LOOP REINFORCEMENT LEARNING FOR SCIENTIFIC DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated strong search capabilities when given well-defined problems and quantitative verifiers. Integrating LLMs into discovery loops has accelerated scientific discovery through iterative refinement guided by verification feedback. However, LLMs tend to generate similar candidate solutions, concentrating exploration within narrow regions of the search space and risking premature convergence that limits the diversity of discovery. Recent approaches seek to broaden exploration through evolutionary algorithms or agentic orchestration, maintaining diversity and expanding search coverage during test-time scaling. Despite their effectiveness, these approaches still rely on manually designed search rules, heuristics, or agentic workflows, which introduce human-imposed inductive biases that constrain the search space. We therefore propose Loop Reinforcement Learning (Loop RL). We first introduce a minimal harness called Simple Loop that consists only of sequential iteration, parallel execution, and fully asynchronous shared memory and gives the model the flexibility to learn its own search strategies without predefined heuristics. We then use Loop RL to optimize the discovery performance of this loop through credit assignment across sequential iterations and parallel loops on a global asynchronous timeline. Across scientific discovery tasks, Loop RL improves search efficiency and solution diversity, attaining state-of-the-art results on several tasks.

1 INTRODUCTION

Large language models are powerful probabilistic searchers with performance that scales with compute (Brown et al., 2024). Integrating LLMs into evolutionary search systems replaces the laborious manual loop of proposing candidate solutions, evaluating results, and iterating (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2025). Within well-defined feasible search spaces, such systems have achieved notable progress on scientific discovery problems (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2025; Shojaee et al., 2025a).

In practice, LLMs tend to generate similar candidates (Kirk et al., 2024; Yun et al., 2025; Jiang et al., 2025) that concentrate search within a narrow region of the whole solution space despite increased computing for test time scaling, which reduces search efficiency and leads to premature convergence. To address this limitation, existing methods explicitly control the search process to maintain diversity and balance exploration with exploitation, for example, through island-based populations (Novikov et al., 2025), adaptive compute allocation (Cemri et al., 2026), and multi-agent coordination (Qu et al., 2026). However, these strategies rely heavily on human design (Liu et al., 2026) and introduce inductive biases that constrain the search space of candidate solutions.

In this paper, we introduce Simple Loop, a minimal loop framework without additional sophisticated search heuristics. It retains only three essential components for a discovery loop: (1) *sequential refinement* guided by verification feedback; (2) *parallel execution* for broader exploration and greater search efficiency; and (3) *shared memory* for exchanging experience across parallel loops asynchronously. Other design choices like context management, diversity control, and collaboration are leave to the model to learn. We then propose Loop Reinforcement Learning (Loop RL), which trains the model to search and cooperate effectively by directly optimizing the discovery performance of the entire loop. Since candidate solution spaces differ across problems, the flexibility of Simple Loop allows the model to learn search strategies adapted to each problem. Despite asynchronous

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

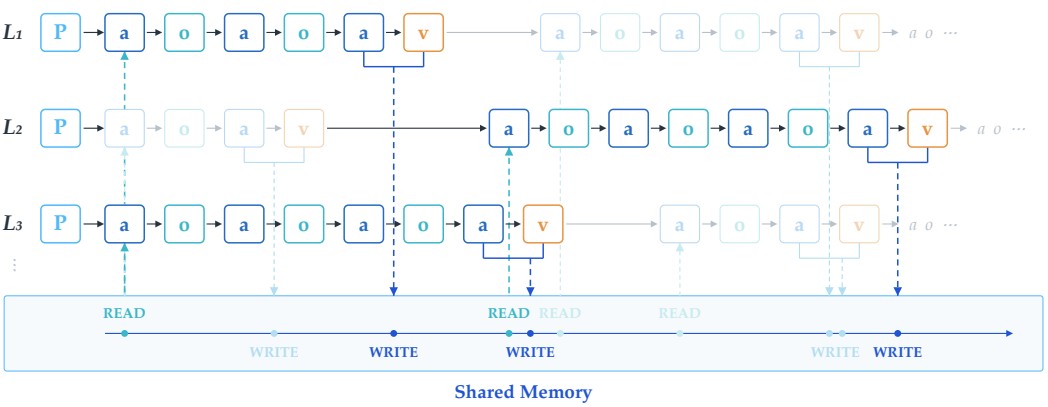

Figure 1: **Overview of Loop RL.** Each parallel loop L_i iteratively refines candidate solutions guided by verification feedback v and executes multi-turn trajectories with interleaved actions a and observations o, exchanging experience through shared memory. The spacing and temporal offsets between iterations reflect differences in the wall-clock time required by individual actions. Each loop runs continuously at its own pace without waiting for other loops; the gaps in the diagram do not indicate idle time. The timestamps of each iteration's first `read` and completed `write` actions are recorded on a global asynchronous timeline to establish temporal dependencies between iterations for credit assignment.

communication, Loop RL uses read and write timestamps to place each iteration on a global timeline and establish temporal dependencies based on information availability. It rewards each iteration for improving upon the best result available at its first read and propagates subsequent rewards backward according to these dependencies, as shown in Figure 1. Loop RL also supports online learning during search by periodically updating model parameters using collected trajectories, enabling the model to learn new behaviors for the problem as discovery progresses.

