold metrics, and ablation variants. Appendix C provides Algorithm 1 and details of search, evaluation, model updates, and resource and time accounting. Together, these details specify the experimental protocol and the aggregation of the reported results. The accompanying code package covers the Qwen3-8B comparisons among FEED, PUCT, TTT-Discover, and OpenEvolve on the six main-evaluation tasks.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Timo Berthold, Dominik Kamp, Gioni Mexi, Sebastian Pokutta, and Imre Polik. Out-of-the-box global optimization for packing problems: New models and improved solutions. *arXiv preprint arXiv:2605.04850*, 2026. URL https://arxiv.org/abs/2605.04850.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787.

Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Learning from Natural Language Feedback. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=xo3hI5MwvU.

Hao Chen, Guanxi Lu, Yasuyuki Okoshi, Zhiwen Mo, Masato Motomura, and Hongxiang Fan. Rethinking Optimal Verification Granularity for Compute-Efficient Test-Time Scaling. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 88650–88674. Curran Associates, Inc., 2025. doi: 10.52202/085713-2964. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8011b23e1dc3f57e1b6211ccad498919-Abstract-Conference.html.

Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. Teaching Large Language Models to Self-Debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2460396f2d0d421885997dd1612ac56b-Abstract-Conference.html.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.6.1.80.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. HSEvo: Elevating Automatic Heuristic Design with Diversity-Driven Harmony Search and Genetic Algorithm Using LLMs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):26931–26938, 2025. doi: 10.1609/aaai.v39i25.34898. URL https://doi.org/10.1609/aaai.v39i25.34898.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. RAFT: Reward rAnked FineTuning for Generative Foundation Model Alignment. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=m7p5O7zblY.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-ReST: Reflection-Reinforced Self-Training for Language Agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15394–15411. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.861. URL https://aclanthology.org/2024.emnlp-main.861/.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling Test-Time Compute for Software Engineering. *arXiv preprint arXiv:2501.14723*, 2025. URL https://arxiv.org/abs/2501.14723.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Model Alignment as Prospect Theoretic Optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12634–12651. PMLR, 2024. URL https://proceedings.mlr.press/v235/ethayarajh24a.html.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. doi: 10.1038/s41586-022-05172-4. URL https://www.nature.com/articles/s41586-022-05172-4.

Leonidas Gee, Milan Gritta, Gerasimos Lampouras, and Ignacio Iacobacci. Code-Optimise: Self-Generated Preference Data for Correctness and Efficiency. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 79–94. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.5. URL https://aclanthology.org/2025.findings-naacl.5/.

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.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced Self-Training (ReST) for Language Modeling. *arXiv preprint arXiv:2308.08998*, 2023. URL https://arxiv.org/abs/2308.08998.

Akshat Gupta, Jermaine Lei, Alexander Lu, Gopala Anumanchipalli, and Leshem Choshen. Automated Discovery Has No Universally Superior Harness. *arXiv preprint arXiv:2607.18235*, 2026. URL https://arxiv.org/abs/2607.18235.

Michael Hassid, Tal Remez, Jonas Gehring, Roy Schwartz, and Yossi Adi. The Larger the Better? Improved LLM Code-Generation via Budget Reallocation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=QJvfpWSpWm.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11170–11189, 2024. doi: 10.18653/v1/2024.emnlp-main.626. URL https://aclanthology.org/2024.emnlp-main.626/.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STaR: Training Verifiers for Self-Taught Reasoners. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=stmqBSW2dV.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of Algorithms and Language Model for Automatic Heuristic Design. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/75ed738c083f8941cfc990245e4f6265-Abstract-Conference.html.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement Learning via Self-Distillation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QkfkxyRizZ.

Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-Turn Code Generation Through Single-Step Rewards. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 26700–26716. PMLR, 2025. URL https://proceedings.mlr.press/v267/jain25a.html.

Yizhu Jiao, Ruixiang Zhang, Richard Bai, Jiawei Han, Ronan Collobert, and Yizhe Zhang. Primal Generation, Dual Judgment: Self-Training from Test-Time Scaling. *arXiv preprint arXiv:2605.11299*, 2026. URL https://arxiv.org/abs/2605.11299.

D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974. doi: 10.1137/0203025. URL https://epubs.siam.org/doi/10.1137/0203025.

Kezhao Lai, Yutao Lai, and Hai-Lin Liu. Beyond the Node: Clade-level Selection for Efficient MCTS in Automatic Heuristic Design. *arXiv preprint arXiv:2602.00549*, 2026. URL https://arxiv.org/abs/2602.00549.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards Open-Ended and Sample-Efficient Program Evolution. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7886b9bafe76c52fd568db10ff9772df-Abstract-Conference.html.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328. Curran Associates, Inc., 2022. doi: 10.52202/068431-1549. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html.

Young-Jun Lee, Seungone Kim, Minki Kang, Alistair Cheong Liang Chuen, Zerui Chen, Seungho Han, Taehee Jung, and Dongyeop Kang. Evolution Fine-Tuning: Learning to Discover Across 371 Optimization Tasks. *arXiv preprint arXiv:2606.29082*, 2026. URL https://arxiv.org/abs/2606.29082.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.865. URL https://aclanthology.org/2025.findings-emnlp.865/.

