AlphaResearch: Can Peer-reviewed Language Models Accelerate New Algorithm Discovery?

Anonymous ACL submission

Abstract

LLMs have made significant progress in complex but easy-to-verify problems, yet they still struggle with discovering the unknown. In this paper, we present **AlphaResearch**, an autonomous research agent designed to discover new algorithms on open-ended problems by iteratively running the following steps: (1) propose new ideas (2) program to verify (3) optimize the research proposals. To synergize the feasibility and innovation of the discovery process, we construct a novel dual environment by combining the execution-based verifiable reward and reward from simulated real-world peer review environment in AlphaResearch. We construct **AlphaResearchComp**, a set of questions that includes an eight open-ended algorithmic problems competition to benchmark AlphaResearch. Experimental results show that AlphaResearch achieves stronger discovery performance than other agentic discovery systems on six open-ended problems. Notably, the algorithm discovered by AlphaResearch on the *"packing circles"* problem achieves the best-of-known performance, surpassing the results of human researchers and strong baselines from recent work (e.g., AlphaEvolve). Additionally, we conduct a comprehensive analysis of the benefits and remaining challenges of autonomous research agent, providing valuable insights for future research.

1 Introduction

Recent progress has shown that sophisticated coding agent scaffolds (Yang et al., 2024) could help frontier LLMs (OpenAI, 2025; Comanici et al., 2025) achieve expert-level performance in end-to-end coding problems (e.g., modify the code to pass the test cases in sandbox (Jimenez et al., 2024)). However, for open-ended coding problems where the target endpoint (i.e., the optimal implementation) is left underspecified, end-to-end coding agents often struggle to converge on the best possible solution with just a few attempts.

AlphaEvolve (Novikov et al., 2025) shows that scaling the number of agent attempts can reveal an emerging ability to derive stronger solutions from suboptimal prior results. However, its brute-force search process repeatedly samples new attempts from a program database until a preset iteration limit is reached, substantially increasing validation cost. This raises a key question: *Can we accelerate new algorithm discovery by reducing ineffective attempts?*

ShinkaEvolve (Lange et al., 2025) advances this direction by introducing a more sample-efficient evolutionary framework for open-ended program evolution. Specifically, it filters out redundant or minimally changed edits before expensive validation using a novelty rejection-sampling mechanism that combines embedding-based code similarity with an LLM-based novelty judge. This improves exploration efficiency relative to pure brute-force sampling, but important limitations remain. Embedding-based similarity thresholds can miss semantically meaningful changes or, conversely, fail to detect subtle redundancies. In addition, LLM-based novelty judgments are inherently unreliable because there is little high-quality training data explicitly annotated for program novelty in open-ended evolutionary search. As a result, general-purpose LLMs may rely on superficial cues and produce judgments that diverge from expert assessments of functional improvement, algorithmic originality, or non-trivial semantic change (Zheng et al., 2023; Szymanski et al., 2025; Lin et al., 2025). These limitations motivate the need for more robust and efficient mechanisms for balancing exploration and exploitation in scalable algorithm discovery systems.

In this paper, we introduce **AlphaResearch**, an autonomous research-based coding agent that could reduce ineffective attempts and accelerate new al-

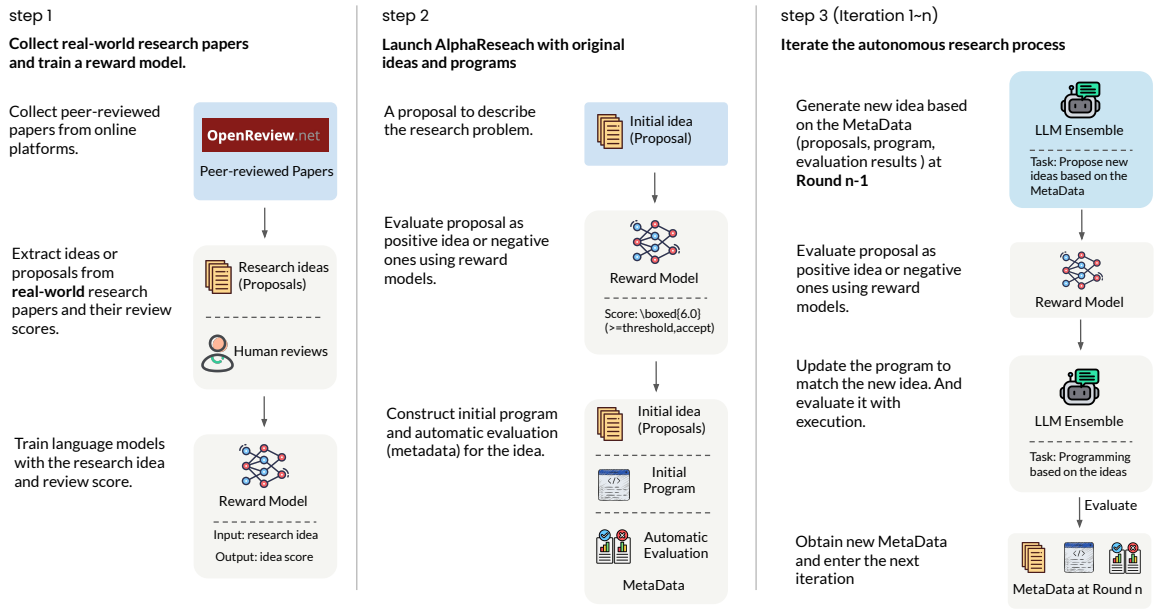

Figure 1: Overview of AlphaResearch. This system accelerates discovery process with interleaved idea generation and program generation, where it get reward from peer-review model and program execution.

gorithm discovery by interacting with real-world peer-review environments. As shown in Figure 1 and Table 1, AlphaResearch construct a novel dual research-based environment, where the research ideas proposed by LLMs could be judged by a simulated real-world peer-review environment and then attempt by code sandbox. Specifically, we (1) train a reward model **AlphaResearch-RM-7B** with real-world peer-review records, addressing the limitation of prior coding-only approaches that lack real-world research feedback, and use it to score the fresh ideas generated by LLMs; (2) construct an automatic program-based verifiable environment that executes these ideas with an interpreter. This dual environment facilitates a rigorous algorithm discovery process for autonomous research agents. As illustrated in Figure 1, AlphaResearch discovers new algorithms by iteratively running the following steps: (i) proposing new research ideas, (ii) verify the ideas in the dual research-based environment, and (iii) optimizing the proposals for higher reward from the environment. The synergy between an iterative real-world peer review environment and program-based verification empowers AlphaResearch to continuously explore novel research ideas and verify the attempts via less program execution.