To demonstrate its effectiveness, we evaluate Loop RL on scientific discovery tasks spanning mathematical construction, signal processing, and symbolic regression. Trajectory analyses show how agents verify and adapt shared experience to improve their search. Signal Processing ablations examine parallel–sequential budget allocation and show that cross-loop credit assignment achieves a higher final score than within-loop credit assignment.

Our contributions are threefold: (1) Simple Loop, a minimal harness for asynchronous discovery with free-form shared memory; (2) Loop RL, a training algorithm that assigns credit across sequential and parallel iterations; and (3) empirical analyses of experience reuse, search-budget allocation, and cross-loop credit assignment.

2 Related Work

Test-time Discovery. Test-time scaling improves discovery through loops guided by verification feedback, where LLMs act as mutation operators for candidate solutions, as in FunSearch (Romera-Paredes et al., 2024). To maintain search diversity and explore the search space effectively, subsequent work orchestrates search through MAP-Elites and multi-island evolution (Novikov et al., 2025; Sharma, 2025), adaptive parent sampling and novelty filtering (Lange et al., 2025), and Pareto-based selection (Agrawal et al., 2026). Further approaches introduce agentic adaptive search control and strategy evolution (Cemri et al., 2026; Liu et al., 2026), seperate agent roles for mangage discovery directions (Liu et al., 2025; Hyra Team, 2026), or active agentic mutation (Chen et al., 2026; Qu et al., 2026). However, these methods still depend on handcrafted search heuristics to explicitly steer exploration for maintaining diversity and avoiding stagnation, like periodic adjustments to search intensity across different search directions (Cemri et al., 2026; Liu et al., 2026; Qu et al., 2026), which introduce inductive biases that constrain how LLMs explore the large search space of candidate solutions. We instead build Simple Loop as a minimal framework that leaves the model

ample freedom to develop its own search and cooperation strategies, and use Loop RL to directly optimize the discovery performance of the entire loop.

Learning to Discover. Another line of work improves the model's search capability through training, including distillation from stronger models (Lee et al., 2026), iterative learning from past attempts (Surina et al., 2025; Ye et al., 2026), and online parameter updates during search using predefined reward functions, as in TTT-Discover and ThetaEvolve (Yuksekgonul et al., 2026; Wang et al., 2025). Yet these methods optimize single-turn search conditioned on the curated prompt, whereas discovery loops require multi-turn iterative improvement informed by past experience. In kernel optimization, Kevin trains multi-turn refinement with discounted returns computed from individual attempt scores within each trajectory, but does not account for improvements over the global best score available across cooperating loops (Baronio et al., 2025). Loop RL instead trains a single policy model to improve the entire discovery loop through learning sequential refinement and experience sharing strategy across parallel loops.

3 Approach

Our approach follows a simple principle: construct a minimal discovery loop framework to preserve the search space, then train the policy model with Loop RL to learn its own search and cooperation strategies for better discovery result.

3.1 Simple Loop

Let θ denote the policy model parameters. Given a problem P, the agent (the model together with its execution harness) executes a T-turn interaction trajectory

$$\tau\colon (a_1, o_1, \ldots, a_T, o_T), \qquad a_t \sim \pi_\theta(\cdot \mid P, a_{<t}, o_{<t}),$$

and produces an artifact, which is evaluated by an external verifier. The verification score is then provided to the agent for the next iterative improvement. We use v to denote the verification score.

A sequential loop can be written as

$$L = P,\ \tau_1\colon (a, o, a, o, \ldots),\ v_1,\ \ldots,\ \tau_m\colon (a, o, a, o, \ldots),\ v_m, \tag{1}$$

where a and o denote actions and observations, respectively, τ_j is the mulit-turn trajectory of iteration j, and m is the number of sequential iterations.

Extending this to n parallel loops gives

$$L_i = P,\ \tau_{i,1},\ v_{i,1},\ \ldots,\ \tau_{i,j}\colon (a, o, a, o, \ldots),\ v_{i,j},\ \ldots,\ \tau_{i,m},\ v_{i,m}. \tag{2}$$

Here, i indexes parallel loops and j indexes sequential iterations. Each iteration is represented by the pair $(\tau_{i,j}, v_{i,j})$, consisting of an interaction trajectory and its verification score.

