

# 000 001 AlphaResearch: ACCELERATING NEW ALGORITHM 002 DISCOVERY WITH LANGUAGE MODELS 003 004

005 **Anonymous authors**  
006 Paper under double-blind review  
007  
008  
009

## 010 ABSTRACT 011

012 Large language models have made significant progress in complex but easy-to-  
013 verify problems, yet they still struggle with discovering the unknown. In this  
014 paper, we present **AlphaResearch**, an autonomous research agent designed to dis-  
015 cover new algorithms on open-ended problems by iteratively running the follow-  
016 ing steps: (1) propose new ideas (2) program to verify (3) optimize the research  
017 proposals. To synergize the feasibility and innovation of the discovery process,  
018 we construct a new reward environment by combining the execution-based verifi-  
019 able reward and reward from simulated real-world peer review environment. We  
020 construct **AlphaResearchComp**, a new evaluation benchmark that includes an  
021 eight open-ended algorithmic problems competition, with each problem carefully  
022 curated and verified through executable pipelines, objective metrics, and repro-  
023 ducibility checks. AlphaResearch gets a 2/8 win rate in head-to-head comparison  
024 with human researchers. Notably, the algorithm discovered by AlphaResearch on  
025 the “*packing circles*” problem achieves the best-of-known performance, surpass-  
026 ing the results of human researchers and strong baselines from recent work (e.g.,  
027 AlphaEvolve). Additionally, we conduct a comprehensive analysis of the bene-  
028 fits and remaining challenges of autonomous research agent, providing valuable  
029 insights for future research.  
030

## 031 1 INTRODUCTION 032

033 Recent progress has shown that frontier LLMs like GPT-5 (OpenAI, 2025) and Gemini 2.5 (Co-  
034 manici et al., 2025) could achieve expert-level performance in complex tasks such as mathemat-  
035 ics (Trinh et al., 2024; Lin et al., 2025) and programming (Jimenez et al., 2024; Jain et al., 2025).  
036 While LLMs excel at processing and reasoning on problems that are within the boundary of ex-  
037 isting human knowledge (Wang et al., 2024b; Phan et al., 2025), their capacity for independent  
038 discovery that pushes the boundaries of human knowledge still remains a question of paramount  
039 importance (Novikov et al., 2025). *Can these models create advanced knowledge or algorithms that*  
040 *surpass human researchers?*

041 Previous studies demonstrate that LLMs can generate novel ideas at a human expert level (Si et al.,  
042 2024; Wang et al., 2024a). However, the outcome evaluation of LLM-generated research ideas still  
043 struggles with biased verification methods (Ye et al., 2024) that constrain the exploration of out-of-  
044 boundary machine knowledge, such as LLM-as-a-judge (Lu et al., 2024), where misaligned LLMs  
045 are used to evaluate fresh ideas and inevitably favor solutions within existing knowledge bound-  
046 aries. Furthermore, the ideation–execution gap (Si et al., 2025) between generating and executing  
047 new ideas also hinders models from producing advanced research outcomes. Moreover, prior at-  
048 tempts at autonomous algorithm discovery face a fundamental tension. Execution-based verification  
049 systems like AlphaEvolve Novikov et al. (2025) can rigorously validate whether code runs and  
050 meets constraints, but this verification alone might not be completely sufficient for discovery. For  
051 example, these systems could converge on technically correct but scientifically uninteresting or less  
052 impactful solutions—code that executes successfully yet offers no advancement over existing meth-  
053 ods. Conversely, idea-generation systems evaluated purely by LLM judges can propose innovative  
054 concepts that prove computationally infeasible or violate problem constraints when implemented.  
055 The absence of real-world research environment rewards in execution-based agents and execution-

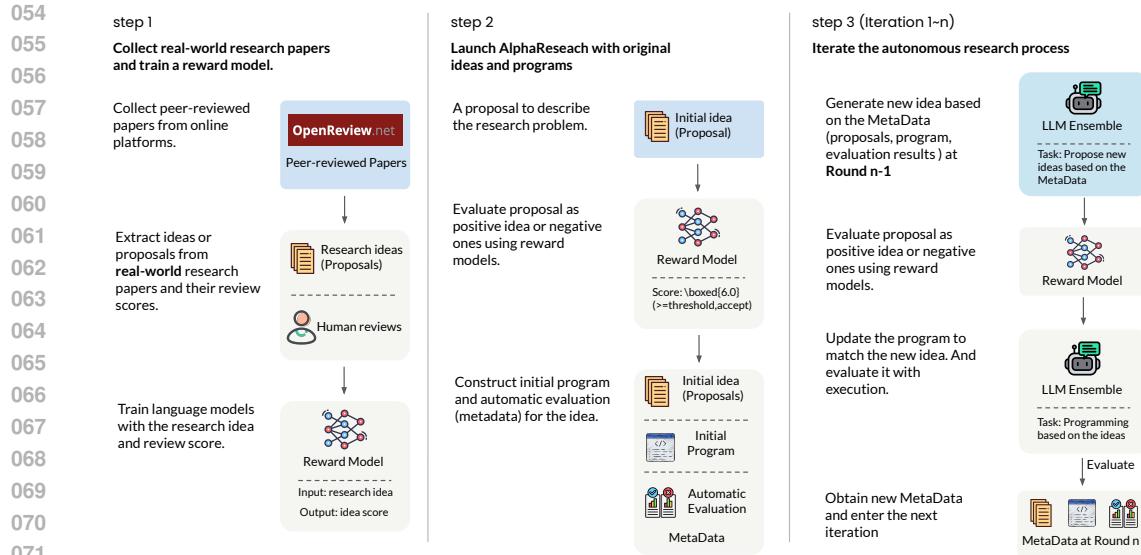


Figure 1: The launch of AlphaResearch contains two steps. (1) Train reward models with real-world peer-reviewed records. (2) Prepare initial research proposals, initial programs and evalution program. AlphaResearch will refine the research proposals and programs autonomously.

based reward in idea-generation systems renders the discovery of new knowledge and algorithms challenging for current autonomous research agents (Tian et al., 2024).

To combine the feasibility and innovation of the algorithm discovery process, we introduce **AlphaResearch**, an autonomous research agent that could discover new advanced algorithms with a suite of research skills including idea generation and code implementation that could interact with the environment. To synergize these research skills during the discovery process, we construct a novel dual research-based environment (Tian et al., 2024), where novel insights are forged by the simulated real-world peer-reviewed environment and execution-based verification. We use this dual environments to accelerate the discovery process because many research ideas can be evaluated before even implementing and executing on the idea, based on factors such as novelty, literature and the knowledge used. Specifically, we (1) train a reward model **AlphaResearch-RM-7B** with real-world peer-reviewed 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 them via program execution. Once the generated optimal program surpasses current human-best achievements, these validated novel ideas could form feasible algorithms, thereby pushing the boundaries of human research forward.

To compare AlphaResearch with human researchers on novel algorithm discovery, we construct **AlphaResearchComp**, a simulated discovery competition between research agents and human researchers, by collecting 8 open-ended research problems and their best-of-human records (shown in Appendix I). Our results demonstrate that AlphaResearch surpasses human researchers on two problems but fails on the other six. The novel algorithms discovered by AlphaResearch not only surpass best-of-human performance but also significantly outperform the state-of-the-art results achieved by AlphaEvolve. Specifically, AlphaResearch optimizes the result of “*Packing Circles (n=32)*” problem to 2.939, where the goal is to pack  $n$  disjoint circles inside a unit square so as to maximize the sum of their radii, surpassing the results of best-of-human and previous SoTA results achieved

---

108 **Algorithm 1** AlphaResearch

---

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

111 1:  $\tau_0 \leftarrow (i_0, p_0, r_0)$ ,  $r_{best} = 0$  ▷ Initialization  
 112 2: **for**  $k = 1$  to  $n$  **do**  
 113 3:  $(i_t, p_t, r_t) \sim \mathbb{P}(\cdot | \tau_{k-1})$  ▷ States Sampling  
 114 4:  $i_k \sim \mathbb{P}_{\mathcal{A}}(\cdot | i_t \oplus p_t \oplus r_t)$  ▷ New Idea Generation (Eq. 1)  
 115 5: **if**  $\mathcal{RM}(i_k) < \text{threshold}$  **then**  
 116 6: **continue** ▷ Reward Model for New Idea  
 117 7: **end if**  
 118 8:  $p_k \sim \mathbb{P}_{\mathcal{A}}(\cdot | p_t \oplus i_k)$  ▷ Program Generation (Eq. 2)  
 119 9:  $r_k \leftarrow \mathcal{E}(p_k)$  ▷ Program-based Execution  
 120 10: **if**  $r_k > r_{best}$  **then**  
 121 11:  $(i_{best}, p_{best}, r_{best}) = (i_k, p_k, r_k)$  ▷ Trajectory Update (Eq. 3)  
 122 12: **end if**  
 123 13:  $\tau_k \leftarrow \tau_{k-1} \oplus i_k \oplus p_k \oplus r_k$   
 124 14: **end for**  
 125 15: **return**  $(i_{best}, p_{best}, r_{best})$

---

126 by AlphaEvolve (as shown in Appendix G). These entirely novel ideas and algorithms constitute  
 127 the most advanced solutions currently present in the human knowledge base, demonstrating the fea-  
 128 sibility of employing LLMs to advance the frontiers of human knowledge. The six failure modes  
 129 in AlphaResearchComp demonstrate the challenges for the autonomous algorithm discovery with  
 130 research agents. We analyze the benefits and remaining challenges of autonomous research agents  
 131 for knowledge discovery, providing valuable insights for future work.

## 2 ALPHARESEARCH

### 2.1 OVERVIEW

132 AlphaResearch discovers out-of-boundary novel algorithms by continuously optimizing the re-  
 133 search outcome from the dual reward that synergizes rigorous program verification and a simu-  
 134 lated real-world peer review environment. As shown in Figure 1, given initial idea  $i_0$  and pro-  
 135 gram  $p_0$ , AlphaResearch runs the program  $p_0$  with execution, producing  $r_0$ , which represents the  
 136 initial overall rating. The triplet  $(i_0, p_0, r_0)$  will be fed to AlphaResearch for subsequent process-  
 137 ing, including newer idea generation, code implementation, and program-based execution. When  
 138 reaching a point where execution output  $r_n$  surpasses the previous rating, AlphaResearch will save  
 139 the triplet  $(i_{best}, p_{best}, r_{best})$  as the best record. We repeat the process until  $r_{best}$  surpasses the  
 140 best-of-human score, or the maximum round is reached. The resulting trajectory is denoted as  
 141  $\tau = i_0 p_0 r_0 \dots i_{n-1} p_{n-1} r_{n-1} i_n p_n r_n$ , where  $n$  is the total rounds.