Zun Li, John Schultz, Daniel Hennes, and Marc Lanctot. Discovering multiagent learning algorithms with large language models. *arXiv preprint arXiv:2602.16928*, 2026. URL https://arxiv.org/abs/2602.16928.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, Alexandre Sablayrolles, Amélie Héliou, Amos You, Andy Ehrenberg, Andy Lo, Anton Eliseev, Antonia Calvi, Avinash Sooriyarachchi, Baptiste Bout, Baptiste Rozière, Baudouin De Monicault, Clémence Lanfranchi, Corentin Barreau, Cyprien Courtot, Daniele Grattarola, Darius Dabert, Diego de las Casas, Elliot Chane-Sane, Faruk Ahmed, Gabrielle Berrada, Gaëtan Ecrepont, Gauthier Guinet, Georgii Novikov, Guillaume Kunsch, Guillaume Lample, Guillaume Martin, Gunshi Gupta, Jan Ludziejewski, Jason Rute, Joachim Studnia, Jonas Amar, Joséphine Delas, Josselin Somerville Roberts, Karmesh Yadav, Khyathi Chandu, Kush Jain, Laurence Aitchison, Laurent Fainsin, Léonard Blier, Lingxiao Zhao, Louis Martin, Lucile Saulnier, Luyu Gao, Maarten Buyl, Margaret Jennings, Marie Pellat, Mark Prins, Mathieu Poirée, Mathilde Guillaumin, Matthieu Dinot, Matthieu Futeral, Maxime Darrin, Maximilian Augustin, Mia Chiquier, Michel Schimpf, Nathan Grinsztajn, Neha Gupta, Nikhil Raghuraman, Olivier Bousquet, Olivier Duchenne, Patricia Wang, Patrick von Platen, Paul Jacob, Paul Wambergue, Paula Kurylowicz, Pavankumar Reddy Muddireddy, Philomène Chagniot, Pierre Stock, Pravesh Agrawal, Quentin Torroba, Romain Sauvestre, Roman Soletskyi, Rupert Menneer, Sagar Vaze, Samuel Barry, Sanchit Gandhi, Siddhant Waghjale, Siddharth Gandhi, Soham Ghosh, Srijan Mishra, Sumukh Aithal, Szymon Antoniak, Teven Le Scao, Théo Cachet, Theo Simon Sorg, Thibaut Lavril, Thiziri Nait Saada, Thomas Chabal, Thomas Foubert, Thomas Robert, Thomas Wang, Tim Lawson, Tom Bewley, Tom Edwards, Umar Jamil, Umberto Tomasini, Valeriia Nemychnikova, Van Phung, Vincent Maladière, Virgile Richard, Wassim Bouaziz, Wen-Ding Li, William Marshall, Xinghui Li, Xinyu Yang, Yassine El Ouahidi, Yihan Wang, Yunhao Tang, and Zaccharie Ramzi. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026a. URL https://arxiv.org/abs/2601.08584.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024a. URL https://proceedings.mlr.press/v235/liu24bs.html.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. RLTF: Reinforcement Learning from Unit Test Feedback. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=hjYmsV6nXZ.

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. *arXiv preprint arXiv:2602.23413*, 2026b. URL https://arxiv.org/abs/2602.23413.

Zhihan Liu, Miao Lu, Shenao Zhang, Boyi Liu, Hongyi Guo, Yingxiang Yang, Jose Blanchet, and Zhaoran Wang. Provably Mitigating Overoptimization in RLHF: Your SFT Loss is Implicitly an Adversarial Regularizer. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/fa69e968b7319fd42524febd41475fb3-Abstract-Conference.html.

Haoze Lv, Ning Lu, Ziang Zhou, Yew-Soon Ong, and Shengcai Liu. AHD Agent: Agentic Reinforcement Learning for Automatic Heuristic Design. *arXiv preprint arXiv:2605.08756*, 2026. URL https://arxiv.org/abs/2605.08756.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, and anima anandkumar. Eureka: Human-Level Reward Design via Coding Large Language Models. In *International Conference on Learning Representations*, volume 2024, pp. 26516–26560, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/70c26937fbf3d4600b69a129031b66ec-Abstract-Conference.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine:

Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Greg Martin and Kevin O'Bryant. The supremum of autoconvolutions, with applications to additive number theory. *Illinois Journal of Mathematics*, 53(1):219–235, 2009. doi: 10.1215/ijm/1264170847. URL https://doi.org/10.1215/ijm/1264170847.

Máté Matolcsi and Carlos Vinuesa. Improved bounds on the supremum of autoconvolutions. *Journal of Mathematical Analysis and Applications*, 372(2):439–447.e2, 2010. doi: 10.1016/j.jmaa.2010.07.030. URL https://math.bme.hu/~matolcsi/sch0914.pdf.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple Preference Optimization with a Reference-Free Reward. In *Advances in Neural Information Processing Systems*, volume 37, pp. 124198–124235. Curran Associates, Inc., 2024. doi: 10.52202/079017-3946. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e099c1c9699814af0be873a175361713-Abstract-Conference.html.

Sora Miyamoto, Daisuke Oba, and Naoaki Okazaki. Aligning Tree-Search Policies with Fixed Token Budgets in Test-Time Scaling of LLMs. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=TgaVuBX8dx.

Ansh Nagda, Prabhakar Raghavan, and Abhradeep Thakurta. Reinforced generation of combinatorial structures: Ramsey numbers. *arXiv preprint arXiv:2603.09172*, 2026. URL https://arxiv.org/abs/2603.09172.

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. URL https://arxiv.org/abs/2506.13131v1.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-Imitation Learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 3878–3887. PMLR, 2018. URL https://proceedings.mlr.press/v80/oh18b.html.

William P. Orrick, Bruce Solomon, Roland Dowdeswell, and Warren D. Smith. New lower bounds for the maximal determinant problem, 2003. URL https://arxiv.org/abs/math/0304410v1.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative Reasoning Preference Optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 116617–116637. Curran Associates, Inc., 2024. doi: 10.52202/079017-3702. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d37c9ad425fe5b65304d500c6edcba00-Abstract-Conference.html.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *arXiv preprint arXiv:1910.00177*, 2019. URL https://arxiv.org/abs/1910.00177v3.

Peter Phan, Dhruv Agarwal, Kavitha Srinivas, Horst Samulowitz, Pavan Kapanipathi, and Andrew McCallum. MiGrATe: Mixed-Policy GRPO for Adaptation at Test-Time. *arXiv preprint arXiv:2508.08641*, 2025. URL https://arxiv.org/abs/2508.08641v1.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Esteban Real, Chen Liang, David So, and Quoc Le. AutoML-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8007–8019. PMLR, 2020. URL https://proceedings.mlr.press/v119/real20a.html.

Kainat Riaz, Muhammad Ahmed Mohsin, Ahsan Bilal, Muhammad Umer, Ayesha Mohsin, Aqib Riaz, Ali Subhan, and John M. Cioffi. Epistemic Uncertainty for Test-Time Discovery. *arXiv preprint arXiv:2605.11328*, 2026. URL https://arxiv.org/abs/2605.11328.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering. *arXiv preprint arXiv:2401.08500*, 2024. URL https://arxiv.org/abs/2401.08500.