To facilitate the comparison between AlphaResearch and other baselines (e.g., AlphaEvolve and ShinkaEvolve), we collect 8 open-ended research problems and their best-of-human records **AlphaResearchComp** and simulate an algorithm discovery **comp**etition between AlphaResearch and other agents for discovery. Our experimental results show that AlphaResearch achieves a score of 2.939 on the *"Packing Circles (n=32)"* problem, surpassing the previous results obtained by AlphaEvolve (see Appendix D). Furthermore, compared with ShinkaEvolve, which adopts code-novelty-based rejection sampling, AlphaResearch demonstrates faster convergence to higher-performing programs on three out of four open-ended problems. This highlights the effectiveness of the peer-review environment constructed in AlphaResearch. We also analyze the advantages and remaining challenges of autonomous research agents for knowledge discovery, providing insights for future work.

To summarize, our key contributions are the following:

- We introduce AlphaResearch, a novel autonomous research agent designed to accelerate the discovery of new algorithms by synergistically combining idea generation, code implementation, and verification in a dual environment of simulated peer review and executable program validation.
- We benchmark AlphaResearch with other agents for discovery across diverse verifiable problems to showcase the advantages of AlphaResearch in reducing ineffective attempts and accelerating new algorithm discovery.

Agent	Environments		
	Sandbox	Database	Peer-review Feedback
SWE-agent (Yang et al., 2024)	✔	✘	✘
AlphaEvolve (Novikov et al., 2025)	✔	✔	✘
ShinkaEvolve (Lange et al., 2025)	✔	✔	✘
AlphaResearch (ours)	✔	✔	✔

Table 1: Comparison of AlphaResearch and other agents.

- We present systematic ablations and analysis to understand the importance of incorporating peer-review environments into LLM-based autonomous discovery system.

2 Related Work

LLMs for New Ideas. Several recent works explored methods to improve research idea generation, such as iterative novelty refinement (Wang et al., 2024; Baek et al., 2024). These works focus on improving the research idea over vanilla prompting but critically miss an effective verification method. To promote more reliable AI-generated research ideas, many studies have proposed solutions from different perspectives, such as comparisons with any human expert (Si et al., 2024), using LLMs for executing experiments by generating code with human-curated research problems (Huang et al., 2024; Tian et al., 2024), and executing LLM-generated research ideas with LLM-generated programs (Li et al., 2024; Lu et al., 2024; Aygün et al., 2025). These works either use automatic program evaluation or unverifiable LLM evaluator method, which presents a challenge for their scalability to real-world advanced algorithm discovery. Our AlphaResearch presents a more feasible direction by combining program execution with RM training from real-world peer-reviewed research records.

LLMs for Code Generation. In autonomous research agents, code generation serves as a fundamental step. Previous models (Guo et al., 2024; Yu et al., 2023; Hui et al., 2024) and benchmarks (Chen et al., 2021; Yu et al., 2025) for code generation are in a longstanding pursuit of synthesizing code from natural language descriptions. SWE-Bench (Jimenez et al., 2024), PaperBench (Starace et al., 2025), MLE-Bench (Chan et al., 2024) introduces the problems in real-world agentic coding. Many studies on SWE-Bench have greatly contributed to the emergence of coding agents like SWE-Agent (Yang et al., 2024) and OpenHands (Wang et al., 2025). These agent frameworks greatly facilitate the training of agentic LLMs like Kimi-K2 (Team et al., 2025) and GLM-4.5 (Zeng et al., 2025). The surge of these models on SWE-Bench underscores a critical need to reassess the future directions of coding agent research.

3 AlphaResearch

3.1 Overview

AlphaResearch accelerates algorithm discovery process by continuously optimizing the research outcome from the dual reward that synergizes rigorous program verification and a real-world peer review environment. As shown in Figure 1, given initial idea i_0 and program p_0, AlphaResearch runs the program p_0 with execution, producing r_0, which represents the initial overall rating. The triplet (i_0, p_0, r_0) will be fed to AlphaResearch for subsequent processing, including newer idea generation, code implementation, and program-based execution. When reaching a point where execution output r_n surpasses the previous rating, AlphaResearch will save the triplet $(i_{best}, p_{best}, r_{best})$ as the best record. We repeat the process until r_{best} surpasses the best-of-human score, or the maximum round is reached. The resulting trajectory is denoted as $\tau = i_0 p_0 r_0 ... i_{n-1} p_{n-1} r_{n-1} i_n p_n r_n$, where n is the total rounds.

3.2 Actions

New Idea Generation. For each step k, AlphaResearch start with generating a new idea i_k based on a sampled previous step (i_t, p_t, r_t) from previous trajectory $\tau_{k-1} = i_0 p_0 r_0 ... i_{k-1} p_{k-1} r_{k-1}$. This process can be denoted as:

$$i_k \sim \mathbb{P}_{\mathcal{A}}(\cdot | i_t \oplus p_t \oplus r_t) \quad (1)$$

where $\oplus$ means concatenation, t is the sampled step from trajectory τ_{i-1} and $\mathbb{P}_{\mathcal{A}}()$ indicates random sampling. We use a reward model to select

high-quality ideas overall. If $\mathcal{RM}(i_n)$ outputs a negative score, we cease the subsequent actions in this round.