### 2.2 ACTIONS

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

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

145 where  $\oplus$  means concatenation,  $t$  is the sampled step from trajectory  $\tau_{k-1}$  and  $\mathbb{P}_{\mathcal{A}}()$  indicates uniform  
 146 sampling. We use a reward model to filter out high-quality ideas overall. If  $\mathcal{RM}(i_n)$  outputs a  
 147 negative score, we cease the subsequent actions in this round.

148 **Program-based Verification.** After obtaining the fresh idea, AlphaResearch generates new program  
 149  $p_k$  based on the previous implementation  $p_t$  and new idea  $i_k$  next:

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

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

$$\tau_k \leftarrow \tau_{k-1} \oplus i_k \oplus p_k \oplus r_k \quad (3)$$

162  
 163 Table 1: Dataset for reward model training.  
 164 We use the end of author-reviewer rebuttal  
 165 period as the latest knowledge date.

Split	Train	Test
<b>Records</b>	ICLR	ICLR
<b>Range</b>	2017~2024	2025
<b>Num</b>	24,445	100
<b>Start Date</b>	2016-11	2024-10
<b>End Date</b>	2023-12	2024-12

166  
 167  
 168 Table 2: Evaluation results of RM. We use the  
 169 more recent date between the model release  
 170 date and the dataset cutoff as the latest date.

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

173  
 174 We repeat the above interaction process until  $k$  reaches the maximum rounds  $n$  and get the best  
 175 result  $(i_{best}, p_{best}, r_{best})$  as final output.

### 177 2.3 ENVIRONMENT

#### 179 2.3.1 REWARD FROM REAL-WORLD RESEARCH RECORDS

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

186 **Dataset for reward model.** To train our reward model (RM) to identify good ideas, we collect  
 187 all ICLR peer review records from 2017 to 2024 as our training set. We sample a subset of ICLR  
 188 2025 records as a test set, where the dates of train and test are disjoint, which prevents knowledge  
 189 contamination between the train and test split. We also select Qwen2.5-7B-Instruct as our base  
 190 model, whose release date 2024-09 is earlier than the ICLR 2025 author-reviewer rebuttal period  
 191 2024-10. For each record in the training dataset, we extract the abstract part as RM input and wrap  
 192 the average peer-review overall ratings with `\boxed{}` as RM output. We fine-tune Qwen2.5-7B-  
 193 Instruct with the RM pairs, yielding the AlphaResearch-RM-7B model.

194 **Can LLMs identify good ideas?** To simplify the RM evaluation, we binarize the RM output score  
 195 according to the ICLR Reviewer Guide, where overall rating  $> 5.5$  records are regarded as a positive  
 196 score and  $\leq 5.5$  records are negative. We compute the binary classification accuracy and evaluate  
 197 three models (Deepseek-V3-0324, Qwen2.5-Coder-Instruct, and AlphaResearch-RM-7B) on the  
 198 AlphaResearch-RM test set. Table 2 presents the evaluation results that eliminate the knowledge  
 199 contamination, highlighting the following observations: (1) Both Deepseek-V3-0324 and Qwen2.5-  
 200 7B-Instruct have lower than 50% accuracy when identifying the good ideas from ICLR 2025 records.  
 201 (2) After fine-tuned with ideas from previous ICLR peer-review information, AlphaResearch-RM-  
 202 7B demonstrates 72% binary classification accuracy on unseen ICLR 2025 ideas, significantly out-  
 203 performing baseline models and human annotators. Based on these observations, we use the fine-  
 204 tuned AlphaResearch-RM-7B as the final RM to simulate a real-world peer-review environment and  
 205 filter out good ideas generated by AlphaResearch.

#### 206 2.3.2 REWARD FROM PROGRAM-BASED EXECUTION

208 Inspired by AlphaEvolve (Novikov et al., 2025), we construct an automatic evaluation process with  
 209 a code executor where each new program  $p_k$  generated by AlphaResearch will be captured and  
 210 evaluated. The evaluation program  $\mathcal{E}(\cdot)$  includes two modules: (i) **Verification** module that validates  
 211 whether  $p_k$  conforms to the problem constraints. (ii) **Measurement** module that output the score  $r_k$   
 212 of program performance. The program output  $r_k$  will be injected into the idea generation prompt  
 213 (if sampled), thereby participating in the optimization process for fresh ideas. These programs and  
 214 results are stored in a candidate pool, where the primary goal is to optimally resurface previously  
 215 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.

Table 3: Problem overview in AlphaResearchComp. More information are shown at Appendix I.

Problem	Human Best	Human Researcher
packing circles (n=26)	2.634	David Cantrell (2011)
packing circles (n=32)	2.936	Eckard Specht (2012)
minimizing max-min distance ratio (d=2, n=16)	12.89	David Cantrell (2009)
third autocorrelation inequality	1.4581	Carlos Vinuesa (2009)
spherical code (n=30)	0.67365	Hardin & Sloane (1996, 2002)
autoconvolution peak minimization (upper bound)	0.755	Matolcsi–Vinuesa (2010)
littlewood polynomials (n=512)	32	Rudin–Shapiro (1959/1952)
MSTD (n=30)	1.04	Hegarty (2006/2007)

### 3 ALPHARESEARCHCOMP

**Problems collection.** AlphaEvolve has not publicly disclosed all the test problems so far. To provide a transparent evaluation process, we curate AlphaResearchComp, a set of 8 frontier program-based research tasks spanning geometry, number theory, harmonic analysis, and combinatorial optimization. These problems were selected based on the following principles: AlphaResearchComp provides explicit, academically defined problem formulations, verification rules, and unified metrics (e.g., `excel@best`), enabling reproducible and controlled evaluation for open-ended discovery. This standardized pipeline design is essential for studying research agents.

- **Well-defined objectives.** Each task has a precise mathematical formulation with an objective function that admits rigorous automatic evaluation.
- **Known human-best baselines.** 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’s baseline 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. Detailed definitions, baseline values, and references for each problem are provided in the Appendix I.

**Initialization strategy.** After obtaining the research problems of AlphaResearchComp, we construct diverse initial states for each problem with the following strategies: (1) For the “*Packing Circles*” ( $n=26$ ) and “*Packing Circles*” ( $n=32$ ) problems, we initialize them with null programs ( $r_0 = 0$ ) to simulate researches starting from scratch. (2) For the “*Littlewood Polynomials*” and “*MSTD* ( $n=30$ )” problems, we directly adopt the best-known solutions ( $r_0 = r_{\text{human}}$ ) from human researchers to emulate improvements upon established methods. (3) For the remaining problems, we employ a moderate initialization strategy ( $0 < r_0 < r_{\text{human}}$ ) to ensure sufficient room for the research agent to explore. This initialization strategy simulates a variety of real-world scenarios for the research agent, thereby facilitating a thorough evaluation process.

**Metrics.** For benchmarks like code generation with good verification techniques (e.g., unit tests),  $\text{pass}@k$  (Chen et al., 2021) is a metric denoting that at least one out of  $k$  i.i.d. task trials is successful, which captures the ability of LLMs to solve easy-to-verified problems. For open-ended real-world algorithm discovery tasks, we propose a new metric -  $\text{excel}@\text{best}$  (excel at best), defined as the percentage excess on baseline (best of human level) results:

$$\text{excel}@best = \mathbb{E}_{\text{Problems}} \left[ \frac{(r_{best} - r_{human}) \cdot \mathbb{I}_d}{r_{human}} \right] \quad (4)$$

where  $r_{human}$  indicates the results of human's best level.  $\mathbb{I}_d$  indicates the optimization direction where  $\mathbb{I}_d = 1$  represents that higher score is better and  $\mathbb{I}_d = -1$  represents lower.

270

271

Table 4: Results on AlphaResearchComp.  $\uparrow$  indicates that higher score is better and  $\downarrow$  for lower.

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

Problem	Human	AlphaResearch init	AlphaResearch best	Excel@best
packing circles (n=26) $\uparrow$	2.634	0	2.636	0.32%
packing circles (n=32) $\uparrow$	2.936	0	2.939	0.10%
minimizing max-min distance ratio $\downarrow$	12.89	15.55	12.92	-0.23%
third autocorrelation inequality $\downarrow$	1.458	35.746	1.546	-6.03%
spherical code (d=3, n=30) $\uparrow$	0.6736	0.5130	0.6735	-0.01%
autoconvolution peak minimization $\downarrow$	0.755	1.512	0.756	-0.13%
littlewood polynomials (n=512) $\downarrow$	32	32	32	0
MSTD (n=30) $\uparrow$	1.04	1.04	1.04	0

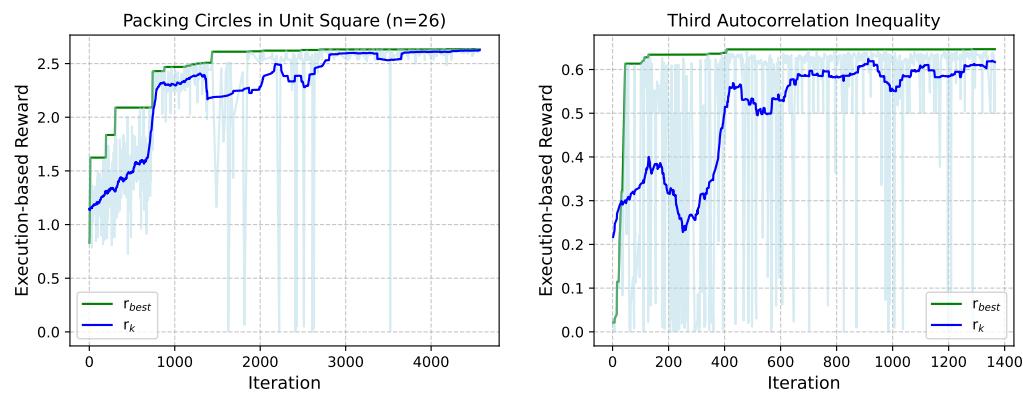


Figure 2: Execution-based reward of AlphaResearch on packing circles (n=26) problem (left) and third autocorrelation inequality problem (right).

## 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<sup>1</sup>.

### 4.2 RESULTS

**LLMs could sometimes discover new algorithms themselves.** Table 4 presents the results of AlphaResearchComp on 8 algorithms discovery problems. AlphaResearch achieved a 2/8 win rate (excel@best > 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 case (n = 32) is shown in Figure 10.