To support information sharing across parallel loops for efficient search, we introduce a simple shared memory that stores free-form text and provides a search interface supporting arbitrary retrieval functions. The agent can use `write` to share information and `read` together with its own retrieval functions to find relevant information. Implementation details are provided in Appendix A.

Simple Loop is similar to CORAL (Qu et al., 2026) in supporting asynchronous parallel search, but differs in three aspects:

1. **Memory storage.** CORAL uses predefined `attempts/`, `notes/`, and `skills/` categories, prompts agents to organize knowledge, and enables code reuse through shared Git history. Simple Loop stores free-form text organized by the agent and runs each parallel loop in a separate container. Information exchange is restricted to shared memory, preventing direct access to or forking of other loops' code repositories to ensure independence.
2. **Experience retrieval.** CORAL presents past attempts in a score-based leaderboard, which can bias the agent toward current local optima and reduce search diversity. However, Simple Loop imposes no metric-based ranking and instead provides a general interface through which the agent can implement arbitrary retrieval functions to find relevant experience.

3. **Reflection and knowledge consolidation.** CORAL uses periodic heartbeat prompts for reflection and knowledge consolidation to reduce local search stagnation and prevent agents from overlooking shared knowledge. Simple Loop instead omits these human-designed interventions and leaves these decisions to the agent, using Loop RL to train the policy model to learn effective strategies.

3.2 LOOP RL

Single-turn RL optimizes model actions conditioned on a prompt directly, while multi-turn RL optimizes a trajectory of interleaved observations and actions. However, optimizing trajectories independently does not account for cross-trajectory information exchange, which is important for effective parallel search. We thus propose Loop RL to improve discovery performance by training the model to learn its own search and cooperation strategies across parallel loops.

For each iteration $(\tau_{i,j}, v_{i,j})$ in Eq. (2), we record a timestamp pair $(t^s_{i,j}, t^e_{i,j})$, marking its first `read` time and the completion time of its `write` to shared memory, respectively. These timestamps place all iterations on a global timeline, where their execution intervals can overlap.

The reward of $(\tau_{i,j}, v_{i,j})$ is defined as its verification score minus the best score written to memory before its first read, clipped at zero, encouraging the agent to improve discovery results based on the context it received by refining its candidate solution using its own past experience or drawing on experience shared by other loops:

$$r_{i,j} = \max\left(0,\ v_{i,j} - \max\{v_{a,b} \mid t^e_{a,b} < t^s_{i,j}\}\right). \tag{3}$$

An iteration receives a positive reward if it improves upon the global best score provided in its context at the first read, and zero otherwise, in order to avoid assigning negative rewards to experience with potential benefits for other iterations. For iterations that overlap in time, the baseline for each iteration is fixed at its first read, so results published by other loops during its execution do not change it.

To account for subsequent improvements across parallel loops, we construct a global return using the same temporal relation. For iteration $(\tau_{i,j}, v_{i,j})$, let (a_k, b_k), $k = 1, \ldots, K$, index the K iterations in the same discovery run satisfying $t^e_{i,j} < t^s_{a_k,b_k}$, ordered by their first `read` times. This relation defines a temporal dependency: information written by $(\tau_{i,j}, v_{i,j})$ is available before each of these iterations begins reading. We define its global return as

$$G_{i,j} = r_{i,j} + \sum_{k=1}^{K} \gamma^k r_{a_k,b_k}, \tag{4}$$

where $\gamma \in (0, 1]$ is the discount factor, with $\gamma = 1$ giving an undiscounted return. Each subsequent reward contributes once to $G_{i,j}$, allowing future improvements from both the same loop and other parallel loops to inform the current iteration's update.

Implementation. We use GRPO (Shao et al., 2024) to optimize the policy across the entire discovery loop, considering its stable updates without relying on a value model that can be biased on discovery tasks. At each training step, we normalize the global returns $G_{i,j}$ within each problem across sequential and parallel iterations to obtain an advantage for each iteration, which is then broadcast to all trainable action tokens in the corresponding trajectory $\tau_{i,j}$, including those generated during self-compaction for long-context management; prompt and observation tokens are excluded from the policy loss. The shared memory for each problem persists across model training steps.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate Loop RL on a fixed suite of ten tasks spanning mathematical construction, signal processing, and symbolic regression (Table 1). The suite includes mathematical problems studied in AlphaEvolve and ThetaEvolve (Novikov et al., 2025; Wang et al., 2025), the signal processing task used in EvoX (Liu et al., 2026), and two symbolic regression problems from

Table 1: Benchmark tasks, evaluation metrics, and brief descriptions. Arrows indicate whether higher (↑) or lower (↓) values are better. ID NMSE denotes in-distribution normalized mean squared error.