Program-based Verification. After obtain the fresh idea, AlphaResearch generates new program p_k based on the previous implementation p_t and new idea i_k next:

$$p_k \sim \mathbb{P}_{\mathcal{A}}(\cdot|p_t \oplus i_k) \tag{2}$$

and yield the evaluation result r_k by verifying p_k with code executor $r_k \leftarrow \mathcal{E}(p_k)$. Then, we update the trajectory τ_k with the newly generated idea i_k, program p_k and result r_k:

$$\tau_k \leftarrow \tau_{k-1} \oplus i_k \oplus p_k \oplus r_k \tag{3}$$

We repeat the above interaction process until k reaches the maximum rounds n and get the best result $(i_{best}, p_{best}, r_{best})$ as final output.

3.3 Environment

3.3.1 Reward from Real-world Research Records

Existing autonomous idea generation process suffers from a trade-off where highly novel research ideas may lack feasibility (Guo et al., 2025; Si et al., 2025). To address this gap and ensure the feasibility of idea candidates, we train a reward model with ideas from real-world peer-review information to simulate the real-world peer-review environment.

Split	Train	Test
Data Source	ICLR	ICLR
Range of Date	2017∼2024	2025
Environment Nums	24,445	100
Start Date	2016-11	2024-10
End Date	2023-12	2024-12

Table 2: Dataset for reward model training. We use the end of author-reviewer rebuttal period as the latest knowledge date.

Dataset for reward model. To train RM to identify good ideas, we collect all ICLR peer review records from 2017 to 2024 as our training set. We sample a subset of ICLR 2025 records as a test set, where the dates of train and test are disjoint, which prevents knowledge contamination between the train and test split. We also select Qwen2.5-7B-Instruct[1] as our base model. For each record, we extract the abstract part as RM input and wrap the average peer-review overall ratings with `\boxed{}` as RM output. We fine-tune the model with RM pairs, yielding the AlphaResearch-RM-7B model.

[1]Its release date `2024-09` is earlier than the ICLR 2025 author-reviewer rebuttal period `2024-10`.

RM threshold. To simplify the RM evaluation, we binarize the RM output score according to the ICLR Reviewer Guide, where overall rating > 5.5 records are regarded as a positive score and ≤ 5.5 records are negative. We do not treat the RM threshold as a hyperparameter in this work, as the score carries real-world interpretability where it corresponds to the midpoint between the acceptance and rejection scores at ICLR. We compute the binary classification accuracy and evaluate three models (GPT-5, Qwen2.5-Coder-Instruct, and AlphaResearch-RM-7B) on the AlphaResearch-RM test set.

Reward Model	Cutoff	Acc
Random (theoretical)	-	50.0%
Human Annotator	-	65.0%
GPT-5 (medium)	2025-08	53.0%
Qwen2.5-7B-Instruct	2024-09	37.0%
AlphaResearch-RM-7B	2024-09	72.0%

Table 3: Evaluation results of different RMs. We use the more recent date between the model release date and the dataset cutoff as the latest date.

Can LLMs identify good ideas? To establish a human annotator baseline, we select 3 researchers with relevant backgrounds who have published papers and served as a reviewer on their assigned topics. Table 3 presents the evaluation results that eliminate the knowledge contamination, highlighting the following observations: (1) Both GPT-5 and Qwen2.5-7B-Instruct achieve lower than 60% accuracy when identifying the good ideas from ICLR 2025 records. (2) After being fine-tuned with ideas from previous ICLR peer-review information, AlphaResearch-RM-7B demonstrates 72% binary classification accuracy on unseen ICLR 2025 ideas, significantly outperforming baseline models and human annotators. Based on these observations, we use the fine-tuned AlphaResearch-RM-7B as the final RM to simulate a real-world peer-review environment and filter out good ideas generated by AlphaResearch.

Algorithm 1 AlphaResearch

Require: initial idea i_0, initial program p_0, initial result r_0, model $\mathcal{A}$, evaluation program $\mathcal{E}(\cdot)$, maximum iteration rounds n,

$\tau_0 \leftarrow (i_0, p_0, r_0)$, $r_{best} = 0$ ▷ Initialization
for $k = 1$ to n do **do**
  $(i_t, p_t, r_t) \sim \mathbb{P}(\cdot|\tau_{k-1})$ ▷ States Sampling
  $i_k \sim \mathbb{P}_{\mathcal{A}}(\cdot|i_t \oplus p_t \oplus r_t)$ ▷ New Idea Generation (Eq. 1)
  if $\mathcal{RM}(i_k)$ < threshold **then**
    continue ▷ Reward Model for New Idea
  end if
  $p_k \sim \mathbb{P}_{\mathcal{A}}(\cdot|p_t \oplus i_k)$ ▷ Program Generation (Eq. 2)
  $r_k \leftarrow \mathcal{E}(p_k)$ ▷ Program-based Execution
  if $r_k > r_{best}$ **then**
    $(i_{best}, p_{best}, r_{best})$ = (i_k, p_k, r_k)
  end if
  $\tau_k \leftarrow \tau_{k-1} \oplus i_k \oplus p_k \oplus r_k$ ▷ Trajectory Update (Eq. 3)
end for
return $(i_{best}, p_{best}, r_{best})$

3.3.2 Reward from Program-based Execution

We construct an automatic evaluation process with a code executor where each new program p_k generated by AlphaResearch will be captured and evaluated. The evaluation program $\mathcal{E}(\cdot)$ includes two modules: (i) **Verification** module that validates whether p_k conforms to the problem constraints. (ii) **Measurement** module that output the score r_k of program performance. The program output r_k will be injected into the idea generation prompt (if sampled), thereby participating in the optimization process for fresh ideas. These programs and results are stored in a candidate pool, where the primary goal is to optimally resurface previously explored ideas in future generations. The verifiable reward by code executor significantly simplifies the action spaces of AlphaResearch, thereby enhancing the efficiency of the discovery process.

4 Experiments

4.1 Setup

We select `o4-mini`, a strong but cost-efficient LLM as our research agent and run AlphaResearch on each problem to get the best algorithm. We perform supervised finetuning on `Qwen-2.5-7B-Instruct` (Yang et al., 2025) with the collected ICLR records, yielding AlphaResearch-RM-7B. We do not compute loss on paper information, only on the average rating scores within `\boxed{}`. For fine-tuning hyperparameters, we train our model with a learning rate of 1e-5 warmed up linearly for 100 steps. We train all the models in bfloat16 precision with Pytorch Fully Shard Data Parallel (FSDP) and set a global batch size to 128 for 2 epochs. All other settings not mentioned in this paper follow the default values of Huggingface Trainer [2]. Due to the unavailability of the AlphaEvolve codebase, we adopted OpenEvolve and ShinkaEvolve as our baseline approaches.