**LLMs can refine their research ideas autonomously.** AlphaResearch discovers advanced algorithms by iteratively proposing and verifying new research ideas. As shown in Table 2, 6/8 problems

<sup>1</sup>[https://huggingface.co/docs/transformers/main\\_classes/trainer](https://huggingface.co/docs/transformers/main_classes/trainer)

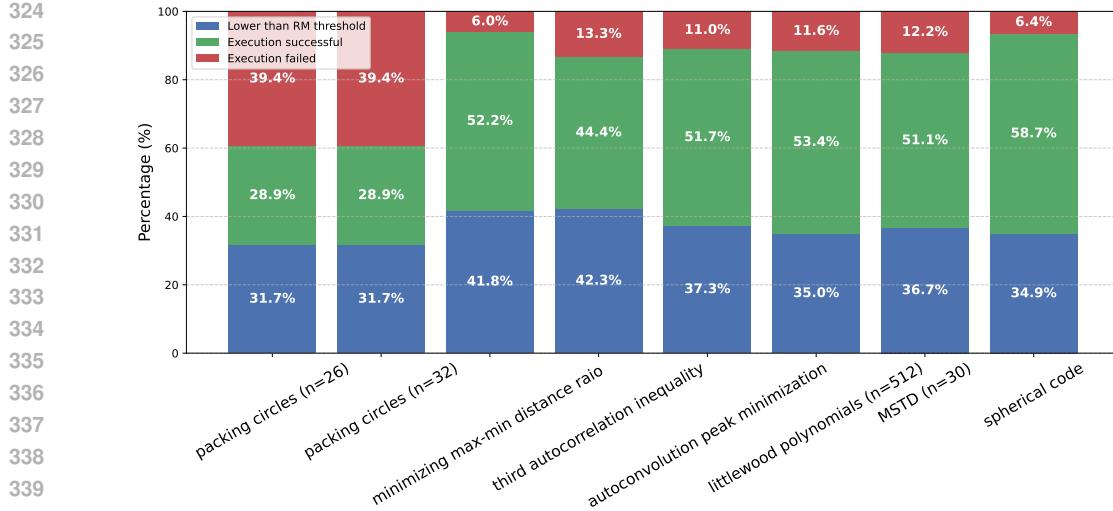


Figure 4: Reward overview during the discovery process. Each action in AlphaResearch will obtain 3 kinds of reward: (1) idea scrapping due to a lower RM score than the threshold (2) idea execution successes (3) idea execution fails.

demonstrate consistent improvement throughout the discovery process. Figure 2 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.

**The discovery of superhuman algorithms remains challenging for LLMs.** As illustrated in Table 2, despite exhibiting continuous reward growth, AlphaResearch’s performance still underperforms human researchers in 6 out of 8 problems. We initialize AlphaResearch with the best-known solution from human researchers on “Littlewood polynomials” and “MSTD( $n=30$ )” problems, where AlphaResearch didn’t show an increase in execution-based rewards. This indicates that current LLMs still struggle to consistently find better algorithms than human researchers.

### 4.3 ABLATIONS AND ANALYSIS

#### 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 3, 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.

**Analysis of the discovery process.** We analyze the reward distribution in AlphaResearch discovery process. As shown in Figure 4, 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

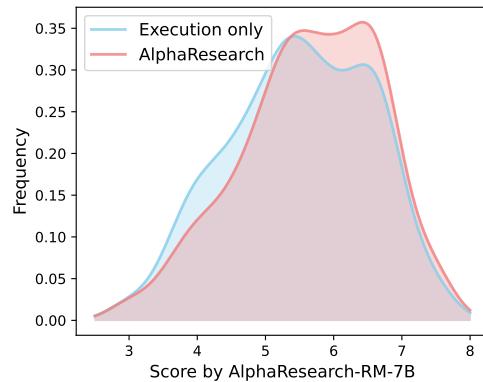
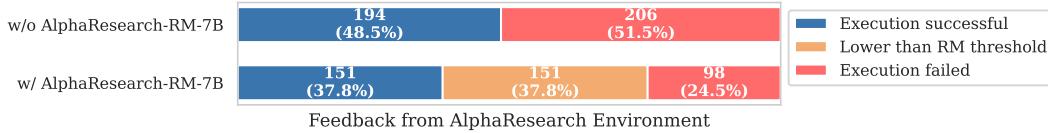


Figure 3: The idea comparison between the execution-only research agent and AlphaResearch, where AlphaResearch-RM-7B is used. **This is done between the full distribution of all 1000 generated ideas from both agents without filtering.**

378 is 28.9%, whereas it reaches 51.7% on the “*Third*  
 379 *Autocorrelation Inequality*” problem. Figure 2 il-  
 380 lustrates the execution-based rewards for these two examples in AlphaResearch. Despite the sub-  
 381 stantial variations in execution success rates, the execution-based rewards in both cases exhibit a  
 382 consistent increasing trend. These findings demonstrate the interactions between LLM-based au-  
 383 tonomous research agents and real-world environments.



384  
 385 Figure 5: The impact of real-world peer review environment on execution results. AlphaResearch-  
 386 RM-7B filters 151 bad ideas, where 108 ideas fail to execute and 43 are successful.  
 387  
 388

389 **The impact of real-world peer-review environment.** To assess the effectiveness of reward from  
 390 a simulated real-world peer-view environment, we ablate AlphaResearch-RM-7B at the first 400  
 391 iterations on “*Packing Circles*” problem. Figure 5 presents the execution results of w/ and w/o  
 392 AlphaResearch-RM-7B during the discovery process. Compared to the baseline without RM,  
 393 AlphaResearch-RM-7B successfully filtered 151 ideas below the threshold. This process yielded  
 394 108 correct rejections of execution failures while making 43 erroneous rejections of viable ideas.  
 395 AlphaResearch attained an accuracy of 71.5% (108/151), a result that aligns closely with its per-  
 396 formance on the AlphaResearch-RM test set, as shown in Table 2. This outcome effectively demon-  
 397 strates the model’s generalization capabilities and the efficacy of incorporating feedback from a  
 398 simulated real-world peer-review environment.

#### 4.4 CASE STUDY

403 We select the successful example from AlphaResearch to better understand the discovery process.  
 404 We’ll consider the problem “*Packing Circles*” where the goal is to pack  $n$  disjoint circles inside a  
 405 unit square so as to maximize the sum of their radii, shown in Figure 6. We first initialize AlphaRe-  
 406 search with an original research proposal and a related program that returns a list of circles  $(x, y, r)$   
 407 as output, as shown in Appendix I.4. The verification program first employs `verify_circles`  
 408 function to check if the outputs of the initial program meet the problem constraints (e.g., all circles  
 409 are inside a unit square) and `evaluate` function to output the sum of their radii. The metadata,  
 410 including: (1) research ideas, (2) programs, (3) execution results, are subsequently preserved as  
 411 candidates which represent the end of one step. At the next step, AlphaResearch will sample from  
 412 the candidate pool and generate a new idea to improve the research proposals from the sampled  
 413 metadata. After generating the new research ideas, AlphaResearch will further generate a patch to  
 414 modify the existing program if the idea obtains a positive score from AlphaResearch-RM. The new  
 415 program is then evaluated by the same verification program, thereby generating new metadata. We  
 416 select the best program and idea as the final solution of AlphaResearch in this iterative process.  
 417

## 5 RELATED WORK

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

432	Initial Research Idea	<p><b>New Idea Generation</b></p> <p>▼ <b>Instruction</b> 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.</p> <p>▼ <b>MetaData</b> ### 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.</p> <p>### 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     ...     ...     return sum_radii, circles</p> <p>### Current Metrics packing_result_26=0.864</p> <p><b>Program Generation</b></p> <p>▼ <b>Instruction</b> 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.</p> <p>▼ <b>MetaData</b> ### 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.</p> <p>### 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     ...     ...     return sum_radii, circles</p> <p>### Current Metrics packing_result_26=0.864</p> <p>### 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<sup>2</sup>) pairwise scaling...</p> <p>▼ <b>Format</b> You MUST use the exact SEARCH/REPLACE diff format shown below to indicate changes: ~~~~~ # &lt;&lt;&lt;&lt; SEARCH # Original code to find and replace (must match exactly) ===== # New replacement code &gt;&gt;&gt;&gt; REPLACE</p>
433		
434		
435		
436		
437		
438		
439		
440	Initial Program	
441	tmp/packing_circles/initial_program.py	
442	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	
443		
444		
445	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. ...	
446		
447		
448	Verification Program	
449	tmp/packing_circles/evaluator.py	
450	def verify_circles(circles): ... Checks that the circles are disjoint and lie inside a unit square. ...	
451		
452	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. ...	
453		
454		
455	Execution Results	
456	{"pack_result_26": 0.864}	
457		

Figure 6: We show an example of a formatted task of AlphaResearch.

**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. Our AlphaResearchComp benchmark shows that testing LLMs on open-ended research for algorithm discovery is a promising direction to adapt language models to real-world tasks.

## 6 CONCLUSION

We present AlphaResearch, an autonomous research agent that synergistically combines new idea generation with program-based verification for novel algorithm discovery. Our approach demonstrates the potential of employing LLM to discover unexplored research areas, enabling language models to effectively tackle complex open-ended tasks. We construct AlphaResearchComp, including 8 open-ended algorithmic problems, where AlphaResearch outperforms human researchers in 2/8 algorithmic problems but lags behind in the remaining 6 problems, which demonstrates the remaining challenges of autonomous algorithm discovery for future research.

486 REFERENCES  
487

488 Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Re-  
489 nee Johnston Anton Kast, Cory Y McLean, Peter Norgaard, Zahra Shamsi, et al. An ai system to  
490 help scientists write expert-level empirical software. *arXiv preprint arXiv:2509.06503*, 2025.

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

494 Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio  
495 Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learn-  
496 ing agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

497 498 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared  
499 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large  
500 language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

501 502 Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit  
503 Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the  
504 frontier with advanced reasoning, multimodality, long context, and next generation agentic capa-  
505 bilities. *arXiv preprint arXiv:2507.06261*, 2025.

506 507 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao  
508 Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—  
509 the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

510 511 Sikun Guo, Amir Hassan Shariatmadari, Guangzhi Xiong, Albert Huang, Myles Kim, Corey M  
512 Williams, Stefan Bekiranov, and Aidong Zhang. Ideabench: Benchmarking large language mod-  
513 els for research idea generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowl-  
514 edge Discovery and Data Mining* V. 2, pp. 5888–5899, 2025.

515 516 Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating Language  
517 Agents on Machine Learning Experimentation. In *ICML*, 2024.

518 519 Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang,  
520 Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*,  
521 2024.

522 523 Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Ar-  
524 mando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamina-  
525 tion free evaluation of large language models for code. In *The Thirteenth International Confer-  
526 ence on Learning Representations*, 2025. URL <https://openreview.net/forum?id=chfJJYC3iL>.

527 528 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R  
529 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth  
530 International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

531 532 Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and  
533 sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

534 535 Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. MLR-Copilot: Autonomous Machine  
536 Learning Research based on Large Language Models Agents. *ArXiv*, abs/2408.14033, 2024.