Task	Metric	Description
Second autocorrelation	Reference ratio ↑	Construct nonnegative functions to improve a lower bound on an autocorrelation constant.
Third autocorrelation	Reference ratio ↑	Construct signed functions to improve an upper bound on an autocorrelation constant.
Erdős minimum overlap	Reference ratio ↑	Construct step functions to reduce the upper bound on the minimum-overlap constant.
Heilbronn convex / 13	Reference ratio ↑	Place 13 points in a unit-area convex region to maximize the smallest triangle area.
Hexagon packing / 11	Reference ratio ↑	Pack 11 unit regular hexagons into the smallest possible regular hexagon.
Signal processing	Composite score ↑	Design a filter balancing signal fidelity, smoothness, lag, and false trend changes.
Circle packing / 26	Sum of radii ↑	Pack 26 non-overlapping circles in a unit square to maximize the sum of their radii.
Hadamard / 29	Determinant ratio ↑	Maximize the absolute determinant of a 29×29 matrix with entries in $\{-1, +1\}$.
Bio / BPG0	ID NMSE ↓	Discover a population-growth equation from data.
Materials / MatSci0	ID NMSE ↓	Discover a stress–strain relation from data.

LLM-SRBench (Shojaee et al., 2025b). Each parallel loop maintains a persistent workspace isolated from other loops. Agents run local experiments and consult shared experience before submitting an artifact to an external evaluator. The resulting score is recorded in shared memory alongside a summary of the artifact for other loops to read. All loops run asynchronously at their own pace, without synchronization barriers or fixed iteration durations.

Model. We use Qwen3.8-27B (Qwen Team, 2026) as the policy model for discovery and training, with Claude Code (Anthropic, 2025) as the agent harness. We set the context window to 256K tokens. For each problem, we set the number of parallel loops to n and the number of sequential iterations per loop to m. We update the model after all $m \times n$ iterations are complete. Trajectories can span multiple context windows. During automatic context compaction, generated summary tokens contribute to the policy loss, while the same summary is masked out when reused as context in the next window. For detailed ablations, we compare with CORAL (Qu et al., 2026), using the same policy model and agent harness in both single-agent and multi-agent settings.

Allocated Computing Resources. We do not impose a fixed wall-clock budget, since the inference speed of open-weight models depends on the deployment hardware. Instead, we use the total number of iterations, $m \times n$, to control the search budget across methods. Loop RL requires additional computation for training, providing another way to scale scientific discovery when inference alone is insufficient. We account for both inference and training time when reporting the total computational cost. Following CORAL, we report results averaged over four independent runs.

4.2 Results

We analyze a run with four parallel loops and up to ten submissions per loop, supplemented by archived cases beyond the ten-task suite. Search decisions are reconstructed from summaries, references, and submitted code. Endpoints come from external evaluations; intermediate comparisons are agent-reported local experiments.

Parallel Breadth and Sequential Refinement. Figure 2 compares prefixes of the main run, using a fixed group of loops with eight consecutive scored submissions per task. For $n \in \{1, 2\}$ and $m \in \{1, 2, 4, 8\}$, we average the best score among the first m submissions over all subsets of n eligible loops. At eight submission slots, $(n, m) = (2, 4)$ outperforms $(1, 8)$ on Signal Processing (0.6410

Hexagon / 11 — 2 eligible loops

	$m=1$	$m=2$	$m=4$	$m=8$
$n=1$	0.500267	1.001035	1.001070	1.001111
$n=2$	1.000535	1.001347	1.001404	1.001420

Reference ratio ↑

Signal processing — 3 eligible loops

	$m=1$	$m=2$	$m=4$	$m=8$
$n=1$	0.5400	0.5686	0.6162	0.6166
$n=2$	0.5610	0.5954	0.6410	0.6418

Composite score ↑

Erdős minimum overlap — 2 eligible loops

	$m=1$	$m=2$	$m=4$	$m=8$
$n=1$	0.999230	1.000094	1.000117	1.000119
$n=2$	1.000008	1.000111	1.000122	1.000125

Reference ratio ↑

Third autocorrelation — 2 eligible loops

	$m=1$	$m=2$	$m=4$	$m=8$
$n=1$	0.996992	0.998502	0.998537	0.999077
$n=2$	0.998548	0.998984	0.999047	0.999105

Reference ratio ↑

Figure 2: **Parallel and sequential prefixes.** Entries average the best score over subsets of n eligible loops and their first m submissions. Outlines compare eight submission slots. Eligible groups are fixed within each task; prefixes retain their original shared experience.