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*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

A. Schinzel and W. M. Schmidt. Comparison of L^1- and L^∞-norms of squares of polynomials. *Acta Arithmetica*, 104(3):283–296, 2002. doi: 10.4064/aa104-3-4. URL https://www.impan.pl/shop/en/publication/transaction/download/product/82617.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 1889–1897. PMLR, 2015. URL https://proceedings.mlr.press/v37/schulman15.html.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan Reddy. LLM-SR: Scientific Equation Discovery via Programming with Large Language Models. In *International Conference on Learning Representations*, volume 2025, pp. 16054–16085, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/28df8e730c054c5331855fd4d5403ba9-Abstract-Conference.html.

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://www.nature.com/articles/nature24270.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lNAyUngGFK.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by Distilling Context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. Preference Ranking Optimization for Human Alignment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):18990–18998, 2024. doi: 10.1609/aaai.v38i17.29865. URL https://doi.org/10.1609/aaai.v38i17.29865.

Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm Discovery With LLMs: Evolutionary Search Meets Reinforcement Learning. *arXiv preprint arXiv:2504.05108*, 2025. URL https://arxiv.org/abs/2504.05108v4.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference Fine-Tuning of LLMs Should Leverage Suboptimal, On-Policy Data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47441–47474. PMLR, 2024. URL https://proceedings.mlr.press/v235/tajwar24a.html.

Team Olmo. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025. URL https://arxiv.org/abs/2512.13961.

Niki van Stein and Thomas Bäck. LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2):331–345, 2025. doi: 10.1109/TEVC.2024.3497793. URL https://doi.org/10.1109/TEVC.2024.3497793.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Every Rollout Counts: Optimal Resource Allocation for Efficient Test-Time Scaling. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 102312–102338. Curran Associates, Inc., 2025. doi: 10.52202/085713-3422. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/940b66bcdb34fb9ae47d85de838c372d-Abstract-Conference.html.

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. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=XHP568OuIl.

Jie Wu, Haoling Li, Xin Zhang, Xiao Liu, Yangyu Huang, Jianwen Luo, Yizhen Zhang, Zuchao Li, Ruihang Chu, Yujiu Yang, and Scarlett Li. Teaching Your Models to Understand Code via Focal Preference Alignment. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14003–14023. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.emnlp-main.707. URL https://aclanthology.org/2025.emnlp-main.707/.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for LLM Problem-Solving. In *International Conference on Learning Representations*, volume 2025, pp. 55824–55845, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/8c3caae2f725c8e2a55ecd600563d172-Abstract-Conference.html.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Weili Wang, Ed H. Chi, Shivaram Venkataraman, Wang-Cheng Kang, Derek Zhiyuan Cheng, and Beidou Wang. PACEvolve++: Improving Test-time Learning for Evolutionary Search Agents, 2026a. URL https://arxiv.org/abs/2605.07039v1.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, Chi Wang, Ed H. Chi, Fernando Pereira, Wang-Cheng Kang, Derek Zhiyuan Cheng, and Beidou Wang. PACEvolve: Enabling Progress-Aware Consistent Evolution. *arXiv preprint arXiv:2601.10657*, 2026b. URL https://arxiv.org/abs/2601.10657.

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. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388v1.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3339f19c5fcee3ad74502947a32be9e6-Abstract-Conference.html.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/4ced59d480e07d290b6f29fc8798f195-Abstract-Conference.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://www.nature.com/articles/s41586-025-08661-4.

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. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=96zNuQrH9Y.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64735–64772. Curran Associates, Inc., 2024a. doi: 10.52202/079017-2066. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/76ec4dc30e9faaf0e4b6093eaa377218-Abstract-Conference.html.

Dylan Zhang, Shizhe Diao, Xueyan Zou, and Hao Peng. PLUM: Improving Code LMs with Execution-Guided On-Policy Preference Learning Driven By Synthetic Test Cases. *arXiv preprint arXiv:2406.06887*, 2024b. URL https://arxiv.org/abs/2406.06887.

Kechi Zhang, Ge Li, Yihong Dong, Jingjing Xu, Jun Zhang, Jing Su, Yongfei Liu, and Zhi Jin. CodeDPO: Aligning code models with self generated and verified source code. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15854–15871. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.771. URL https://aclanthology.org/2025.acl-long.771/.

Yansen Zhang, Yilu Liu, Tianyu Liu, Jiamin Chen, Xiaokun Zhang, Kai Xie, Xue Liu, Yiyan Qi, and Chen Ma. Budget-Aware LLM Discovery via Cost-Calibrated Frontier Utility. *arXiv preprint arXiv:2607.26828*, 2026. URL https://arxiv.org/abs/2607.26828v3.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 78338–78373. PMLR, 2025. URL https://proceedings.mlr.press/v267/zheng25o.html.

Zhi Zheng, Rongsheng Chen, Yunpeng Ba, Zhenkun Wang, Yee Whye Teh, and Wee Sun Lee. Agentic ESOpt: Fine-Tuning Long-Horizon LLM Agents with Minimal GPU Requirements. *arXiv preprint arXiv:2608.17310*, 2026. URL https://arxiv.org/abs/2608.17310.

A Related Work

Scientific discovery through executable programs. Advances in artificial intelligence have opened new possibilities for scientific discovery, including the search for mathematical constructions, algorithms, and scientific models. AutoML-Zero (Real et al., 2020) and AlphaTensor (Fawzi et al., 2022) demonstrate that computational search can discover learning algorithms and matrix multiplication procedures. Language-model-based methods extend this direction through iterative program generation and evaluation. FunSearch (Romera-Paredes et al., 2024) uses a database of successful programs to guide subsequent generation, while AlphaEvolve (Novikov et al., 2025) develops evolutionary program search across mathematical and algorithmic applications. Further work explores mathematical discovery across diverse problems (Georgiev et al., 2025), heuristic and metaheuristic design in EoH (Liu et al., 2024a) and LLaMEA (van Stein & Bäck, 2025), equation discovery in LLM-SR (Shojaee et al., 2025), and reward design in Eureka (Ma et al., 2024). Across these applications, evaluated programs and their outcomes guide the selection and revision of subsequent candidates. Test-time learning extends this use of search experience to parameter updates, adapting the model that proposes those candidates.