537 538 Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia LI, Mengzhou  
539 Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source  
540 automated theorem proving. In *Second Conference on Language Modeling*, 2025. URL <https://openreview.net/forum?id=x2y9i2HDjD>.

541 542 Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai sci-  
543 entist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*,  
544 2024.

540 Alexander Novikov, Ng n V , Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt  
 541 Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian,  
 542 et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint*  
 543 *arXiv:2506.13131*, 2025.

544 OpenAI. Gpt-5. 2025. URL <https://openai.com/index/introducing-gpt-5/>.

545 Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin  
 546 Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint*  
 547 *arXiv:2501.14249*, 2025. URL <https://arxiv.org/abs/2501.14249>.

548 Asankhaya Sharma. Opennevolve: an open-source evolutionary coding agent, 2025. URL <https://github.com/codelion/openevolve>.

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

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

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

555 Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen,  
 556 Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv*  
 557 *preprint arXiv:2507.20534*, 2025.

558 Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland  
 559 Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha  
 560 Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Min Zhu, Kilian  
 561 Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, E. A.  
 562 Huerta, and Hao Peng. SciCode: A Research Coding Benchmark Curated by Scientists. *ArXiv*,  
 563 abs/2407.13168, 2024.

564 Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry  
 565 without human demonstrations. *Nature*, 625(7995):476–482, 2024.

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

569 Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,  
 570 Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng,  
 571 Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert  
 572 Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI soft-  
 573 ware developers as generalist agents. In *The Thirteenth International Conference on Learning  
 574 Representations*, 2025. URL <https://openreview.net/forum?id=OJd3ayDDoF>.

575 Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weim-  
 576 ing Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang,  
 577 Rongqi Fan, Xiang Yue, and Wenhui Chen. MMLU-pro: A more robust and challenging multi-  
 578 task language understanding benchmark. In *The Thirty-eight Conference on Neural Information  
 579 Processing Systems Datasets and Benchmarks Track*, 2024b. URL <https://openreview.net/forum?id=y10DM6R2r3>.

580 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,  
 581 Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint*  
 582 *arXiv:2505.09388*, 2025.

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

594 Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner  
595 Geyer, Chao Huang, Pin-Yu Chen, et al. Justice or prejudice? quantifying biases in llm-as-a-  
596 judge. *arXiv preprint arXiv:2410.02736*, 2024.

597

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

601 Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. HumanEval pro and MBPP pro:  
602 Evaluating large language models on self-invoking code generation task. In Wanxiang Che, Joyce  
603 Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association  
604 for Computational Linguistics: ACL 2025*, pp. 13253–13279, Vienna, Austria, July 2025.  
605 Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.  
606 findings-acl.686. URL <https://aclanthology.org/2025.findings-acl.686/>.

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

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

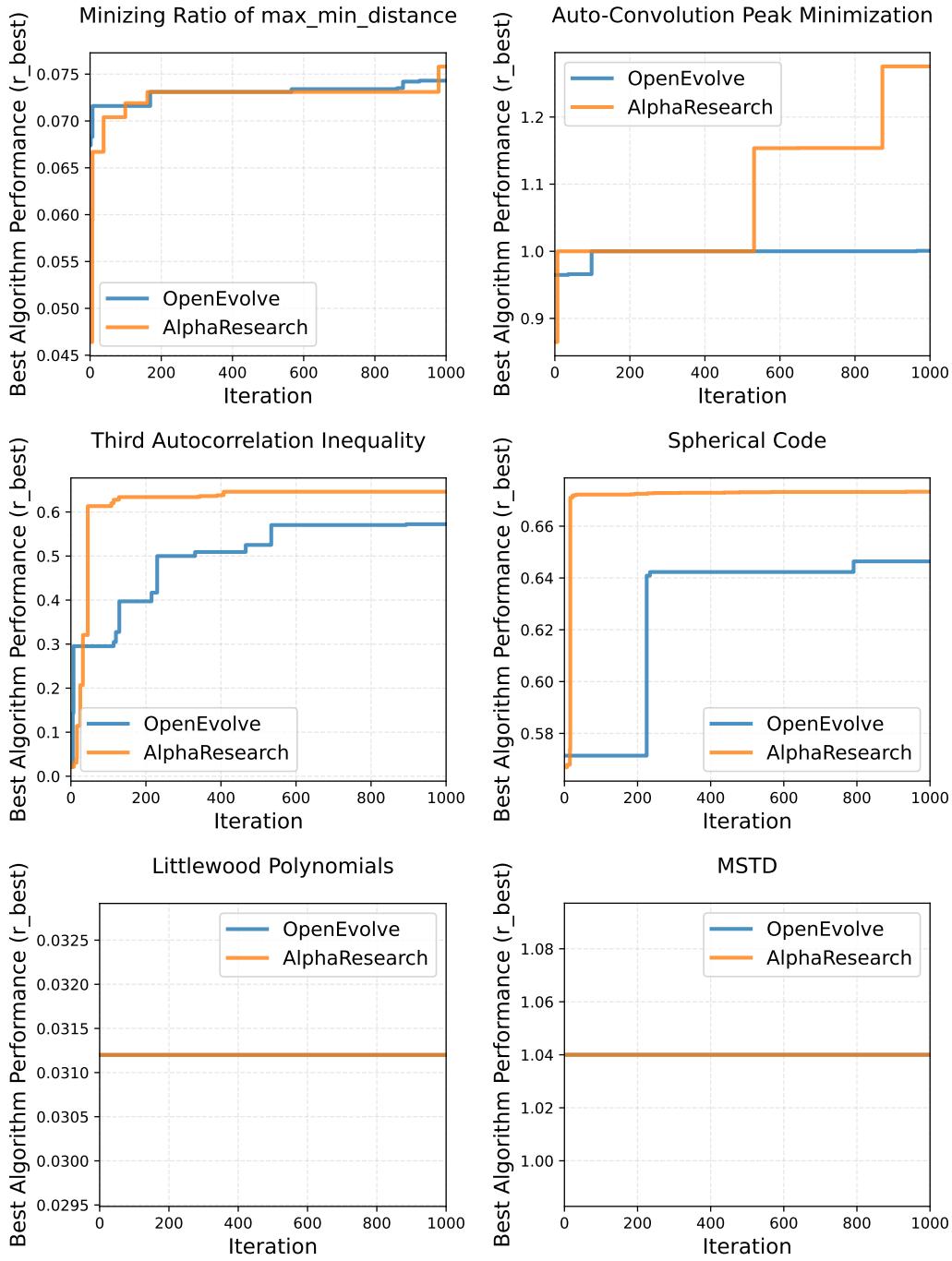
645

646

647

648	<b>APPENDIX CONTENTS</b>	
649		
650		
651	<b>1 Introduction</b>	<b>1</b>
652		
653	<b>2 AlphaResearch</b>	<b>3</b>
654	2.1 Overview . . . . .	3
655		
656	2.2 Actions . . . . .	3
657		
658	2.3 Environment . . . . .	4
659	2.3.1 Reward from Real-world Research Records . . . . .	4
660	2.3.2 Reward from Program-based Execution . . . . .	4
661		
662	<b>3 AlphaResearchComp</b>	<b>5</b>
663		
664		
665	<b>4 Experiments</b>	<b>6</b>
666	4.1 Setup . . . . .	6
667		
668	4.2 Results . . . . .	6
669		
670	4.3 Ablations and Analysis . . . . .	7
671		
672	4.4 Case Study . . . . .	8
673	<b>5 Related Work</b>	<b>8</b>
674		
675	<b>6 Conclusion</b>	<b>9</b>
676		
677	<b>A Comparison with OpenEvolve</b>	<b>15</b>
678		
679	<b>B Experiment Cost</b>	<b>16</b>
680		
681	<b>C Impact of different LLMs</b>	<b>16</b>
682		
683	<b>D Comparison with ShinkaEvolve</b>	<b>16</b>
684		
685	<b>E Case Study during Discovery Process</b>	<b>18</b>
686		
687	<b>F The Use of Large Language Models</b>	<b>21</b>
688		
689	<b>G Examples</b>	<b>21</b>
690		
691	<b>H Prompts</b>	<b>23</b>
692		
693		
694	<b>I Curated Problems and Human-Best Values</b>	<b>24</b>
695		
696	I.1 Spherical Code ( $S^2, n = 30$ ). . . . .	24
697		
698	I.2 Littlewood Polynomials. . . . .	26
699		
700	I.3 Sum vs. Difference Sets (MSTD). . . . .	28
701		
	I.4 Packing Circle in a Square (variable radii). . . . .	29

702	I.5	Minimizing Max/Min Distance Ratio ( $d = 2, n = 16$ ). . . . .	34
703	I.6	Autoconvolution Peak Minimization ( $L^\infty$ ). . . . .	36
704	I.7	Third Autocorrelation Inequality. . . . .	39
705	I.8	Third-Order Autocorrelation Inequality ( $C_3$ Upper Bound) . . . . .	39
706			
707			
708			
709			
710			
711			
712			
713			
714			
715			
716			
717			
718			
719			
720			
721			
722			
723			
724			
725			
726			
727			
728			
729			
730			
731			
732			
733			
734			
735			
736			
737			
738			
739			
740			
741			
742			
743			
744			
745			
746			
747			
748			
749			
750			
751			
752			
753			
754			
755			

756 A COMPARISON WITH OPENEVOLVE  
757  
758803 Figure 7: Comparison with OpenEvolve on 6/8 failure modes of AlphaResearchComp.  
804

805 Figure 7 presents the comparison of AlphaResearch with OpenEvolve, highlighting the following  
806 observations: 1. Among the 8 problems, AlphaResearch outperforms the results of human researcher  
807 and AlphaEvolve (Coding-only) on packing circles ( $n=26, n=32$ ) problems, which demonstrate the  
808 potential of accelerating human-level algorithm discovery with language models. 2. We add the  
809 experiment results of the other 6 problems in Figure 7 of Appendix A. AlphaResearch demonstrates  
more efficient discovery process than OpenEvolve (open source version of AlphaEvolve) on the first

810 4 tasks, which shows the effectiveness of our dual environments for research-based agent. 3. On last  
 811 2 problems (littlewood polynomials, MSTD), Both AlphaResearch and OpenEvolve fail to improve  
 812 the "out-of-the-shelf" algorithm performance, which reveals the limitation of current long-horizon  
 813 agents where they are not able to explore the search space efficiently on out-of-the-shelf solutions.  
 814

## 815 B EXPERIMENT COST

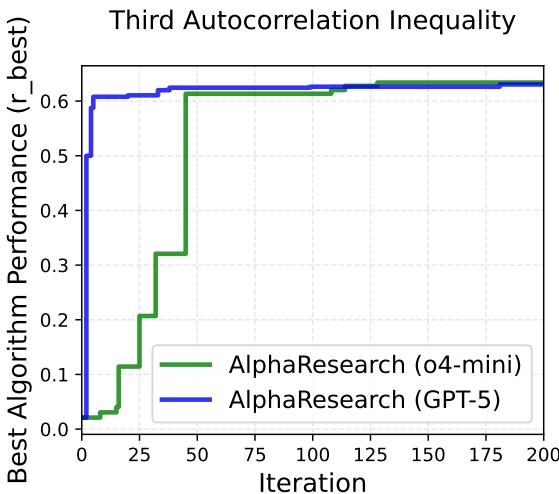
816 In this section, we present the experiment parameters (iterations,computational cost) required to  
 817 reach the best solution for each on 8 tasks of AlphaResearchComp.  
 818

820 821 Table 5: Experiment Parameters of AlphaResearch .