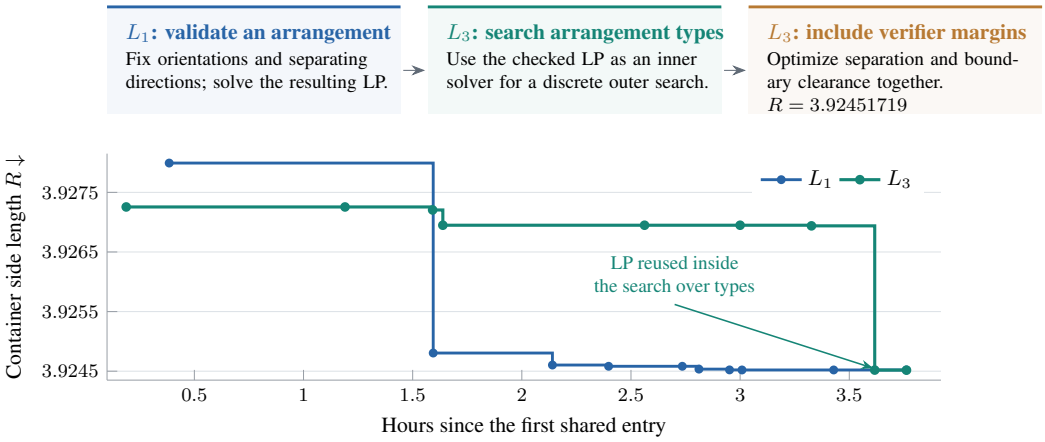

Figure 3: **Reusing a shared LP in Hexagon Packing.** L_3 verifies L_1's formulation, uses it to search arrangement types, and incorporates verifier margins. Steps connect evaluated submissions; coordinates are reconstructed from the archived figure.

versus 0.6166) and Hexagon Packing (1.001404 versus 1.001111). Third Autocorrelation slightly favors deeper continuation (0.999077 versus 0.999047), while Erdős Minimum Overlap differs little. These observed prefixes retain the original communication histories and are not independently rerun ablations under matched computational cost.

Turning Shared Experience into a Search Procedure. In Hexagon Packing, perturbations, annealing, and local refinement repeatedly return one loop to a similar configuration. Another loop shares a linear program (LP): fixing orientations and pairwise separating directions makes the remaining position and container constraints linear. After reproducing a known packing, the receiving loop uses this LP as an inner solver for a search over arrangement types and orientations, reaching a region missed by its continuous search (Figure 3). It then incorporates the evaluator's separation and boundary margins directly into the LP instead of uniformly expanding the configuration. The submitted side length reaches $R = 3.92451719$. A shared procedure for checking arrangements thus becomes a method for discovering new ones.

Revising Search Objectives and Assumptions. In the archived Heilbronn-14 run, local moves and restarts recover the same configuration. Peers achieve a larger minimum triangle area with the same hull and interior point counts but different tight triangles (Figure 4). The agent introduces an

	EARLIER SEARCH	NEW EVIDENCE	NEXT EXPERIMENT
Heilbronn-14 Generate new starts	Max–min search returns to the same set of tight triangles.	→ Peers find a better layout with the same hull counts but different tight triangles.	→ Penalize small triangles to generate starts, then apply max–min refinement.
Borcea Reopen a search direction	Repeated searches favor $K = 0$, which becomes a restriction on later attempts.	→ A peer benefits from $K \neq 0$; local scoring disagrees with reconstruction.	→ Free K and align local scoring with polynomial reconstruction. $0.919 \longrightarrow 0.972$
Kernel Shorten the serial tail	Index computation and boundary checks follow the hash.	→ Candidate indices can be prepared early; only their final selection depends on the hash.	→ Prepare and clamp both candidates in advance, then select. $2{,}103 \longrightarrow 2{,}100$ cycles

Figure 4: **Shared evidence changes the next experiment.** Agents revise candidate generation, search restrictions, scoring, and computation dependencies. Endpoints refer to submitted artifacts; the recorded sequences do not isolate the effect of individual changes.

objective penalizing all triangles below a gradually increasing area target. This objective alone does not improve the incumbent; using it to generate starts for the original max–min refinement reaches the stronger peer construction. Shared experience changes candidate generation while preserving the existing refinement procedure.