Learning from discovery experience. Existing methods improve discovery through learning from evaluated candidates. During test-time discovery, TTT-Discover (Yuksekgonul et al., 2026) uses an entropic objective to emphasize high-reward outcomes, while CALM (Huang et al., 2026) integrates reinforcement learning into heuristic evolution and ThetaEvolve (Wang et al., 2026) incorporates validity and task performance into learning for open problems. These updates can draw on both current evaluations and accumulated search experience. EvoTune (Surina et al., 2025) constructs same-prompt preference pairs, including valid–invalid comparisons, and reuses experience across search stages. MiGrATe (Phan et al., 2025) incorporates current-policy samples and historical high-quality candidates into mixed-policy GRPO updates. SDPO (Hübotter et al., 2026) uses a feedback-conditioned self-teacher to provide distributional supervision. Other work investigates how adaptation is carried out. UG-TTT (Riaz et al., 2026) explores uncertainty-guided exploration, Agentic ESOpt (Zheng et al., 2026) uses evolution strategies for parameter adaptation, and PACEvolve++ (Yan et al., 2026a) updates an advisor whose proposals are implemented by a separate model. Discovery experience also supports training before deployment on new tasks. Evolution Fine-Tuning (Lee et al., 2026) learns from evolutionary trajectories across tasks, while AHD Agent (Lv et al., 2026) trains transferable agents for heuristic design. These approaches use discovery experience either to adapt models during search or to develop capabilities that transfer to subsequent tasks. Yet acquiring that experience remains costly. Each new candidate requires generation and evaluation, so expanding the learning material entails further search expenditure.

Discovery efficiency and evaluation budgets. Improving solution quality through additional search requires balancing performance gains against computational cost. Large Language Monkeys (Brown et al., 2024) demonstrates that repeated sampling expands solution coverage, establishing additional inference computation as a means of improving success rates. Under fixed budgets, however, studies of compute-optimal inference (Snell et al., 2025; Wu et al., 2025b) show that the gains vary with search strategy, problem difficulty, and model size. Budget reallocation in code generation (Hassid et al., 2024) shows that repeated sampling from smaller models can outperform larger models under matched budgets in specific settings. These findings shift the emphasis from increasing search effort to using that effort effectively through the choice of generation and search procedures.

Search methods make use of each evaluation by allowing its outcome to guide subsequent attempts. In code generation, S* (Li et al., 2025), CodeMonkeys (Ehrlich et al., 2025), and AlphaCodium (Ridnik et al., 2024) incorporate execution feedback into candidate refinement and selection. This use of feedback extends from revising individual candidates to guiding the generation of new ones. ReEvo (Ye et al., 2024) derives reflections from heuristic comparisons, while OPRO (Yang et al., 2024) and GEPA (Agrawal et al., 2026) guide optimization through evaluated proposals and reflective feedback. As candidates accumulate, their evaluation also informs which search directions receive further exploration. MCTS-AHD (Zheng et al., 2025) and Clade-AHD (Lai et al., 2026) investigate tree-based exploration, while HSEvo (Dat et al., 2025) emphasizes diversity in heuristic evolution. ShinkaEvolve (Lange et al., 2026) targets sample-efficient program evolution through parent selection, novelty filtering, and model selection, while PACEvolve (Yan et al., 2026b) and EvoX (Liu et al., 2026b) adapt context or search strategies as discovery progresses. Across these

approaches, past evaluations guide the choice of subsequent search directions. Pursuing those directions requires further generation and evaluation, making it important to allocate the available search budget effectively.

Budget-aware search considers not only which directions are promising, but also whether pursuing them justifies the computational cost. In program discovery, CostAda (Zhang et al., 2026) evaluates frontier progress relative to its realized token cost and the remaining budget. Related reasoning methods address how budget constraints shape exploration. DORA (Wang et al., 2025) allocates a fixed rollout budget across reasoning directions, while BG-MCTS (Miyamoto et al., 2026) shifts from broad exploration toward refinement and answer completion as the budget decreases. Allocation can also operate across search frameworks, using early progress to prune weaker runs and redirect computation to stronger ones (Gupta et al., 2026). Assessment also consumes resources, leading VG-Search (Chen et al., 2025) to balance verification guidance against its cost by adjusting verification frequency. These methods improve how search expenditure produces useful outcomes. Learning from those outcomes can extend the value of that expenditure through the selection, weighting, and reuse of evaluated candidates.

Learning from evaluated experience. Learning from search experience is particularly important when new improvements become difficult to obtain. ShinkaEvolve (Lange et al., 2026) reports early plateaus in its hill-climbing ablation, while PACEvolve (Yan et al., 2026b) identifies repeated exploration of flawed hypotheses and convergence to suboptimal solutions. Previously evaluated candidates remain available for learning when progress stalls. Self-Imitation Learning (Oh et al., 2018) reuses past successful behavior, while Advantage-Weighted Regression (Peng et al., 2019) weights policy fitting by advantage using off-policy experience. In language models, evaluation guides the selection of self-generated training targets. RAFT (Dong et al., 2023) selects reward-ranked responses for fine-tuning, while STaR (Zelikman et al., 2022) and ReST-EM (Singh et al., 2024) use correctness to select examples for iterative self-training. The supply of useful examples can also be extended through reuse and refinement. ReST (Gulcehre et al., 2023) reuses generated data across successive improvement steps, while Re-ReST (Dou et al., 2024) and ReST-MCTS* (Zhang et al., 2024a) use reflection and process-reward-guided search, respectively, to improve training examples. These approaches develop positive supervision from generated experience. V-STaR (Hosseini et al., 2024) also uses incorrect responses to train a verifier, showing that unsuccessful attempts can support learning beyond fitting successful outputs.