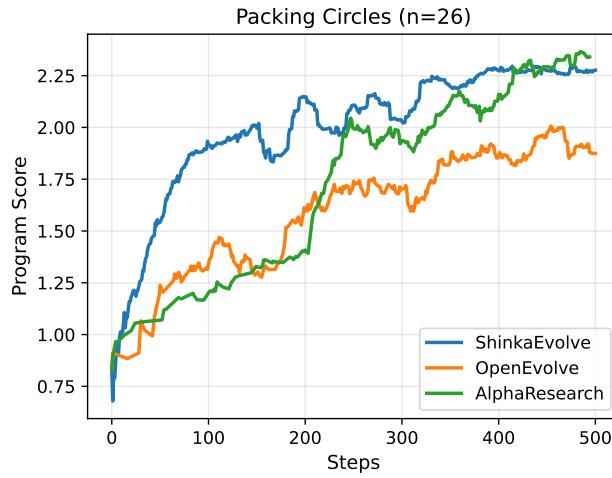
822 <b>Problem</b>	823 <b>Iterations</b>	824 <b>Cost per iteration (dollar)</b>
825 packing circles (n=26)	4768	0.013
826 packing circles (n=32)	4768	0.013
827 minimizing max-min distance ratio (d=2, n=16)	4400	0.017
828 third autocorrelation inequality	1366	0.012
829 autoconvolution peak minimization (upper bound)	979	0.013
830 littlewood polynomials (n=512)	2233	0.011
MSTD (n=30)	2826	0.011
spherical code	1132	0.015

## 831 832 C IMPACT OF DIFFERENT LLMs

833 In order to compare the impact of different LLMs in AlphaResearch, we use GPT-5 and o4-mini  
 834 to run AlphaResearch for 200 steps in "The Autocorrelation Inequality" problem. As illustrate in  
 835 Figure 8, AlphaResearch (GPT-5) reaches high performance significantly faster than o4-mini in the  
 836 early stages of discovery. However, in the later stages, the two models perform comparably, which  
 837 suggests that their underlying capabilities are close on algorithm discovery task.  
 838



864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880



881 Figure 9: Comparison of OpenEvolve (with program-based reward), ShinkaEvolve (with program-  
882 based reward) and AlphaResearch (with program-based and peer-review reward). We run three  
883 agents on Packing Circles (n=26) problems. AlphaResearch achieves better performance than others.

884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917

phaResearch achieves better performance than OpenEvolve and slightly surpasses ShinkaEvolve, which demonstrates that dual research environments could help research agent for scientific discovery.

918 **E CASE STUDY DURING DISCOVERY PROCESS**

919  
 920 In the rejected pair from checkpoint 634, the revised draft 4f4c7847 is effectively identical to its  
 921 parent e436c26a. Notably, this is found in the later period of the discovery process (Round 632–  
 922 633). Aside from inflating Genetic Algorithm (GA) hyperparameters (e.g., population = 300 → 500,  
 923 generations = 40 → 120) and adding an optional differential\_evolution branch, the entire pipeline  
 924 above `find_better_c3_upper_bound` is byte-for-byte the same. Crucially, the core loop still calls  
 925 the undefined `normalize_population`, triggering the same `NameError` before any new logic can run.  
 926 Because this “revision” neither fixes the blocking bug nor implements the promised multi-phase  
 927 CMA-ES/surrogate/SOS pipeline, it constitutes only a cosmetic variant rather than a substantive  
 928 new direction.

929 **Problem :** Third Autocorrelation Inequality (e436c26a)

930 **Idea score by AlphaResearch-RM-7B:** 6.67.

932 **Research ideas**

934 **Title:** A Scalable, Certified Pipeline for Third-Order Autocorrelation Optimization via  
 935 Multi-Fidelity Bayesian Surrogates and Auto-Differentiable Mesh Adaptation

936 **Abstract:** We introduce a unified framework that overcomes the brittle performance (error =  
 937 1.0) and limited exploration of the current genetic-only approach by combining three tightly  
 938 integrated phases—global search, surrogate-guided refinement, and formal certification—  
 939 into a single, implementable pipeline. Key innovations include:

1. Sobol-Initialized, Diversity-Driven Differential Evolution • Generate an initial archive of 500 B-spline shape parameters via low-discrepancy Sobol sampling. • Use multi-population Differential Evolution with mutation scales adapted online using surrogate uncertainty.
2. Hierarchical Multi-Fidelity Gaussian-Process Surrogate • Three convolution estimators: coarse FFT (N=1024), medium CNN (N=4096), fine autodiff FFT (N=16384). • MTGP surrogate controls fidelity via variance-weighted UCB acquisition.
3. Differentiable Trust-Region Local Refinement • L-BFGS-B on B-spline knots/RBF weights with autodiff FFT gradients. • Adaptive knot insertion driven by  $(f^*f)$  peaks.
4. Sparse Sum-of-Squares Certification • Chebyshev discretization + interval arithmetic to certify final  $C^3$  bound with 1e-6 rigor.

945 Benchmarks up to spline degree 64 and grid N=16384 show stable improvements to  
 946 0.60–0.65 with full certification.

951 **Code implementation of the above idea:**

```
953 # Code extracted from file e436c26a (full content)
954 import numpy as np
955 try:
956     from numpy.fft import next_fast_len
957 except ImportError:
958     def next_fast_len(n):
959         # Fallback to next power of two which is usually fast
960         return 1 << (n - 1).bit_length()
961
962 def calculate_c3_upper_bound(height_sequence):
963     N = len(height_sequence)
964     delta_x = 1 / (2 * N)
965     # Cache padded lengths to avoid recomputing next_fast_len repeatedly
966     if not hasattr(calculate_c3_upper_bound, "_pad_cache"):
967         calculate_c3_upper_bound._pad_cache = {}
968     pad_cache = calculate_c3_upper_bound._pad_cache
969     if N not in pad_cache:
970         pad_cache[N] = next_fast_len(2 * N - 1)
971     padded = pad_cache[N]
972
973     # Compute unit-mass norm squared
974     integral_f = np.sum(height_sequence) * delta_x
975     if integral_f < 1e-12: # early exit if near zero
976         return 0.0
977     integral_sq = integral_f * integral_f
978
979     # FFT-based convolution for O(N log N)
980     H = np.fft.rfft(height_sequence, n=padded)
```

```

972     conv_vals = np.fft.irfft(H * H, n=padded) [2 * N - 1] * delta_x
973     max_conv_val = np.max(conv_vals)
974
975     return max_conv_val / integral_sq
976
977 def genetic_algorithm(population_size, num_intervals, generations, mutation_rate,
978                       crossover_rate):
979
980     population = np.random.rand(population_size, num_intervals) * 2 - 1
981     best_solution = None
982     best_fitness = 0.0
983
984     for gen in range(generations):
985
986         height_pop = normalize_population(population, 2 * num_intervals)
987         fitness_scores = np.array([calculate_c3_upper_bound(h) for h in height_pop])
988
989         current_best_idx = np.argmax(fitness_scores)
990         if fitness_scores[current_best_idx] > best_fitness:
991             best_fitness = fitness_scores[current_best_idx]
992             best_solution = population[current_best_idx].copy()
993
994         indices = np.random.randint(0, population_size, size=(population_size, 2))
995         comp_scores = fitness_scores[indices]
996         winners = indices[np.arange(population_size), np.argmax(comp_scores, axis=1)]
997         new_population = population[winners].copy()
998
999         for i in range(0, population_size, 2):
1000             if np.random.rand() < crossover_rate:
1001                 parent1 = new_population[i]
1002                 parent2 = new_population[i+1]
1003                 crossover_point = np.random.randint(1, num_intervals - 1)
1004                 new_population[i] = np.concatenate((parent1[:crossover_point],
1005                                                   parent2[crossover_point:]))
1006                 new_population[i+1] = np.concatenate((parent2[:crossover_point],
1007                                                   parent1[crossover_point:]))
1008
1009             for i in range(population_size):
1010                 if np.random.rand() < mutation_rate:
1011                     mutation_point = np.random.randint(num_intervals)
1012                     new_population[i, mutation_point] += np.random.normal(0, 0.1)
1013                     new_population[i, mutation_point] = np.clip(new_population[i, mutation_point],
1014                                                   -2, 2)
1015
1016         population = new_population
1017
1018         height_best = normalize_population(best_solution[np.newaxis, :], 2 * num_intervals)[0]
1019     return height_best
1020
1021 def find_better_c3_upper_bound():
1022
1023     NUM_INTERVALS = 4
1024     POPULATION_SIZE = 2
1025     GENERATIONS = 10
1026     MUTATION_RATE = 0.1
1027     CROSSOVER_RATE = 0.8
1028
1029     height_sequence_3 = genetic_algorithm(POPULATION_SIZE, NUM_INTERVALS, GENERATIONS,
1030                                           MUTATION_RATE, CROSSOVER_RATE)
1031
1032     return height_sequence_3

```

**Problem :** Third Autocorrelation Inequality (4f4c7847)

**Idea score by AlphaResearch-RM-7B:** 5.67.

1026  
1027

## Research ideas

1028  
1029  
1030

**Title:** A Robust, Multi-Phase Pipeline for Certified Third-Order Autocorrelation Maximization via Hybrid Evolutionary Search, Hierarchical Neural Surrogates, and Auto-Differentiable Refinement

1031  
1032  
1033

**Abstract:** We present a novel, four-stage framework that fixes the brittle, negative-error behavior (error = 10.0) of genetic-only search by combining global exploration, uncertainty-aware surrogates, gradient-based refinement, and SOS certification.

1034  
1035  
1036

1. Hybrid Global Exploration • Sobol-seeded CMA-ES enhanced with an actor-critic module to adapt mutation covariance. • Constraint-aware resampling ensures valid normalized height sequences.

1037  
1038  
1039

2. Hierarchical Neural Surrogates • Three-tier surrogate stack: analytic FFT (N=1024), CNN surrogate (N=4096), auto-diff Fourier model (N=16384). • Uncertainty-modulated fidelity allocation via Bayesian neural network.