4.2 Evaluation

Problem collection. We curate AlphaResearch-Comp, a set of frontier program-based research tasks including geometry, number theory, harmonic analysis, and combinatorial optimization. These problems were selected based on the following principles: (1) Each task has a precise mathematical formulation with an objective function that admits rigorous automatic evaluation. (2) For every problem, we provide the best-known human result from the literature. These represent conjectured best-known values rather than proven optima, ensuring ample room for further improvement. The curated problems are either inherited from prior work (e.g., AlphaEvolve) or collected from online repositories and domain experts. Each problem is supported by verifiable resources in the corresponding field. This design enables AlphaResearch to demonstrate both the *reproducibility* of established mathematical results and the *potential for discovery* beyond current human-best achievements.

4.3 Main Results

Figure 2 presents the comparison of AlphaResearch with OpenEvolve and ShinkaEvolve, highlighting the following observations: (1) With interleaved idea generation and program generation, AlphaResearch consistently surpasses OpenEvolve across the first four open-ended problems, which demonstrates the effectiveness of incorporating

[2] https://huggingface.co/docs/transformers/main_classes/trainer

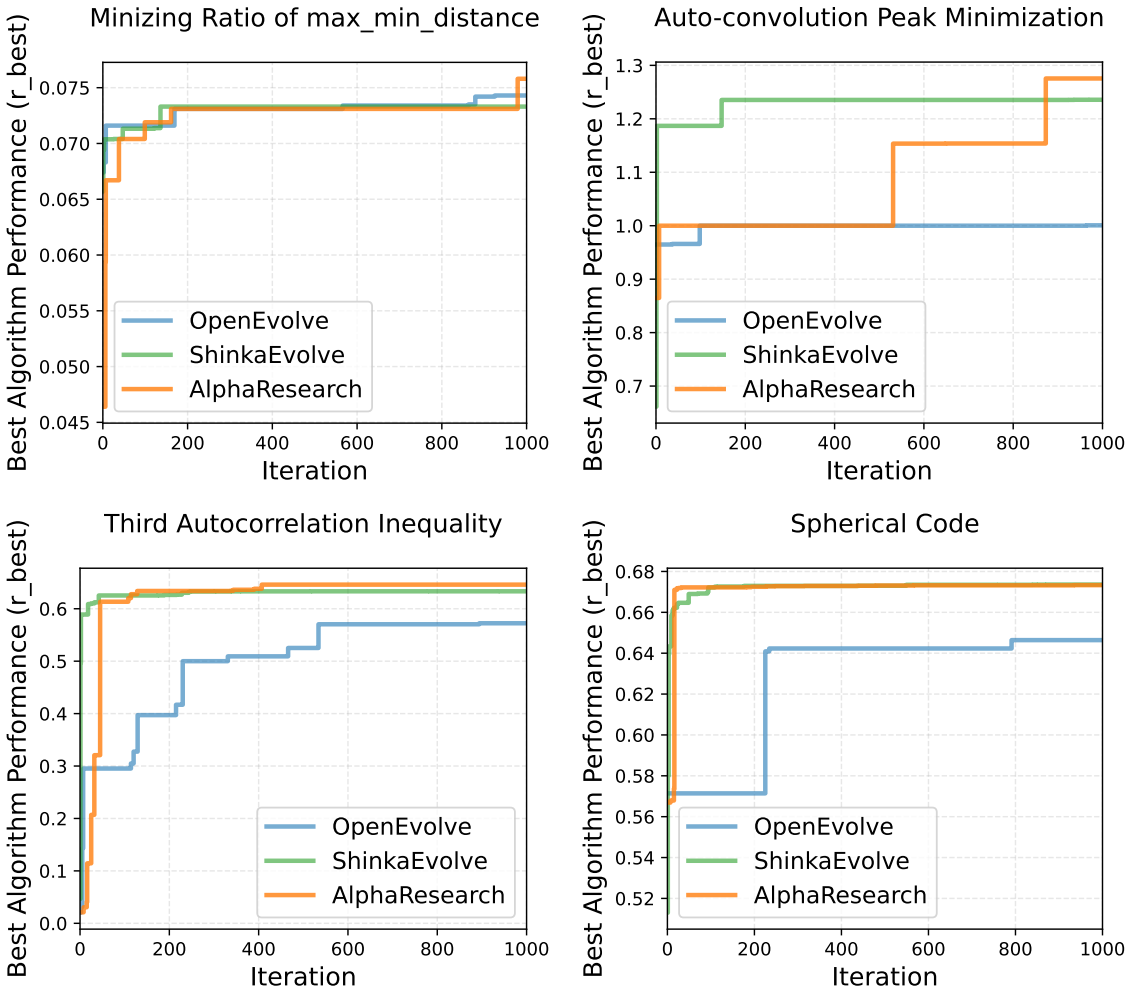

Figure 2: Comparison of AlphaResearch, OpenEvolve, and ShinkaEvolve throughput the 1000 step discovery process.

peer-review environments into autonomous discovery process. (2) Compared to ShinkaEvolve, which employs code novelty-based rejection sampling to reduce the number of attempts, AlphaResearch surpasses it on three out of four tasks within the first 1,000 iterations. Although the initial performance gains are modest, these advantages tend to accumulate into more substantial improvements as the number of iterations increases. This highlights the superiority of the peer-review environment over approaches that rely solely on code novelty-based rejection sampling.

4.4 Ablations and Analysis

Successful Cases. Table 4 presents the results of AlphaResearchComp on 8 algorithms discovery problems. AlphaResearch achieves a 2/8 win rate ($\Delta > 0$) against human researchers, with one notable success: the algorithm discovered by AlphaResearch for *"Packing Circles"* problem reaches the best-of-known performance (2.636 for n=26, 2.939 for n=32), outperforming human researchers (2.634 for n=26, 2.936 for n=32) and AlphaEvolve (2.635 for n=26, 2.937 for n=32), where the case (n = 32) is shown in Figure 9.