Learning from evaluation feedback. Execution-feedback methods exploit unsuccessful attempts through reinforcement learning, revision, and preference optimization. CodeRL (Le et al., 2022) and RLTF (Liu et al., 2023) use execution-related feedback for reinforcement learning, while Imitation Learning from Language Feedback (ILF) (Chen et al., 2024a) trains models on code refined using natural language feedback. Evaluating multiple candidates also enables comparative supervision. PLUM (Zhang et al., 2024b) constructs execution-guided preferences from model-generated programs, while Code-Optimise (Gee et al., 2025) and CodeDPO (Zhang et al., 2025) incorporate correctness and execution efficiency into preference construction. Rather than comparing whole programs uniformly, Target-DPO (Wu et al., 2025a) focuses preference supervision on differences identified through code revisions. Candidate evaluation can further train the selection process itself. μCode (Jain et al., 2025) learns from a local expert in multi-turn code generation, while DuST (Jiao et al., 2026) uses successful and unsuccessful candidates to improve judgment. These methods show that execution outcomes can supervise both generation and candidate assessment. Their use in learning involves not only identifying successful and unsuccessful outcomes, but also expressing the information those outcomes provide.

Feedback can guide learning through explanations of an outcome, rather than only its score or relative ranking. Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), and Self-Debugging (Chen et al., 2024b) use reflections or revisions to improve subsequent outputs, while TextGrad (Yuksekgonul et al., 2025) uses textual feedback to guide optimization. Distillation carries the benefits of feedback-informed generation into subsequent model updates. Context distillation (Snell et al., 2022) trains models to retain capabilities elicited by additional context, while on-policy distillation (Agarwal et al., 2024) provides teacher supervision on student-generated sequences. SDPO (Hübotter et al., 2026) brings evaluation feedback into this process through a feedback-conditioned

self-teacher. These studies show how information beyond an evaluation score can guide subsequent generation through textual feedback or inform parameter updates through distillation.

Joint supervision for discovery. Direct supervision and preference learning provide complementary ways to update a model from evaluated outputs. DPO (Rafailov et al., 2023) uses reference-relative likelihoods to optimize chosen–rejected comparisons, while SimPO (Meng et al., 2024), KTO (Ethayarajh et al., 2024), and Preference Ranking Optimization (Song et al., 2024) explore alternative objectives or supervision structures. Research on preference fine-tuning (Tajwar et al., 2024) further examines the benefits of on-policy sampling and updates that suppress disfavored responses. Such updates can be paired with direct supervision on preferred outputs. ORPO (Hong et al., 2024) integrates positive likelihood training with an odds-ratio preference objective, and RPO (Liu et al., 2024b) studies supervised regularization alongside preference optimization. For reasoning, Iterative Reasoning Preference Optimization (Pang et al., 2024) jointly optimizes the likelihood of correct generated responses and a DPO-style preference term.

In program discovery, EvoTune (Surina et al., 2025) evaluates preference optimization and self-training as alternative updates and identifies their combination as a direction for future work. FEED investigates joint supervision in this setting through a shared quality-weighting scheme. Selected current and historical valid programs receive quality-weighted self-training, and the same weights scale their prompt-matched success–failure preference terms. Programs without matched failures retain their direct supervision. We evaluate whether this use of accumulated experience improves discovery efficiency under budgets that include both search and learning, through comparisons with single-objective variants and search-only methods.

B Limitations

Our findings are limited to the evaluated tasks, models, and time budgets. The six-task evaluation uses Qwen3-8B, while additional model sizes and families are tested only on 32-circle. Benefits vary across backbones and quality thresholds, leaving broader task–model coverage and longer-horizon evaluation for future work. Despite these limitations, our results demonstrate that joint self-training and preference learning can accelerate high-quality discovery on multiple tasks within the evaluated budgets.

C Implementation Details

C.1 Update-control details

Let $\mathcal{C}_t$ contain the contexts at sampled completion positions, each comprising the original prompt and preceding completion tokens. For pre-update parameters θ and proposed parameters $\widetilde{\theta}$, accept the update only if the mean full-vocabulary forward KL over contexts $h \in \mathcal{C}_t$ is finite and satisfies

$$\widehat{D}(\theta, \widetilde{\theta}) = \frac{1}{|\mathcal{C}_t|} \sum_{h \in \mathcal{C}_t} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid h) \,\|\, \pi_{\widetilde{\theta}}(\cdot \mid h)\big) \le \varepsilon, \tag{5}$$

Otherwise restart from the same pre-update parameters at the next learning rate in Table 6. Retain the previous parameters if all attempts fail. The KL check compares the proposed policy with the pre-update policy, whereas preference learning uses the fixed initial policy as its reference. This sampled check constrains policy movement but does not guarantee improved solution quality.

C.2 Parent-selection details

Parent selection uses both evaluated child performance and an exploration bonus that favors higher-ranked, less-visited lineages. For an archived parent $a \in \mathcal{H}_t^-$, let $V_t(a)$ be its best evaluated direct-child value, or its own value if no child value is available. Higher values indicate better solutions. Let $r_t(a)$ be its zero-based rank by descending own value, $n_t(a)$ its lineage visit count, and N_t the total counted rollouts. With exploration coefficient $c_{\exp} > 0$ and value scale Δ_t, the selected parent

is

$$a_t \in \arg\max_{a \in \mathcal{H}_t^-} \left[V_t(a) + c_{\exp} \Delta_t \rho_t(a) \frac{\sqrt{1+N_t}}{1+n_t(a)} \right], \quad \rho_t(a) = \frac{|\mathcal{H}_t^-| - r_t(a)}{\sum_{a' \in \mathcal{H}_t^-} (|\mathcal{H}_t^-| - r_t(a'))}, \tag{6}$$

where Δ_t is the range of values among non-seed archive entries, with a lower bound of 10^{-6}. If no non-seed entries exist, all archive entries are used. Counted rollouts, including failed evaluations, increment the visit counts of the selected parent and its recorded ancestors. In contrast, evaluated child values update only the direct parent's value.