In Borcea, repeated returns to an antiderivative constant $K = 0$ lead the agent to restrict subsequent searches. A peer's stronger construction with nonzero K challenges this restriction. The agent reopens K and discovers that its shortcut scorer disagrees with the evaluator's reconstruction of the polynomial from roots. Aligning local scoring with reconstruction yields an externally evaluated improvement from approximately 0.919 to 0.972, without establishing a counterexample to the underlying conjecture.

In sum–difference sets, the agent favors the largest scale in a nested-set family because it maximizes an intermediate ratio. A peer's stronger intermediate-scale construction reveals that the final exponent also depends on logarithmic normalization. The agent rebuilds the candidate, checks neighboring sets, and revises its earlier claim. The evaluated bound improves from $C_6 \approx 1.12731$ to 1.12912.

Reassessing Earlier Improvements. In Second Autocorrelation, log-space rescaling improves the initial construction and is retained in the program. After repeated optimization passes produce stronger solutions, the agent retests rescaling and finds that it now degrades them, sometimes triggering a fallback to a weaker construction. Removing rescaling while retaining repeated passes yields $C_2 \approx 0.94833$. As gains diminish, the agent proposes searching another region. The useful experience includes when a previously successful operation should be abandoned.

In an additional Signal Processing run, hard thresholding reduces shrinkage but leaves systematic amplitude attenuation. After diagnosing this residual error, the agent adds gains for retained coefficients at different scales. Local tests show that amplification restores detail but exposes spikes, which the median stage removes. A simple geometric gain schedule performs similarly to independently fitted band gains and is retained. The submitted composite score reaches approximately 0.766, with a remaining failure case recorded for one signal type.

Adapting Computation and Representation. In kernel optimization, rewriting parity leaves the same serial dependency chain after a hash computation. A shared precomputation idea leads the agent to prepare and clamp both candidate indices while the hash runs, selecting between them once parity becomes available (Figure 4). It also moves a constant calculation to an underused arithmetic logic unit to avoid a bottleneck load slot. The final submission improves from 2,103 to 2,100 cycles.

For Max-Cut G11, the agent recognizes a narrow toroidal grid with signed weights and uses dynamic programming over row assignments. For G14, which has all-positive weights and different connectivity, it develops local search with incremental flip gains, joint vertex moves, and paths between strong solutions. Recomputed archived partitions give cuts of 564 and 3,064, respectively. The G14 value matches the BLS reference in Acikalin et al. (2025); the agents' additional optimality arguments remain uncertified.

In RNA degradation prediction, the agent aligns validation with the grader's high-signal-to-noise population. A sequence neural network is slower and performs worse locally than a tree model under CPU-only resources. The agent retains the tree model and adds base-pair probabilities to represent structural uncertainty omitted by a single predicted structure. The submitted model achieves a mean column-wise root mean squared error (MCRMSE) of approximately 0.240.

4.3 Discovery Dynamics

Communication Behavior. With free-form shared memory, agents construct retrieval functions according to their current search needs, making parent selection part of their search strategy. We observe four recurring retrieval behaviors. First, agents read batches of records without filtering to inspect previous methods and failed attempts before choosing a starting point, as seen early in Hexagon Packing. Second, agents use timestamp filters, sometimes combined with chronological sorting, to check for new evidence produced by other loops since their last read; this occurs in Hexagon Packing and Circle Packing. Third, agents explicitly exclude their own `parallel_id` to seek experience from other loops in Signal Processing, Hadamard, and Autocorrelation. Fourth, agents apply score thresholds to identify promising attempts for further inspection, such as `combined_score` > 0.9992 in Third Autocorrelation, with similar filtering in Heilbronn. These thresholds select qualifying attempts without requiring the agent to follow the highest-scoring solution. Agents also retrieve by keyword or inspect another loop's full history. After several iterations without progress, they revisit the initial division of work across loops and reconsider their search directions. These behaviors show that the context agents seek extends beyond a fixed score-based leaderboard.

In the main run, 85.9% of explicit references point to other parallel loops, and 45.3% of submissions reference at least two other loops. Agents reinterpret this shared experience for the current problem, using it to construct new search tools, diagnose stagnation, revise assumptions, or identify candidates for independent verification.

Cooperation across Parallel Loops. In the analyzed trajectories, we do not observe agents simply converging on the same local solution or abandoning their own candidates when another loop reports a higher score. Instead, agents treat shared conclusions as claims to verify, often running multiple experiments to reproduce them. They then use the resulting insights to refine their own search directions without adopting another loop's approach wholesale. In Hexagon Packing, the agent verifies a shared LP and gives it a new role in its own search (Figure 3). In sum–difference sets, it rebuilds a peer's candidate and checks neighboring constructions before revising its earlier conclusion.