Failure Cases. Although AlphaResearch has exhibited continuous growth, it still underperforms human researchers on the *"Littlewood polynomials"* and *"MSTD(n=30)"* tasks, with no observable improvement in program score throughout the discovery process. To further analyze this limitation, we compare the performance of AlphaResearch with that of OpenEvolve and ShinkaEvolve on the failure modes. As shown in Figure 10, All of AlphaResearch, OpenEvolve, and ShinkaEvolve fail to improve on the Littlewood polynomials and MSTD (n=30) problems, which indicates that current large language models still face significant challenges in reliably discovering superior algorithms.

Problem	Human Researcher		AlphaResearch		Δ(%)
	research record	*baseline*	*init*	*best*	
packing circles (n=26) ↑	D. Cantrell (2011)	2.634	0	2.636	0.002
packing circles (n=32) ↑	E. Specht (2012)	2.936	0	2.939	0.003
minimizing max-min distance ratio ↓	D. Cantrell (2009)	12.89	15.55	12.92	- 0.03
third autocorrelation inequality ↓	C. Vinuesa (2009)	1.458	35.746	1.546	-0.088
spherical code (d=3, n=30) ↑	Hardin & Sloane (2002)	0.6736	0.5130	0.6735	-0.0001
autoconvolution peak minimization ↓	Matolcsi & Vinuesa (2010)	0.755	1.512	0.756	-0.001
littlewood polynomials (n=512) ↓	Rudin & Shapiro (1959)	32	32	32	0
MSTD (n=30) ↑	Hegarty (2007)	1.04	1.04	1.04	0

Table 4: Results on AlphaResearchComp. ↑ indicates that higher score is better and ↓ for lower. Δ indicates the performance gap between best of AlphaResearch and human baseline.

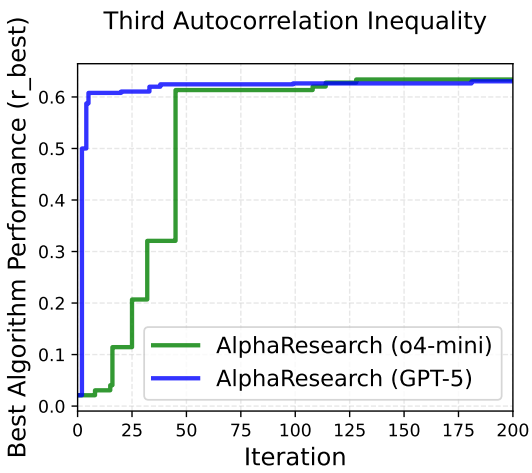

Figure 3: Comparison between different frontier LLMs in AlphaResearch.

Impact of different LLM backbone. To compare the impact of different LLM backbones on AlphaResearch, we used GPT-5 and o4-mini to run AlphaResearch for 200 steps on the "The Autocorrelation Inequality" problem, respectively. As illustrated in Figure 3, AlphaResearch (GPT-5) achieves strong performance much more rapidly than o4-mini during the early stages of discovery. However, in later stages, the two models exhibit comparable performance, suggesting that their underlying capabilities on the algorithm discovery task are similar.

Ablations on real-world peer-review environment. To assess the effectiveness of reward from a simulated real-world peer-view environment, we ablate AlphaResearch-RM-7B at the first 400 iterations on *"Packing Circles"* problem. Figure 4 presents the execution results of w/ and w/o AlphaReasearch-RM-7B during the discovery process. Compared to the baseline without RM, AlphaResearch-RM-7B successfully filtered 151 ideas below the threshold. This process yielded 108 correct rejections of execution failures while making 43 erroneous rejections of viable ideas. AlphaResearch attained an accuracy of 71.5% (108/151), a result that aligns closely with its performance on the AlphaResearch-RM test set, as shown in Table 3. This outcome effectively demonstrates the model's generalization capabilities and the efficacy of incorporating feedback from a simulated real-world peer-review environment.

Analysis of the discovery process. We analyze the rejection distribution in AlphaResearch discovery process. As shown in Figure 5, approximately 30%∼40% of newly proposed ideas fall below the RM threshold and are thus discarded. The remaining ideas are executed, with the success rate of execution largely depending on the inherent characteristics of the problems. For example, the execution success rate on *"Packing Circles"* problem is 28.9%, whereas it reaches 51.7% on the *"Third Autocorrelation Inequality"* problem. Figure 6 illustrates the execution-based rewards for these two examples in AlphaResearch. Despite the substantial variations in execution success rates, the execution-based rewards in both cases exhibit a consistent increasing trend. These findings demonstrate the interactions between LLM-based autonomous research agents and real-world environments.

4.5 Case Study

We select the successful example from AlphaResearch to better understand the discovery process. We'll consider the problem *"Packing Circles"* where the goal is to pack n disjoint circles inside a unit square so as to maximize the sum of their radii, shown in Figure 8. We first initialize

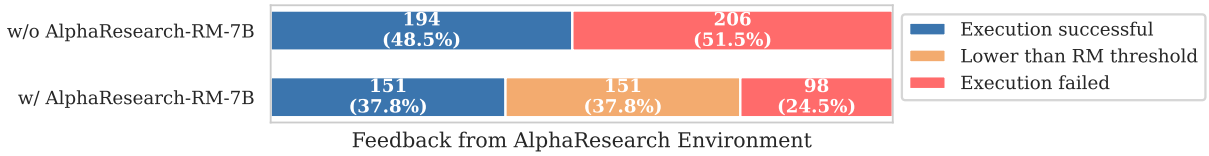

Figure 4: The impact of real-world peer review environment on execution results. AlphaResearch-RM-7B filters 151 bad ideas, where 108 ideas fail to execute and 43 are successful.