Search-and-learning loop. Algorithm 1 summarizes how candidate generation, evaluation, and model updates interact across search rounds. Let b denote the number of candidates generated per round, $\mathcal{S}_t$ contain parent values and visit counts, $R_{\max}$ denote the maximum number of rounds, and $\mathcal{T}$ collect evaluation records across rounds. The inputs θ_0, $\mathcal{H}_0^-$, and $\mathcal{S}_0$ specify the initial model parameters, search archive, and search state. The matched-failure sets are denoted by $\mathcal{F}_t = \{\mathcal{N}_i\}_{i \in G_t}$.

Algorithm 1: FEED

Input: $x, \theta_0, \mathcal{H}_0^-, \mathcal{S}_0, b, B, R_{\max}$; fixed learning settings.
Output: Best valid program evaluated within B, if any, and trace $\mathcal{T}$.

$\mathcal{T} \leftarrow \varnothing$; $\mathcal{P}_0 \leftarrow \varnothing$
for $t = 0, \ldots, R_{\max} - 1$ while $\text{Elapsed}() < B$ **do**
  $(p_{t,i})_{i=1}^{b} \leftarrow \text{BuildPrompts}(x, \mathcal{H}_t^-, \mathcal{S}_t, b)$
  $y_{t,i} \sim \pi_{\theta_t}(\cdot \mid p_{t,i})$, $i = 1, \ldots, b$
  $\mathcal{E}_t \leftarrow \text{Evaluate}(x, \{(p_{t,i}, y_{t,i})\}_{i=1}^{b})$; $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{E}_t$
  $(\mathcal{H}_{t+1}^-, \mathcal{S}_{t+1}) \leftarrow \text{UpdateSearch}(\mathcal{H}_t^-, \mathcal{S}_t, \mathcal{E}_t)$
  $\theta_{t+1} \leftarrow \theta_t$
  if $t + 1 < R_{\max}$ and $\text{Elapsed}() < B$ **then**
    $G_t \leftarrow \text{SelectPositives}(\mathcal{E}_t, \mathcal{P}_t)$
    if $G_t \neq \varnothing$ **then**
      $\mathcal{F}_t \leftarrow \text{MatchFailures}(G_t, \mathcal{E}_t)$
      $w_i \leftarrow$ Equation 1, $i \in G_t$; $\mathcal{L} \leftarrow$ Equation 4
      $\theta_{t+1} \leftarrow \text{ControlledUpdate}(\theta_t, \mathcal{L}, \varepsilon)$
  $\mathcal{P}_{t+1} \leftarrow \text{UpdateLearningPool}(\mathcal{P}_t, \mathcal{E}_t)$
return $\text{BestValid}(\mathcal{T}, B)$, $\mathcal{T}$

C.3 Evaluation settings

Table 5: **Shared model, resources, and discovery budget.** Search histories and selected parents may differ.

Setting	Value
Run seeds	30901–30905
Initialization	Pretrained Qwen3-8B, no inherited task-specific adapter
Maximum search rounds	512
Update schedule	After each eligible batch for methods with parameter updates
Reported comparison	First hour of every run
Resources	1 H100, 20 CPU slots, up to 16 evaluator workers per run
Token limits	16,384 total context, 8,192 for prompt plus first-stage output, 7,168 parent-prompt cap with matched truncation

Learning-material selection. The historical pool $\mathcal{P}_t$ starts empty and accumulates eligible valid programs after each batch. It is deduplicated by stripped program text independently of the search archive. Both current and historical pools are ranked by raw score in the task's improvement direction. Selection first takes up to two historical programs and then up to two current programs. After deduplication across sources, any remaining slots are filled from the current pool before the

historical pool, yielding at most four programs. Nonfinite or negative shifted scores $s_x(y_i) + o_x$ are ineligible. Eligible scores are floored at ϵ_s before taking logarithms.

Rejected candidates must have confirmed evaluation failures and model-generated completion tokens available for loss computation. Programs whose evaluation outcomes are unknown because of infrastructure issues are excluded, as are valid programs with low scores. After deduplication, the first four invalid programs in stored batch order are selected, then filtered to those matching a selected positive's original generation prompt. Each retained failure is paired with every positive sharing that prompt. For both self-training and preference learning, losses cover model-generated completion tokens, including reasoning and code. Prompt and forcibly inserted tokens are masked out.

For each positive i with matched failures, each pair receives weight $w_i/|\mathcal{N}_i|$, so its pair weights sum to w_i. These weights are not renormalized over matched positives. The total preference weight is therefore the sum of w_i over positives with at least one matched failure.

Table 6: **Generation, search, and learning configuration.** Shared entries apply to all methods where relevant. FEED entries also apply to its objective ablations.

Setting	Value
Sampling	Temperature 1.0, top-$p = 1.0$ and top-$k = 0$ (vLLM 0.17.1 defaults, neither is overridden in the generation request)
Decoding	Qwen chat format. If the first-stage budget is exhausted, continue within the remaining context with a 50-token reserve, inserting `</think>` if absent
Search batch	One selected parent, 32 candidate rollouts
Archive	At most 1,000 nodes. Retain the top 2 children per parent at each archive flush
Parent selection	PUCT exploration coefficient $c_{\text{exp}} = 1$
Parameterization	LoRA rank 32, scaling parameter 32, dropout 0. Base-model weights remain fixed
LoRA modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`, `lm_head`
Optimizer	AdamW with zero weight decay, initial learning rate 4×10^{-5}, moments $(0.9, 0.95)$, numerical $\epsilon = 10^{-8}$
Training schedule	One optimizer step per eligible TTT-Discover update or FEED update attempt, no multi-epoch fitting within an attempt
Training execution	One GPU worker, BF16 base-model loading and FP32 log-probability computation, gradient checkpointing enabled
FEED quality weights	$\delta = 0.05$, $\epsilon_s = 10^{-12}$, $o_x = 0$ for all six tasks, equal mixture of uniform and quality-softmax weights
FEED preference	Margin scale $\kappa = 0.5$, fixed initial-model reference, $(\alpha, \beta) = (1, 1)$
FEED KL monitoring	Up to 8 current samples and 8 evenly spaced masked-token positions per sample, full-vocabulary forward KL from the pre-update policy
FEED acceptance	KL $\leq 10^{-3}$, attempted learning rates 4×10^{-5}, 2×10^{-5}, 10^{-5}, stop at the first accepted attempt

Comparison settings. Table 7 lists method-specific learning procedures under the shared search settings. TTT-Discover uses its adaptive entropic advantage function with this study's model and resources, rather than its original publication's full configuration. Positive-only and Preference-only retain FEED's sample selection and update control, setting (α, β) to $(1, 0)$ and $(0, 1)$, respectively, without rescaling the remaining objective. Updates are skipped when the retained objective has no learning signal.