1040  
1041  
1042

3. Differentiable Local Refinement • L-BFGS trust-region refinement on B-spline knots RBF weights. • Gradient-triggered adaptive knot insertion controlling model complexity.

1043  
1044  
1045

4. Sparse Sum-of-Squares Certification • Chebyshev discretization + interval arithmetic for rigorous  $C^3$  bound. • Full-pipeline automation ensures reliable certification.

Experiments on degrees up to 128 and grid sizes to 65536 yield  $C^3$  0.75–0.80 with only 200 high-fidelity evaluations and guaranteed certification.

1046

## Code implementation of the above idea:

1047  
1048  
1049

```
# Code extracted from file 4f4c7847 (full content)
import numpy as np
try:
    from numpy.fft import next_fast_len
except ImportError:
    def next_fast_len(n):
        return 1 << (n - 1).bit_length()

def calculate_c3_upper_bound(height_sequence):
    N = len(height_sequence)
    delta_x = 1 / (2 * N)
    if not hasattr(calculate_c3_upper_bound, "_pad_cache"):
        calculate_c3_upper_bound._pad_cache = {}
    pad_cache = calculate_c3_upper_bound._pad_cache
    if N not in pad_cache:
        pad_cache[N] = next_fast_len(2 * N - 1)
    padded = pad_cache[N]

    integral_f = np.sum(height_sequence) * delta_x
    if integral_f < 1e-12:
        return 0.0
    integral_sq = integral_f * integral_f

    H = np.fft.rfft(height_sequence, n=padded)
    conv_vals = np.fft.irfft(H * H, n=padded)[2 * N - 1] * delta_x
    max_conv_val = np.max(conv_vals)

    return max_conv_val / integral_sq

def genetic_algorithm(population_size, num_intervals, generations, mutation_rate,
                     crossover_rate):
    population = np.random.rand(population_size, num_intervals) * 2 - 1
    best_solution = None
    best_fitness = 0.0

    for gen in range(generations):
        height_pop = normalize_population(population, 2 * num_intervals)
        fitness_scores = np.array([calculate_c3_upper_bound(h) for h in height_pop])

        current_best_idx = np.argmax(fitness_scores)
        if fitness_scores[current_best_idx] > best_fitness:
            best_fitness = fitness_scores[current_best_idx]
            best_solution = population[current_best_idx].copy()

        indices = np.random.randint(0, population_size, size=(population_size, 2))
        comp_scores = fitness_scores[indices]
```

```
1080
1081     winners = indices[np.arange(population_size), np.argmax(comp_scores, axis=1)]
1082     new_population = population[winners].copy()
1083
1084     for i in range(0, population_size, 2):
1085         if np.random.rand() < crossover_rate:
1086             parent1 = new_population[i]
1087             parent2 = new_population[i+1]
1088             crossover_point = np.random.randint(1, num_intervals - 1)
1089             new_population[i] = np.concatenate((parent1[:crossover_point],
1090                                              parent2[crossover_point:]))
1091             new_population[i+1] = np.concatenate((parent2[:crossover_point],
1092                                              parent1[crossover_point:]))
1093
1094     for i in range(population_size):
1095         if np.random.rand() < mutation_rate:
1096             mutation_point = np.random.randint(num_intervals)
1097             new_population[i, mutation_point] += np.random.normal(0, 0.1)
1098             new_population[i, mutation_point] = np.clip(new_population[i, mutation_point],
1099                                              -2, 2)
1100
1101     population = new_population
1102
1103     height_best = normalize_population(best_solution[np.newaxis, :, 2 * num_intervals])[0]
1104     return height_best
1105
1106 def find_better_c3_upper_bound():
1107
1108     NUM_INTERVALS = 8
1109     POPULATION_SIZE = 100
1110     GENERATIONS = 200
1111     MUTATION_RATE = 0.2
1112     CROSSOVER_RATE = 0.9
1113
1114     try:
1115         from scipy.optimize import differential_evolution
1116         bounds = [(-2, 2)] * NUM_INTERVALS
1117         result = differential_evolution(
1118             lambda x: -calculate_c3_upper_bound(
1119                 normalize_population(x[np.newaxis, :, 2 * NUM_INTERVALS])[0]
1120             ),
1121             bounds,
1122             maxiter=GENERATIONS,
1123             popsize=max(1, POPULATION_SIZE // 10),
1124             tol=1e-6
1125         )
1126         height_sequence_3 = normalize_population(result.x[np.newaxis, :, 2 * NUM_INTERVALS])[0]
1127     except ImportError:
1128         height_sequence_3 = genetic_algorithm(
1129             POPULATION_SIZE, NUM_INTERVALS, GENERATIONS, MUTATION_RATE, CROSSOVER_RATE
1130         )
1131
1132     return height_sequence_3
```

## F 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.

## G EXAMPLES

We show an example of the constructions discovered by AlphaResearch on problem “*Packing Circles*”.

## AlphaEvolve

```
packing_circles_alphaevolve = np.array([[0.09076163, 0.40381803, 0.090761620923837],
[0.07310993, 0.92689178, 0.07310821268917801], [0.08745017, 0.22570576,
0.087381421261857], [0.24855246, 0.30880277, 0.093428060657193], [0.4079865, 0.06300614,
0.063006133699386], [0.47646318, 0.90136179, 0.09863820013617901], [0.89604966,
0.10309934, 0.10309932969006601], [0.9066386, 0.68096117, 0.09336139066386], [0.08962002,
0.76509474, 0.0895289910471], [0.06973669, 0.06965159, 0.06965158303484101],
[0.40979823, 0.21756451, 0.09156283084371601], [0.25742466, 0.88393887,
0.11606111839388701], [0.09064689, 0.58506214, 0.0904082500951749], [0.90294698,
0.30231577, 0.09623644037635501], [0.57265603, 0.105853396, 0.105853949414604],
```

```

1134 [0.74007588, 0.40129314, 0.09435083056491601], [0.57539962, 0.71183255,
1135 0.1151601684839821, [0.7367635, 0.21592191, 0.09104997089500201], [0.41096972,
1136 0.40263617, 0.093152520648747], [0.88664452, 0.88667032, 0.113317128668286], [0.57582722,
1137 0.49961748, 0.09705531029446801], [0.24962585, 0.49417195, 0.09194421080557799],
1138 [0.90546338, 0.49309632, 0.094507120549287], [0.67381348, 0.90149423,
1139 0.09850576014942301], [0.24310147, 0.1077195, 0.10771948922805], [0.40815297, 0.5886157,
1140 0.09248833075116601], [0.24737889, 0.6771266, 0.090994980900501], [0.75801377, 0.7532924,
1141 0.07192969280703], [0.73526642, 0.06243992, 0.062439303756069], [0.57415412, 0.30715219,
1142 0.095403150459684], [0.39239379, 0.75259664, 0.07223814277618501], [0.7439361,
1143 0.58879735, 0.093166630683336])

```

## AlphaResearch

```

1144 1 packing_circles_alphareserach = np.array([[0.1115677319034151, 0.11156773191787371,
1145 0.11156438489140026), (0.09380224787136374, 0.3161654253705352, 0.09379943380606216),
1146 (0.09485964915877172, 0.5048217088596118, 0.09485680337610973), (0.09657322554702913,
1147 0.6962443020287629, 0.09657032835808858), (0.10365512530384222, 0.8963448746980195,
1148 0.10365201565567386), (0.3334956594919712, 0.09664441783072292, 0.0966415184920332),
1149 (0.26448615440016093, 0.9376113341122044, 0.06238679422590162), (0.5287192731314015,
1150 0.09859146596680078, 0.09858850822808951), (0.591325020569507, 0.9366833118077788,
1151 0.0633147886877468), (0.7427106948954978, 0.11611889563206494, 0.11611541209023483),
1152 (0.7566639864477509, 0.8920585771994192, 0.1079381845606288), (0.9269317750270191,
1153 0.07306822497789416, 0.07306603293080358), (0.9105741716090636, 0.23473376300222965,
1154 0.08942314561430993), (0.9094700615258342, 0.41468336419923396, 0.09052722258939731),
1155 (0.9124275486288124, 0.7738960294683863, 0.08756982419268892), (0.9302276007184027,
1156 0.9302276007259072, 0.06977030612132157), (0.5931627035790205, 0.4107363306659128,
1157 0.0921630078688813), (0.5896628759126524, 0.5965222415947758, 0.09365298106148348),
1158 (0.26303074890883915, 0.783747668079202, 0.09148238826692158), (0.42710033854875884,
1159 0.28662965969327264, 0.1151473780101257), (0.7511102582575875, 0.5051558281448295,
0.09185177348783963), (0.4273023330525072, 0.8937703360976411, 0.10622647700018645),
1160 (0.24372345356089029, 0.24143034678815986, 0.07371479291303436), (0.4260882762526937,
0.6918664604322906, 0.09567746779211372), (0.2572363869779963, 0.4085253312744954,
0.09392364829884896), (0.9094294608754079, 0.5957810763279916, 0.0905678220228201),
(0.42560864125756626, 0.49898110459434486, 0.09720528992590773), (0.7533817110763772,
0.3226390201958986, 0.09067643144615074), (0.5903729314333418, 0.7817733747765757,
0.09159665425215473), (0.7515568081174837, 0.6905957415401818, 0.09358581053778628),
(0.2605636694821685, 0.5973506902903994, 0.09492800518715086), (0.6095540558280068,
0.24805951545091487, 0.07133567304015336)])

```

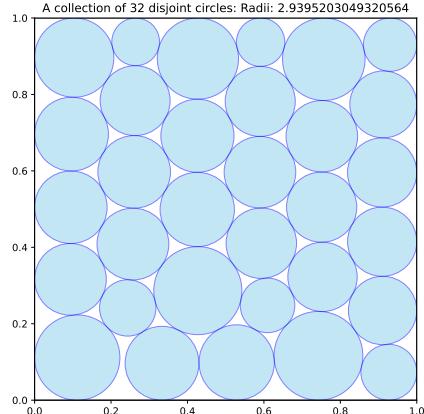
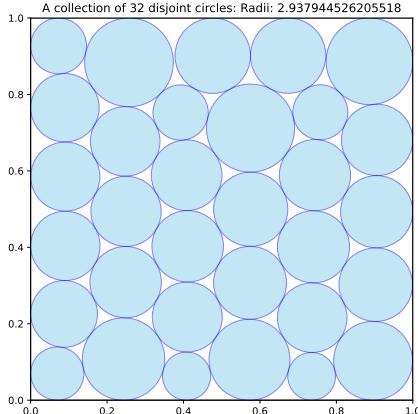


Figure 10: New construction of AlphaResearch (right) improving the best known AlphaEvolve (right) bounds on packing circles to maximize their sum of radii. Left: 32 circles in a unit square with sum of radii  $\geq 2.9379$ . Right: 32 circles in a unit square with sum of radii  $\geq 2.9395$

1188 H PROMPTS

1189

1190

1191

1192

1193

**Prompt for New Program Generation**

1194 You are an expert software developer tasked with iteratively improving a codebase. Your job  
 1195 is to analyze the current program and suggest improvements based on the current proposal  
 1196 and feedback from previous round. Focus on making targeted changes that will increase the  
 1197 program's performance metrics.