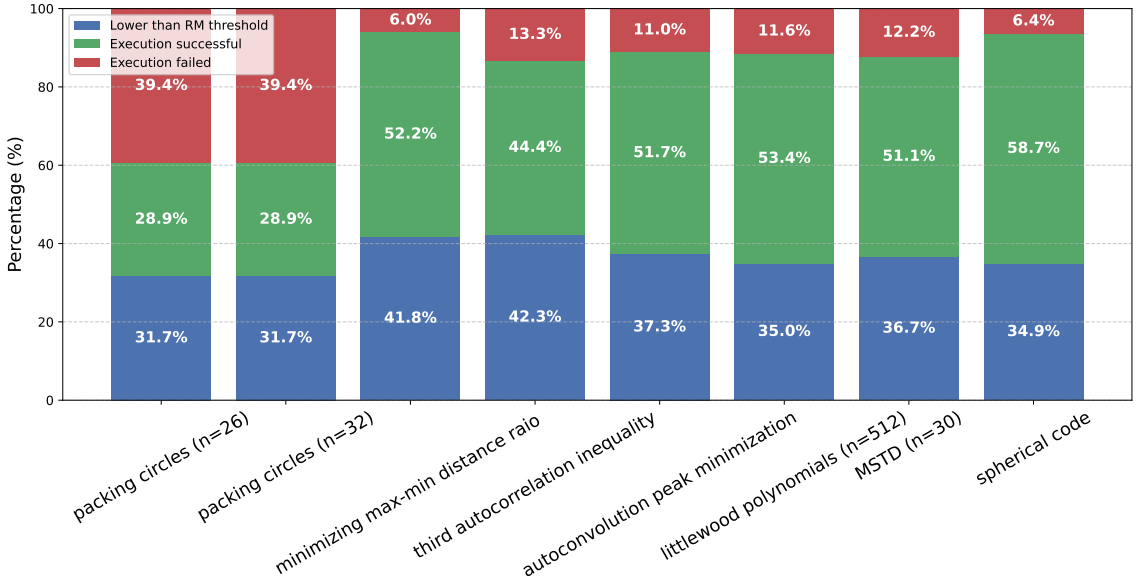

Figure 5: Reward overview during the discovery process. Each action in AlphaResearch will obtain 3 kinds of reward: (1) idea scrapping due to a lower RM score than the threshold (2) idea execution successes (3) idea execution fails.

AlphaResearch with an original research proposal and a related program that returns a list of circles (x, y, r) as output. The verification program first employs `verify_circles` function to check if the outputs of the initial program meet the problem constraints (e.g., all circles are inside a unit square) and `evaluate` function to output the sum of their radii. The metadata, including: (1) research ideas, (2) programs, (3) execution results, are subsequently preserved as candidates which represent the end of one step. At the next step, AlphaResearch will sample from the candidate pool and generate a new idea to improve the research proposals from the sampled metadata. After generating the new research ideas, AlphaResearch will further generate a patch to modify the existing program if the idea obtains a positive score from AlphaResearch-RM. The new program is then evaluated by the same verification program, thereby generating new metadata. We select the best program and idea as the final solution of AlphaResearch in this iterative process.

5 Conclusion

We present AlphaResearch, an autonomous research-oriented coding agent that synergistically combines new idea generation with program-based verification for autonomous algorithm discovery. To accelerate the discovery process of AlphaResearch, we construct a dual research-based environment to reduce ineffective attempts and runtime,. On our collected 8 open-ended algorithmic problems, AlphaResearch outperforms AlphaEvolve for 2/8 algorithmic problems and demonstrates better discovery ability than previous state-of-the-art evolutionary agents, which demonstrates the effectiveness of interleaved idea generation and program generation and constructed peer-review environments in AlphaResearch. Furthermore, our systematic analysis of incorporating autonomous review feedback into autonomous discovery systems provides valuable insights for future research, contributing to the development of more advanced and versatile agentic discovery systems.

Limitations

While AlphaResearch demonstrates promising results in using a reward model (RM) to guide scientific idea search and generation, several important limitations should be acknowledged. First, the AlphaResearch-RM is trained exclusively on ICLR peer-review records, which are heavily concentrated in machine learning and related topics. This creates a potential topic bias, where the model may perform less reliably when evaluating ideas outside core ML domains. Future work will explore training on more diverse and extensive peer-review datasets across broader scientific fields to mitigate this bias. Second, there exists a fundamental mismatch between peer-review scores (which the RM is trained to predict) and the actual downstream usefulness or impact of a research idea. Peer review primarily assesses perceived novelty, technical soundness, and clarity at the time of submission, but does not necessarily reflect long-term scientific value, feasibility of execution, or practical utility. Therefore, optimizing for RM scores may not fully align with optimizing for genuinely useful or groundbreaking research. Third, the current benchmark scope remains relatively narrow and partially inherited from existing datasets. Most evaluations focus on standard machine learning tasks and benchmarks, which may not fully capture the challenges of open-ended scientific discovery in more complex or interdisciplinary settings. We believe addressing these limitations represents important directions for future research in building more reliable and general-purpose AI systems for scientific discovery.

References

Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston Anton Kast, Cory Y McLean, Peter Norgaard, Zahra Shamsi, and 1 others. 2025. An ai system to help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. 2024. ResearchAgent: Iterative Research Idea Generation over Scientific Literature with Large Language Models. *ArXiv*, abs/2404.07738.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Sikun Guo, Amir Hassan Shariatmadari, Guangzhi Xiong, Albert Huang, Myles Kim, Corey M Williams, Stefan Bekiranov, and Aidong Zhang. 2025. Ideabench: Benchmarking large language models for research idea generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5888–5899.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *ICML*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*.

Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. 2024. MLR-Copilot: Autonomous Machine Learning Research based on Large Language Models Agents. *ArXiv*, abs/2408.14033.

Ethan Lin, Zhiyuan Peng, and Yi Fang. 2025. Evaluating and enhancing large language models for novelty assessment in scholarly publications. In *Proceedings of the 1st Workshop on AI and Scientific Discovery: Directions and Opportunities*, pages 46–57.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, and 1 others. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

OpenAI. 2025. Gpt-5.