OpenEvolve comparison settings. OpenEvolve v0.3.2 (Sharma, 2025) keeps model parameters fixed and uses the same model, seeds, and resources as the main experiments (Table 5). It also

Table 7: **Learning differences under the shared model, search, evaluation, and resource settings.**

Method	Learning signal	Update and optimizer state
PUCT	Evaluation feedback used for contextual search only	Fixed language model, no parameter updates
TTT-Discover	Current-batch adaptive entropic advantages, importance-weighted policy loss with base-policy KL coefficient 0.1	One update, Adam moments carried across batches, no FEED acceptance-and-retry procedure
FEED (ours)	Quality-weighted supervision of current and historical valid programs plus prompt-matched failure preferences	Fresh optimizer for each attempt, monitored KL acceptance and learning-rate backtracking

shares their task initializations, evaluators, generation backend, evaluation scheduler, thresholds, and metrics. Parent/inspiration selection, the program database, diff-based evolution, and island migration are retained. Table 8 lists the corresponding settings and interface adaptations.

Table 8: **OpenEvolve configuration and interface adaptations.**

Setting	Value
Database	Population 1,000, archive 100, five islands
Selection	Elite/exploration/exploitation ratios 0.1/0.2/0.7, three top and two diverse programs in prompts
Migration	Every 50 island generations, rate 0.1
Program limit	131,072 characters
Batch execution	Select, generate and evaluate 32 candidates synchronously, then commit results to the database
Generation	Same sampling and decoding settings as Table 6, with token limits from Table 5
Diff interface	Retain task instructions and evaluation context. Remove duplicate parent code and the full-program response instruction from the task adapter
Prompt truncation	Limit inputs to 7,168 tokens. Remove inspiration, diverse-program, previous-attempt, then top-program blocks. Shorten parent excerpt if still needed
Edit application	SEARCH/REPLACE edits must match the full parent, retained separately from the truncated prompt

Task instances and scores. Table 9 lists the task instances, score definitions, and per-candidate time limits. In the autoconvolution scores, $*$ denotes convolution (discrete for sequences), $\|\cdot\|_p$ denotes the L^p norm, and $\|\cdot\|_\infty$ denotes the essential supremum norm. For AC2, each run contributes a score of zero to the cross-run mean until its first valid result.

Table 9: **Task-specific candidate evaluation.** Time limits apply to each candidate.

Task	Instance and reported score	Time limit (s)
32-circle ↑	32 non-overlapping circles inside the unit square, sum of radii of a valid packing	530 timeout
MaxDet ↑	29×29 matrix M with entries in $\{-1, +1\}$, $\lvert\det M\rvert/(342 \cdot 7^{12} \cdot 2^{28})$	300 construction, 350 timeout
OBP ↓	Four fixed CALM training instances (no held-out evaluation): Weibull, 1,000 or 5,000 items, each with bin capacity 100 or 500, mean number of used bins	60 timeout
CVRP-ACO ↓	Ten fixed CALM training instances (no held-out evaluation) with 50 customers and one depot, vehicle capacity 50, mean route cost from ACO with 30 ants and 100 iterations, solver seed 42, on CPU	60 timeout
AC1 ↓	Nonnegative sequence $a = (a_1, \ldots, a_n)$ of length n, discrete autoconvolution bound $2n \max(a * a)/(\sum_i a_i)^2$	1,000 stated in prompt, 1,100 timeout
AC2 ↑	Nonnegative sequence specifying the heights of a step function f, bound $\|f*f\|_2^2/(\|f*f\|_1\|f*f\|_\infty)$, with the evaluator's piecewise-linear integration of the discrete convolution	1,000 stated in prompt, 1,100 timeout

C.4 Prompt example

The following OBP template illustrates how contextual search presents the task instructions, parent program, and evaluation feedback to the model. Parent context is truncated according to Table 5. Here, `timeout_s` is 60 seconds, `score` is the parent's raw score (or `not yet evaluated`), and `previous` is its program text.

````
Design an online bin packing heuristic.  Define
step(item_size:  float, remaining_capacity:
np.ndarray) -> np.ndarray.  Return one priority for
each feasible bin; the largest priority is chosen.
Minimize mean bins used over the fixed training
instances.  Do not mutate inputs.
Whole candidate evaluation timeout:  {timeout_s}
seconds.
Previous program score:  {score}
```python
{previous}
```
Return the complete implementation in one fenced
python block.
````

C.5 TIME ACCOUNTING

Table 10 partitions the first 60 minutes of wall-clock time into generation, evaluation, learning, and other costs, without double-counting overlapping intervals. Evaluation uses batch wall time. Learning includes preparation, KL checks, retries, and cleanup. The Other category includes initialization, otherwise unclassified time, and skipped-update overhead. Only candidates whose evaluations finish within the first 60 minutes contribute to reported scores, even if the rest of their batch finishes later.

Table 10: **First-hour wall-clock cost (minutes, mean ± sample standard deviation over five seeds).** Unrounded means sum to 60 minutes.