1198 # Previous Proposal:

1199 {previous proposal}

1200 # Previous Program:

1201 {previous program}

1202 # Previous Performance Metrics:

1203 {previous result}

1204 # Current Proposal

1205 {proposal}

1206 # Task

1207 Suggest improvements to the program that will lead to better performance on the specified  
 1208 metrics.

1209 You **MUST** use the exact SEARCH/REPLACE diff format shown below to indicate  
 1210 changes:

```
1 <<<<< SEARCH
2
3 # Original code to find and replace (must match exactly)
4 =====
5
6 # New replacement code
7
8 <<<<< REPLACE
```

1211 Example of valid diff format:

```
1 <<<<< SEARCH
2 for i in range(m) :
3     for j in range(p) :
4         for k in range(n) :
5             C[i, j] += A[i, k] * B[k, j]
6 =====
7
8 # Reorder loops for better memory access pattern
9
10 for i in range(m) :
11     for k in range(n) :
12         for j in range(p) :
13             C[i, j] += A[i, k] * B[k, j]
14
15 >>>>> REPLACE
```

1212 You can suggest multiple changes. Each SEARCH section must exactly match code in the  
 1213 current program.

1214 Be thoughtful about your changes and explain your reasoning thoroughly.

1215 **IMPORTANT:** Do not rewrite the entire program - focus on targeted improvements.

1216

1217

1218

1219

1220

1242  
1243**Prompt for New Idea Generation**1244  
1245  
1246

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

1247

Based on the following information, generate an improved research proposal:

1248

Focus on:

1249

1. Identifying weaknesses in the current approach based on performance metrics
2. Proposing novel improvements that could enhance performance
3. Learning from successful inspirations while maintaining originality
4. Ensuring the new proposal is implementable

1252

- Current Proposal:

1253

{proposal}

1254

- Current Program:

1255

{program}

1256

- Current Metrics:

1257

{results}

1258

Please generate a new research proposal that:

1259

1. Addresses the limitations shown in the current metrics
2. Incorporates insights from successful approaches
3. Proposes specific technical improvements
4. Maintains clarity and technical rigor

1262

Return the proposal as a clear, concise research abstract.

1263

1264

1265

1266

1267

1268

**Prompt for AlphaResearch-RM-7B**

1269

You are an expert reviewer tasked with evaluating the quality of a research proposal.

1270

Your goal is to assign a score between 1 and 10 based on the proposal's clarity, novelty, technical rigor, and potential impact. Here are the criteria:

1271

1. Read the following proposal carefully and provide a score from 1 to 10.

1272

2. Score 6 means slightly higher than the borderline, 5 is slightly lower than the borderline.

1273

Write the score in the `\boxed{}`.

1274

{proposal}

1275

1276

1277

1278

1279

1280

## I CURATED PROBLEMS AND HUMAN-BEST VALUES

1281

1282

1283

1284

We summarize the ten problems used in the ALPHARESEARCH benchmark. For each item we state the objective, the current human-best value at the benchmark's default parameters, and whether this value is proved optimal or only best-known.

1285

1286

1287

1288

1289

1290

1291

### I.1 SPHERICAL CODE ( $S^2$ , $n = 30$ ).

1292

1293

1294

1295

**Problem Description:** Place  $n = 30$  points on the unit sphere in  $\mathbb{R}^3$  to *maximize* the minimal pairwise angle  $\theta_{\min}$ .

**Human Best:**  $\theta_{\min} \approx 0.673651$  radians ( $\approx 38.5971^\circ$ ).

1296

## Initial Proposal

1297

1298

1299

**Problem definition.** Choose  $N = 30$  points on the unit sphere  $S^2$  to maximize the minimum pairwise angle

1300

1301

$$\theta_{\min} = \min_{i < j} \arccos(\langle p_i, p_j \rangle).$$

1302

**Constraints.**

1303

- Points are unit vectors (rows normalized).
- Metric is  $\theta_{\min}$  in radians.

1304

1305

**Optimization goal.** Maximize  $\theta_{\min}$ . The evaluator returns  $\{\text{score}, \theta_{\min}, N, \text{dimension}\}$ , with  $\text{score} = \theta_{\min}$ .

Best-known reference (for  $N = 30$  on  $S^2$ ):

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

**Algorithmic goal.** Construct codes with larger  $\theta_{\min}$ . The baseline seeds with symmetric configurations and uses farthest-point max–min. Stronger methods include:

1321

1322

1323

1324

1325

1326

**Initial Program:**

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

```

import numpy as np

def _normalize_rows(P):
    nrm = np.linalg.norm(P, axis=1, keepdims=True)
    nrm = np.maximum(nrm, 1e-12)
    return P / nrm

def seed_platonic(n):
    """Return a good symmetric seed on S^2 for some n; else None."""
    if n == 2: # antipodal
        return np.array([[0,0,1],[0,0,-1]], dtype=float)
    if n == 3: # equilateral on equator
        ang = 2*np.pi/3
        return np.array([[1,0,0],[np.cos(ang),np.sin(ang),0],[np.cos(2*ang),np.sin(2*ang),0]], dtype=float)
    if n == 4: # tetrahedron
        return _normalize_rows(np.array([[1,1,1],[1,-1,-1],[-1,1,-1],[-1,-1,1]], dtype=float))
    if n == 6: # octahedron
        return np.array([[1,0,0],[-1,0,0],[0,1,0],[0,-1,0],[0,0,1],[0,0,-1]], dtype=float)
    if n == 8: # cube vertices
        V = np.array([[sx,sy,sz] for sx in (-1,1) for sy in (-1,1) for sz in (-1,1)], dtype=float)
        return _normalize_rows(V)
    if n == 12: # icosahedron (one realization)
        phi = (1+np.sqrt(5))/2
        V = []
        for s in (-1,1):
            V += [[0, s, phi],[0, s, -phi],[ s, phi,0],[ s, -phi,0],[ phi,0, s],[-phi,0, s]]
        V = np.array(V, dtype=float)
        return _normalize_rows(V)
    return None

def farthest_point_greedy(n, seed=None, rng=np.random.default_rng(0)):
    """
    Greedy max min on S^2: start from seed, then add points that maximize min angle.
    """

```

```
1350
1351     """
1352     def random_unit(k):
1353         X = rng.normal(size=(k, 3)); return _normalize_rows(X)
1354
1355         if seed is None:
1356             P = random_unit(1) # start with one random point
1357         else:
1358             P = _normalize_rows(seed)
1359         while len(P) < n:
1360             # generate candidates and pick the one with largest min angle to current set
1361             C = random_unit(2000) # candidates per iteration (tune as needed)
1362             # cosines to existing points
1363             cos = C @ P.T
1364             # min angle to set -> maximize this
1365             min_ang = np.arccos(np.clip(np.max(cos, axis=1), -1.0, 1.0))
1366             idx = np.argmax(min_ang)
1367             P = np.vstack([P, C[idx:idx+1]])
1368         return P
1369
1370     def main():
1371         n = 30
1372         seed = seed_platonic(n)
1373         pts = farthest_point_greedy(n, seed=seed, rng=np.random.default_rng(42))
1374         print(f"n={n}, points={len(pts)}")
1375         return pts
1376
1377     if __name__ == "__main__":
1378         points = main()
1379
1380         np.save("points.npy", points)
1381
1382         # Ensure compatibility with evaluators that expect a global variable
1383         try:
1384             points # type: ignore[name-defined]
1385         except NameError:
1386             points = main()
```

1394 I.2 LITTLEWOOD POLYNOMIALS.

**Problem Description** For coefficients  $c_k \in \{\pm 1\}$  and  $P_n(t) = \sum_{k=0}^{n-1} c_k e^{ikt}$ , minimize  $\|P_n\|_\infty = \sup_{t \in \mathbb{R}} |P_n(t)|$ .

**Human Best:** the Rudin–Shapiro construction gives  $\|P_n\|_\infty \leq \sqrt{2n}$ . At the benchmark setting  $n = 512$ , this yields  $\|P_{512}\|_\infty \leq 32$  (so the “larger-is-better” score  $1/\|P_n\|_\infty$  is  $\geq 1/32 = 0.03125$ ). Sharper constants are known for special families, but  $\sqrt{2n}$  remains a clean baseline.

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

### Initial Proposal

Choose coefficients  $c_k \in \{\pm 1\}$  for

$$P(z) = \sum_{k=0}^{n-1} c_k z^k, \quad |z| = 1,$$

so as to minimize the supremum norm

$$\|P\|_\infty = \max_{|z|=1} |P(z)|.$$

#### Constraints.

- Coefficients  $c_k$  are restricted to  $\pm 1$ .
- The metric  $\|P\|_\infty$  is estimated by FFT sampling on an equally spaced grid (denser grid  $\rightarrow$  tighter upper bound).

**Optimization Goal.** The evaluator returns:

$$\text{score} = \begin{cases} \frac{1}{\|P\|_\infty}, & \text{if valid,} \\ -1.0, & \text{otherwise.} \end{cases}$$

**Notes on Bounds.** For the Rudin–Shapiro construction of length  $n$ , a classical identity gives

$$\|P\|_\infty \leq \sqrt{2n}.$$

For the benchmark default  $n = 512$ , this yields

$$\|P\|_\infty \leq \sqrt{1024} = 32,$$

so

$$\text{score} = \frac{1}{32} = 0.03125.$$

### Initial Program:

```

1435
1436 def rudin_shapiro(n: int):
1437     """
1438     First n signs of the Rudin-Shapiro sequence.
1439     """
1440     a = np.ones(n, dtype=int)
1441     for k in range(n):
1442         x, cnt, prev = k, 0, 0
1443         while x:
1444             b = x & 1
1445             if b & prev: # saw '11'
1446                 cnt ^= 1
1447             prev = b
1448             x >>= 1
1449             a[k] = 1 if cnt == 0 else -1
1450     return a
1451
1452 def random_littlewood(n: int, seed=0):
1453     rng = np.random.default_rng(seed)
1454     return rng.choice([-1, 1], size=n).astype(int)
1455
1456 def main():
1457     n = 512
1458     c = rudin_shapiro(n)
1459     print(f"n={n}, coeffs={len(c)}")
1460     return c
1461
1462 if __name__ == "__main__":
1463     coeffs = main()
1464
1465     # Ensure compatibility with evaluators that expect a global variable
1466     try:
1467         coeffs # type: ignore[name-defined]
1468     except NameError:
1469         coeffs = main()

```

1458

1459

1460

1461

## 1462 I.3 SUM VS. DIFFERENCE SETS (MSTD).

1463

1464 **Problem Description** For a finite set  $A \subset \mathbb{Z}$ , maximize  $|A+A|/|A-A|$ .