Chenglei Si, Tatsunori Hashimoto, and Diyi Yang. 2025. The ideation-execution gap: Execution outcomes of llm-generated versus human research ideas. *arXiv preprint arXiv:2506.20803*.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. 2024. Can llms generate novel research ideas.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, and 1 others. 2025. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*.

Annalisa Szymanski, Noah Ziems, Heather A Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A Metoyer. 2025. Limitations of the llm-as-a-judge approach for evaluating llm outputs in expert knowledge tasks. In *Proceedings of the 30th international conference on intelligent user interfaces*, pages 952–966.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, and 11 others. 2024. SciCode: A Research Coding Benchmark Curated by Scientists. *ArXiv*, abs/2407.13168.

Qingyun Wang, Doug Downey, Heng Ji, and Tom Hope. 2024. Scimon: Scientific inspiration machines optimized for novelty. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 279–299.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. 2023. Wavecoder: Widespread and versatile enhancement for code large language models by instruction tuning. *arXiv preprint arXiv:2312.14187*.

Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. 2025. HumanEval pro and MBPP pro: Evaluating large language models on self-invoking code generation task. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13253–13279, Vienna, Austria. Association for Computational Linguistics.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

A The Use of Large Language Models

During the preparation of this manuscript, we utilized large language models (LLMs) for grammar checking and writing suggestions to enhance the readability and clarity of the content.

B Other Analysis

LLMs can refine their research ideas autonomously. AlphaResearch discovers advanced algorithms by iteratively proposing and verifying new research ideas. As shown in Table 3, 6/8 problems demonstrate consistent improvement throughout the discovery process. Figure 6 presents two examples of the reward trend in AlphaResearch, where the execution-based reward initially grows rapidly, then slowly plateaus for optimal performance seeking. This improvement trend emphasizes the autonomous discovery ability of research agents.

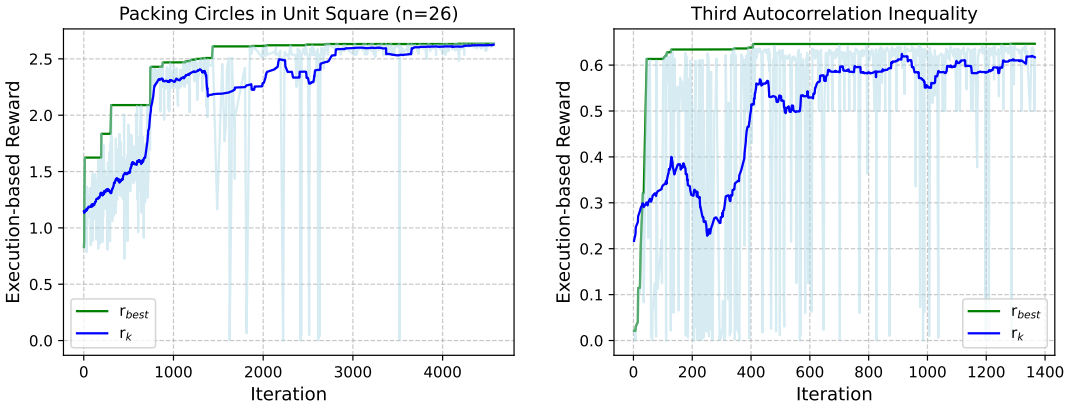

Figure 6: Execution-based reward of AlphaResearch on packing circles (n=26) problem (left) and third autocorrelation inequality problem (right).

Execution-only agent against AlphaResearch. To compare AlphaResearch with execution-only agents, we utilize AlphaResearch-RM-7B to evaluate the novelty of ideas generated by the execution-only agent and ideas produced by AlphaResearch. As illustrated in Figure 7, the ideas generated by AlphaResearch generally achieve higher scores than execution-only research agents. This illustrates that AlphaResearch tends to generate better ideas to get higher external rewards, thus facilitating a more effective research optimization process.

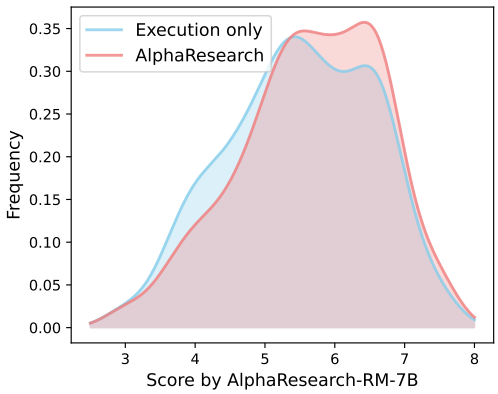

Figure 7: The idea comparison between the execution-only research agent and AlphaResearch, where AlphaResearch-RM-7B is used. This is done between the full distribution of all 1000 generated ideas from both agents without filtering.

C Case Study

Initial Research Idea

The program presents a computational approach to the circle packing problem within a unit square, aiming to maximize the sum of radii for a given number of circles. The pack_circles function initiates a structured placement of 26 circles: one at the center, eight in an inner ring, and sixteen in an outer ring. While this initial arrangement is a predefined pattern, it serves as a foundation for further optimization. The core of the algorithm lies in the compute_max_radii function, which iteratively determines the largest possible radius for each circle. This is achieved by first constraining radii based on proximity to the unit square's boundaries and then adjusting them to prevent overlap between any pair of circles. Overlapping circles have their radii proportionally scaled down to ensure non-intersection, effectively pushing them to a just-touching state. The final output provides the optimized centers, radii, and the total sum of radii for the packed configuration, demonstrating a method for generating dense circle arrangements within a confined space.

Initial Program

tmp/packing_circles/initial_program.py

```python
def pack_circles(n = 26):
  """
  Construct a specific arrangement of 26 circles in a unit square
  that attempts to maximize the sum of their radii.
  """
  ...
  return sum_radii, circles

def compute_max_radii(centers):
  """
  Compute the maximum possible radii for each circle position
  such that they don't overlap and stay within the unit square."""...
```

Verification Program

tmp/packing_circles/evaluator.py

```python
def verify_circles(circles):
  """Checks that the circles are disjoint and lie inside a unit square.  """
...

def evaluate(program_path: str = "results/initial_program.py"):
  """
  Evaluate the pack_circles function from the given program file.
  Returns the total radius sum if valid, otherwise raises an exception.
  """
...
  return {"pack_result_26": total_r_26}
```

Execution Results

{"pack_result_26": 0.864}

New Idea Generation

Instruction

You are a research advisor tasked with improving research proposals.
Your goal is to generate a new research proposal that builds upon the current research idea while addressing its limitations and incorporating insights from successful approaches.