We observe two recurring forms of cooperation. First, agents broadcast errors through shared memory. When one agent identifies an issue, such as an infrastructure error, other loops notice and verify the report, allowing the information to spread across loops. Second, agents correct one another. When an agent finds an error while reproducing another loop's result, it records the issue so that others can avoid the same pitfall. The original agent uses this feedback to revise its understanding and records subsequent experience more cautiously.

Effects of Training. Loop RL targets problem-dependent search and cooperation strategies, including changing the search representation in Hexagon Packing and reassessing transformations in Second Autocorrelation. Across tasks, agents verify shared claims, diagnose stagnation, and adapt earlier procedures. Global returns assign training signals from subsequent improvements across loops, while persistent memory retains problem-specific experience. These cases document in-context adaptation; the ablation below compares global and within-loop credit assignment during training.

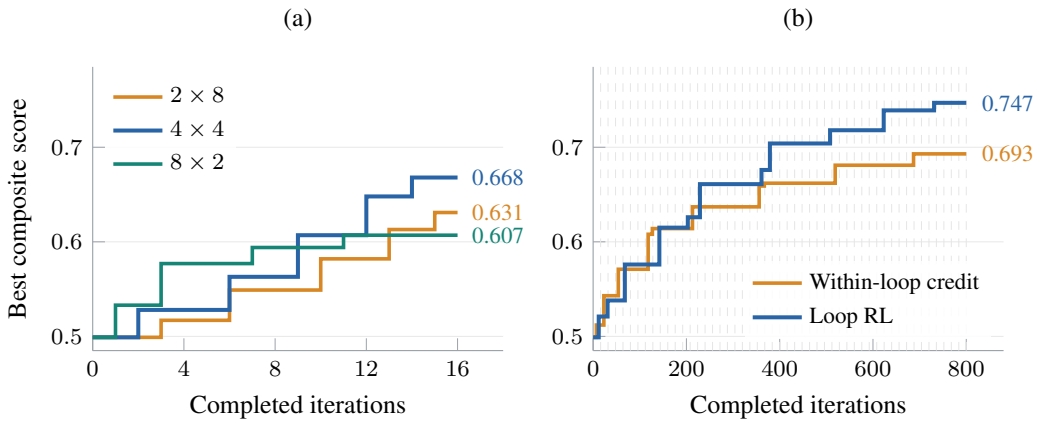

Figure 5: **Signal Processing ablations.** (a) Parallel × sequential allocations with 16 iterations. (b) Within-loop credit assignment and Loop RL with $n = m = 4$. Curves show the best composite score; dashed lines mark updates every 16 iterations.

4.4 Ablations

We illustrate both ablations with Signal Processing traces (Figure 5).

Parallel and Sequential Budget Allocation. We compare 2×8, 4×4, and 8×2 allocations with fixed model parameters, shared memory, and a budget of 16 iterations. In the traces, 8×2 progresses fastest initially, reaching 0.577 after four iterations versus 0.528 for 4×4 and 0.517 for 2×8. However, 4×4 finishes highest at 0.668, followed by 0.631 for 2×8 and 0.607 for 8×2. Parallelism broadens starting ideas, while sequential iterations support refinement and experience reuse through shared memory. These results favor balancing exploration with enough sequential iterations for refinement and interaction across loops.

Credit Assignment across Parallel Loops. We compare a Kevin-inspired multi-turn baseline (Baronio et al., 2025), which normalizes returns globally and assigns credit within each trajectory, with Loop RL, which establishes temporal dependencies on a global asynchronous timeline for backward credit assignment. We set $n = m = 4$ and update every 16 iterations for 50 training steps. The Loop RL trace shows a more pronounced upward trend and a higher final score, reaching 0.747 after 800 iterations versus 0.693 for the baseline. Dashed lines mark model updates after each batch of 16 iterations.

5 Conclusion

In this paper, we propose Simple Loop, a minimal harness for scientific discovery, and Loop RL to train the model to learn effective search and cooperation strategies. Our experiments show that, without human-designed heuristics, Simple Loop exhibits discovery behaviors distinct from those of previous frameworks. Parallel loops cooperate and improve one another's solutions by adapting and combining shared insights without directly copying other loops' solutions, producing diverse candidates and overcoming the loss of diversity imposed by human design. Loop RL further strengthens the model's search and cooperation capabilities, reinforcing different strategies across tasks and accelerating discovery. We will continue running Loop RL with the aim of reporting further results on frontier problems.

AI USE STATEMENT

We used large language models (LLMs) only to polish the manuscript's grammar and assist with coding. AI tools were not used in any other part of the research or manuscript preparation.