Task	Method	Generation	Evaluation	Learning	Other
32-circle	PUCT	38.94 ± 1.64	20.72 ± 1.65	0.00 ± 0.00	0.33 ± 0.02
	TTT-Discover	30.44 ± 7.30	16.97 ± 10.34	12.36 ± 3.05	0.22 ± 0.05
	FEED (ours)	31.56 ± 5.45	13.33 ± 8.49	14.90 ± 3.13	0.22 ± 0.04
MaxDet	PUCT	30.32 ± 5.14	29.29 ± 5.19	0.00 ± 0.00	0.39 ± 0.05
	TTT-Discover	24.51 ± 1.81	25.07 ± 2.91	10.12 ± 1.17	0.31 ± 0.01
	FEED (ours)	27.43 ± 4.12	21.32 ± 5.39	10.93 ± 1.26	0.33 ± 0.02
OBP	PUCT	57.92 ± 0.48	1.45 ± 0.48	0.00 ± 0.00	0.63 ± 0.03
	TTT-Discover	41.83 ± 0.14	0.84 ± 0.47	16.95 ± 0.45	0.37 ± 0.02
	FEED (ours)	44.42 ± 0.20	0.73 ± 0.19	14.50 ± 0.19	0.36 ± 0.01
CVRP-ACO	PUCT	48.25 ± 1.01	10.97 ± 1.01	0.00 ± 0.00	0.78 ± 0.02
	TTT-Discover	36.30 ± 0.82	6.98 ± 0.29	16.09 ± 0.66	0.63 ± 0.02
	FEED (ours)	39.17 ± 0.67	7.13 ± 0.57	13.06 ± 0.41	0.63 ± 0.01
AC1	PUCT	7.94 ± 0.20	51.88 ± 0.21	0.00 ± 0.00	0.17 ± 0.01
	TTT-Discover	8.35 ± 0.18	49.60 ± 0.19	1.88 ± 0.03	0.17 ± 0.01
	FEED (ours)	8.17 ± 0.60	48.70 ± 0.62	2.98 ± 0.11	0.15 ± 0.02
AC2	PUCT	6.41 ± 1.54	53.43 ± 1.53	0.00 ± 0.00	0.16 ± 0.03
	TTT-Discover	6.51 ± 2.29	51.79 ± 3.01	1.56 ± 0.73	0.14 ± 0.02
	FEED (ours)	5.53 ± 2.17	52.69 ± 3.23	1.65 ± 1.09	0.14 ± 0.01

C.6 CROSS-MODEL EVALUATION

The 32-circle model comparisons use batches of 32, 20 CPU slots per run, and a 180-minute horizon. FEED and TTT-Discover each use one H100 GPU for Qwen3-8B, two for Qwen3-14B and Ministral-3-8B-Reasoning-2512, and four for Qwen3-32B and OLMo-3-7B-Think. Run seeds are 30901–30905 for Qwen3-8B and 32101–32105 for the other models. Resources are matched between methods for each model, but hardware and implementation also vary across models.

D ADDITIONAL MAIN RESULTS

Score-threshold sensitivity. Figure 6 reports threshold sensitivity using per-run first-hit times, not crossings of the mean-score curve.

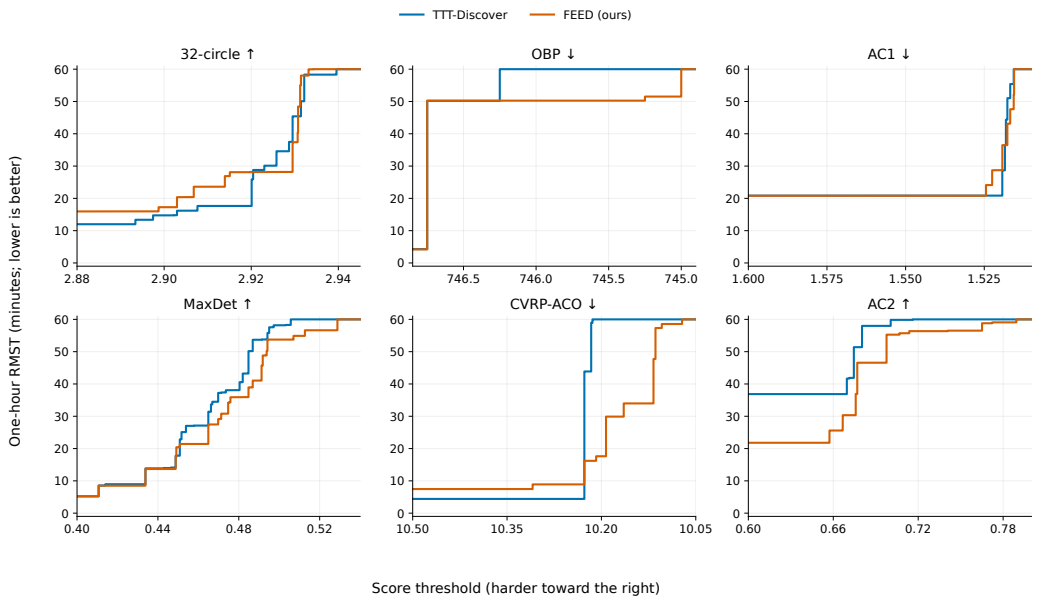

Figure 6: **One-hour RMST across score thresholds, averaged over five runs (lower is better).** Non-hits contribute 60 minutes. Thresholds become harder toward the right. Panels show selected high-quality score ranges. Neighboring thresholds may share a discovery event. FEED's lower RMST extends across neighboring thresholds on CVRP-ACO and AC2, while the ordering varies on other tasks.

Per-seed target attainment. Table 11 reports the individual first-attainment times underlying Table 1.

Table 11: **Per-seed first-attainment times in minutes at the thresholds in Table 1.** Columns identify the random seeds. A value of > 60 denotes no attainment within the one-hour budget and contributes 60 minutes to RMST. Times are rounded to two decimals. RMST uses unrounded times.

Task	Threshold	Method	30901	30902	30903	30904	30905
32-circle	2.93	TTT-Discover	> 60	54.24	51.57	18.80	42.40
		FEED	50.33	44.96	44.53	19.94	27.15
MaxDet	0.52	TTT-Discover	> 60	> 60	> 60	> 60	> 60
		FEED	> 60	> 60	> 60	> 60	43.12
OBP	746.5	TTT-Discover	10.93	> 60	> 60	> 60	> 60
		FEED	> 60	> 60	> 60	11.28	> 60
CVRP-ACO	10.2	TTT-Discover	> 60	> 60	> 60	> 60	> 60
		FEED	11.96	11.70	11.77	26.51	26.17
AC1	1.55	TTT-Discover	36.97	4.79	20.84	20.82	20.82
		FEED	37.14	4.34	20.58	21.14	20.86
AC2	0.6	TTT-Discover	27.05	12.42	24.83	> 60	> 60
		FEED	27.17	11.61	24.01	41.03	5.14