MetaData

Current Research Idea
The program presents a computational approach to the circle packing problem within a unit square, aiming to maximize the sum of radii for a given number of circles.
Current Program
```python
def pack_circles(n = 26):
  """
  Construct a specific arrangement of 26 circles in a unit square
  that attempts to maximize the sum of their radii
  """...
```
Current Metrics
packing_result_26=0.864

Program Generation

Instruction

You are an expert software developer tasked with iteratively improving a codebase.
Your job is to analyze the current program and suggest improvements based on feedback from previous attempts.Focus on making targeted changes that will increase the program's performance metrics.

MetaData

Current Research Idea
The program presents a computational approach to the circle packing problem within a unit square, aiming to maximize the sum of radii for a given number of circles.
Current Program
```python
def pack_circles(n = 26):
  """
  Construct a specific arrangement of 26 circles in a unit square
  that attempts to maximize the sum of their radii
  """... ```
### Current Metrics
packing_result_26=0.864
### New Research Idea
We propose a novel computational framework to improve the dense packing of n variable-radius circles within a unit square, targeting maximization of the total sum of radii. Our approach addresses key limitations in existing methods—namely, rigid initialization patterns, local-only radius adjustment, and O($n^2$) pairwise scaling...

Format

You MUST use the exact SEARCH/REPLACE diff format shown below to indicate changes:
<<<<<<< SEARCH
# Original code to find and replace (must match exactly)
=======
# New replacement code
>>>>>>> REPLACE

Figure 8: We show an example of a formatted task of AlphaResearch.
```

D Examples of Packing Circles

We show an example of the constructions discovered by AlphaResearch on problem *"Packing Circles"* in this section.

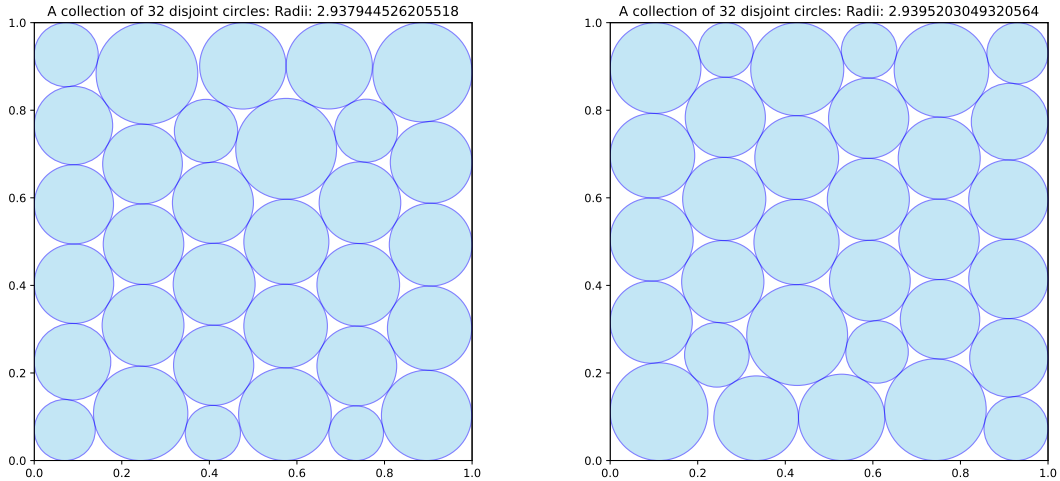

Figure 9: New construction of AlphaResearch (right) improving the best known AlphaEvolve (right) bounds on packing circles to maximize their sum of radii. Left: 32 circles in a unit square with sum of radii ≥ 2.9379. Right: 32 circles in a unit square with sum of radii ≥ 2.9395.

E Example of Failure Cases

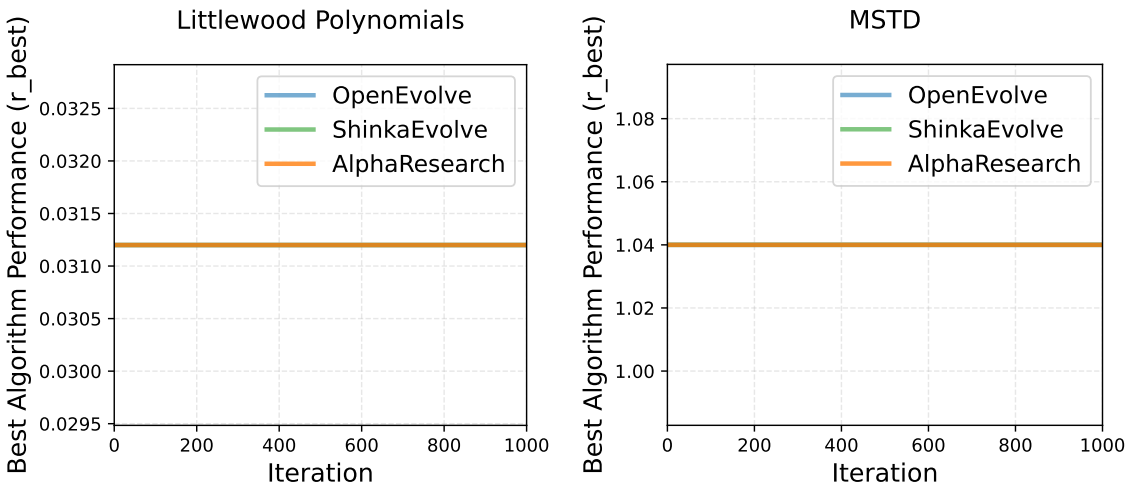

Figure 10: Performance comparison of AlphaResearch, OpenEvolve, and ShinkaEvolve in terms of failure modes of AlphaResearchComp.