ETHICS STATEMENT

This work studies automated scientific discovery through iterative program generation and verification. Automatically generated solutions can exploit gaps in task-specific verifiers. Improved verification scores should therefore be interpreted within the assumptions and coverage of each verifier, and solutions intended for downstream use should undergo independent validation. The computational costs of parallel search and reinforcement learning should also be considered when assessing the practical benefits of the method.

REPRODUCIBILITY STATEMENT

The Approach section specifies the sequential and parallel loop structure, shared-memory operations, and the reward, return, and policy-update definitions.

REFERENCES

Utku Umur Acikalin, Aaron Ferber, and Carla P. Gomes. Learning to explore and exploit with GNNs for unsupervised combinatorial optimization. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/e944bacecce6b06374ac39b260348db0-Paper-Conference.pdf.

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, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Anthropic. Claude Code, 2025. URL https://github.com/anthropics/claude-code.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn RL for generating CUDA kernels. *arXiv preprint arXiv:2507.11948*, 2025. URL https://arxiv.org/abs/2507.11948.

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

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

Terry Chen, Zhifan Ye, Bing Xu, Zihao Ye, Timmy Liu, Ali Hassani, Tianqi Chen, Andrew Kerr, Haicheng Wu, Yang Xu, Yu-Jung Chen, Hanfeng Chen, Aditya Kane, Ronny Krashinsky, Ming-Yu Liu, Vinod Grover, Luis Ceze, Roger Bringmann, John Tran, Wei Liu, Fung Xie, Michael Lightstone, and Humphrey Shi. Avo: Agentic variation operators for autonomous evolutionary search, 2026. URL https://arxiv.org/abs/2603.24517.

Hyra Team. Hyra: Hunyuan research agent. https://hy.tencent.com/research/hyra, 2026.

Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. Artificial Hivemind: The open-ended homogeneity of language models (and beyond). In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2732. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/754d5a526a5ee5a47220664a0eb92751-Abstract-Datasets_and_Benchmarks_Track.html.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of RLHF on LLM generalisation and diversity. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5a68d05006d5b05dd9463dd9c0219db0-Abstract-Conference.html.

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

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, 2026. URL https://arxiv.org/abs/2606.29082.

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

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

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

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. Coral: Towards autonomous multi-agent evolution for open-ended discovery, 2026. URL https://arxiv.org/abs/2604.01658.

Qwen Team. Qwen3.8-27B. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/28df8e730c054c5331855fd4d5403ba9-Abstract-Conference.html.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D. Doan, and Chandan K. Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 55325–55359. PMLR, 2025b. URL https://proceedings.mlr.press/v267/shojaee25a.html.

Anja Surina, Amin Mansouri, Lars C.P.M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.05108.

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

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

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

Longfei Yun, Chenyang An, Zilong Wang, Letian Peng, and Jingbo Shang. The price of format: Diversity collapse in LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15454–15468. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.836. URL https://aclanthology.org/2025.findings-emnlp.836/.

A SHARED MEMORY IMPLEMENTATION

We implement shared memory as a Model Context Protocol (MCP) server with `read` and `write` interfaces, backed by a SQL database. Table 2 lists the fields stored for each entry.

Table 2: Fields stored in shared memory.

Field	Description
`global_id`	Unique identifier of the entry across parallel loops.
`parallel_id`	Identifier of the parallel loop that wrote the entry.
`iteration_id`	Sequential iteration index within that loop.
`summary`	Free-form text written by the agent.
`refs`	Global IDs of entries referenced by this entry.
`ref_by`	Global IDs of entries that reference this entry.
`write_timestamp`	Time when the write completes, corresponding to $t^e_{i,j}$.

The two tool interfaces and their MCP descriptions are given below. The `read` tool accepts retrieval code written by the agent, leaving the selection and ordering of entries to the agent. The `write` tool stores the text and references from the current iteration. The server assigns a global ID, records the write timestamp, and updates `ref_by` for the referenced entries.

`read` interface

```
read(retrieve_fn) -> entries

MCP tool description:
Retrieve shared experience using your own retrieval function.
Select, filter, or order entries according to your current search
    needs.

Parameter:
  retrieve_fn: Source code of a function that queries the shared
               SQL database and returns selected entries.

Returns:
  Memory entries selected by the retrieval function.
```

`write` interface

```
write(parallel_id, iteration_id, summary, refs) -> global_id

MCP tool description:
Write free-form text to shared memory for use by other iterations.
Include the global IDs of any memory entries you reference.

Parameters:
  parallel_id:  ID of the current parallel loop.
  iteration_id: Index of the current sequential iteration.
  summary:      Free-form text to share.
  refs:         Global IDs of referenced entries; an empty list
                if no entries are referenced.

Returns:
  The global ID of the newly written entry.
```