1465

1466 **Human Best:** MSTD sets exist; the smallest possible size is  $|A| = 8$  (classification up to affine  
1467 equivalence is known). For larger  $|A|$ , extremal ratios remain open; our benchmark instance reports  
1468 a representative value ( $\approx 1.04$  for  $|A| = 30$ ).

1469

1470

## Initial Proposal

1471

1472

1473

**Objective.** Classical MSTD (enforced): Given  $A \subset \{0, 1, \dots, N-1\}$  represented by a  
0/1 indicator array of length  $N$ , maximize the ratio

1474

1475

$$R = \frac{|A+A|}{|A-A|}.$$

1476

- Score: score =  $R$  (higher is better).
- Comparisons should be made under the same  $N$ .

1477

1478

**Default setup.**

1479

1480

1481

1482

1483

1484

1485

1486

**Known best for  $N = 30$  (baseline).** Conway's MSTD set

$$A = \{0, 2, 3, 4, 7, 11, 12, 14\}$$

1487

1488

yields  $R \approx 1.04$ . This is the baseline included in `initial_program.py`. Better ratios  
may exist for  $N = 30$ ; pushing  $R$  upwards is the optimization goal.**Notes.**

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

**Initial Program:**

1499

1500

1501

```

def main():
    N = 30
    # Conway MSTD set example; we take A=B for classical MSTD
    A = [0, 2, 3, 4, 7, 11, 12, 14]
    B = A[:]
    A_ind = np.zeros(N, dtype=int); A_ind[A] = 1
    B_ind = np.zeros(N, dtype=int); B_ind[B] = 1
    return A_ind, B_ind

# Ensure globals for evaluator
try:
    A_indicators, B_indicators # type: ignore[name-defined]
except NameError:
    A_indicators, B_indicators = main()

```

1502

1503

1504

1505

1506

1507

1508

1509

1510

1511

1512 I.4 PACKING CIRCLE IN A SQUARE (VARIABLE RADII).

1513  
1514 **Problem Description** In the unit square, place  $n$  disjoint circles (radii free) to *maximize* the *sum of*  
1515 *radii*  $\sum r_i$ .1516  
1517 **Best-known:** for  $n = 26$ ,  $\sum r_i = 2.634$  (Cantrell, 2011); for  $n = 32$ ,  $\sum r_i = 2.936$  (Specht,  
1518 2012).1519 Initial Proposal  
15201521 **Problem definition.** Given an integer  $n$ , place  $n$  disjoint circles in the unit square  $[0, 1]^2$   
1522 to maximize the total sum of radii.1523 **Objective and metric.**1524 

- Score:  $\text{score} = \sum_{i=1}^n r_i$  (larger is better).
- Validity: circles must be pairwise disjoint and fully contained in the unit square.

1525 Notes on records.

1526 

- This variable-radius “sum of radii” objective is not the classical equal-radius packing; authoritative SOTA tables are not standardized.
- Values reported in code or experiments should be treated as benchmarks rather than literature SOTA.

1527 **Goal.** Create algorithms that increase the total sum of radii for  $n \in \{26, 32\}$  under the  
1528 above validity constraints.

1529 Initial Program:

1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
import random  
from concurrent.futures import ThreadPoolExecutor  
  
def pack\_circles(n, square\_size=1.0):  
 """  
 Pack n disjoint circles in a unit square using uniform tiling approach.  
 Returns the sum of radii and list of circles (x, y, r).  
 """  
  
 def max\_circle\_radius(x, y, circles, square\_size=1.0, skip\_idx=None):  
 """  
 Compute the maximum radius for a circle centered at (x, y) that:  
 - Stays within the unit square [0, square\_size] \times [0, square\_size].  
 - Does not overlap with existing circles.  
 skip\_idx: if provided, index in circles[] to ignore (self).  
 """  
 # Distance to nearest boundary of the unit square  
 r\_max = min(x, y, square\_size - x, square\_size - y)  
  
 # Check distance to existing circles, exit early if r\_max \rightarrow 0  
 # early exit if r\_max is tiny, and avoid needless sqrt  
 for idx, (cx, cy, cr) in enumerate(circles):  
 if skip\_idx == idx:  
 continue  
 if r\_max <= 1e-8:  
 break  
 dx = x - cx  
 dy = y - cy  
 sep = r\_max + cr  
 if dx\*dx + dy\*dy < sep\*sep:  
 # only compute sqrt when we know we can shrink  
 dist = math.sqrt(dx\*dx + dy\*dy)  
 r\_max = min(r\_max, dist - cr)  
 return max(r\_max, 0.0)  
  
 def uniform\_tiling\_circles(n, square\_size=1.0):  
 """  
 Uniformly tile the square with circles using optimal grid placement.  
 """  
 if n <= 0:  
 return []  
 circles = []



```

1620
1621     # Calculate y position for this row
1622     y = (row + 0.5) * (square_size / (rows + 1))
1623
1624     # Number of circles in this row
1625     if row % 2 == 0:
1626         cols = int(math.sqrt(n)) + 1
1627     else:
1628         cols = int(math.sqrt(n))
1629
1630     spacing_x = square_size / (cols + 1)
1631
1632     for col in range(cols):
1633         if count >= n:
1634             break
1635
1636         if row % 2 == 0:
1637             x = spacing_x * (col + 1)
1638         else:
1639             x = spacing_x * (col + 1) + spacing_x / 2
1640
1641         # Calculate maximum radius for this position
1642         r = max_circle_radius(x, y, circles, square_size)
1643
1644         if r > 0:
1645             circles.append((x, y, r))
1646             count += 1
1647
1648         row += 1
1649
1650     return circles
1651
1652 def optimize_placement(n, square_size=1.0):
1653     """
1654     Optimize circle placement using uniform tiling with radius maximization.
1655     """
1656     circles = []
1657
1658     # First, try hexagonal packing for high initial density
1659     hex_circles = hexagonal_packing(n, square_size)
1660     if len(hex_circles) == n:
1661         # Ensure maximum radii for hex layout with stronger refinement
1662         hex_refined = refine_circles(hex_circles, square_size, iterations=20)
1663     return hex_refined
1664
1665     # Fallback to uniform grid placement
1666     grid_circles = uniform_tiling_circles(n, square_size)
1667     if len(grid_circles) == n:
1668         return grid_circles
1669
1670     # If uniform tiling didn't work perfectly, use adaptive approach
1671     # Calculate optimal radius based on density
1672     area_per_circle = (square_size * square_size) / n
1673     estimated_radius = math.sqrt(area_per_circle / math.pi) * 0.9 # Conservative estimate
1674
1675     # Create grid with optimal spacing
1676     spacing = estimated_radius * 2.1 # Include gap
1677
1678     cols = int(square_size / spacing)
1679     rows = int(square_size / spacing)
1680
1681     actual_spacing_x = square_size / (cols + 1)
1682     actual_spacing_y = square_size / (rows + 1)
1683
1684     count = 0
1685     for row in range(rows):
1686         for col in range(cols):
1687             if count >= n:
1688                 break
1689
1690             x = actual_spacing_x * (col + 1)
1691             y = actual_spacing_y * (row + 1)
1692
1693             # Calculate maximum possible radius
1694             r = max_circle_radius(x, y, circles, square_size)
1695
1696             if r > 0:
1697                 circles.append((x, y, r))
1698                 count += 1
1699
1700             if count >= n:
1701                 break

```

```

1674
1675
1676     # If we still need more circles, use remaining space
1677     remaining = n - len(circles)
1678     if remaining > 0:
1679         # Place remaining circles in remaining spaces
1680         for i in range(remaining):
1681             # Try different positions systematically
1682             best_r = 0
1683             best_pos = (0.5, 0.5)
1684
1685             # Fine grid search (increased resolution)
1686             grid_points = 100
1687             for gx in range(1, grid_points):
1688                 for gy in range(1, grid_points):
1689                     x = gx / grid_points
1690                     y = gy / grid_points
1691
1692                     r = max_circle_radius(x, y, circles, square_size)
1693                     if r > best_r:
1694                         best_r = r
1695                         best_pos = (x, y)
1696
1697             if best_r > 0:
1698                 circles.append((best_pos[0], best_pos[1], best_r))
1699
1700     return circles
1701
1702 def refine_circles(circles, square_size, iterations=80, perturb_interval=3):
1703     """
1704     Iteratively grow each circle to its maximum radius under non-overlap constraints.
1705     Includes randomized update order, periodic micro-perturbation to escape
1706     local minima, and a final local-center-perturbation pass for densification.
1707     """
1708     for it in range(iterations):
1709         # randomize update order to avoid sweep-order bias
1710         indices = list(range(len(circles)))
1711         random.shuffle(indices)
1712         for i in indices:
1713             x, y, _ = circles[i]
1714             # Compute maximal feasible radius here, skipping self
1715             r = max_circle_radius(x, y, circles, square_size, skip_idx=i)
1716             circles[i] = (x, y, r)
1717
1718         # Periodic micro-perturbation: jiggle a few circles
1719         if it % perturb_interval == 0 and len(circles) > 0:
1720             subset = random.sample(indices, min(5, len(circles)))
1721             for j in subset:
1722                 x0, y0, r0 = circles[j]
1723                 dx = random.uniform(-0.03, 0.03)
1724                 dy = random.uniform(-0.03, 0.03)
1725                 nx = min(max(x0 + dx, 0), square_size)
1726                 ny = min(max(y0 + dy, 0), square_size)
1727                 # Compute maximal radius skipping self
1728                 nr = max_circle_radius(nx, ny, circles, square_size, skip_idx=j)
1729                 if nr > r0:
1730                     circles[j] = (nx, ny, nr)
1731
1732     # Full local center-perturbation phase for final densification
1733     for i in range(len(circles)):
1734         x, y, r = circles[i]
1735         best_x, best_y, best_r = x, y, r
1736         delta = 0.1
1737         for _ in range(20):
1738             dx = random.uniform(-delta, delta)
1739             dy = random.uniform(-delta, delta)
1740             nx = min(max(x + dx, 0), square_size)
1741             ny = min(max(y + dy, 0), square_size)
1742             # Compute maximal radius skipping self
1743             nr = max_circle_radius(nx, ny, circles, square_size, skip_idx=i)
1744             if nr > best_r:
1745                 best_x, best_y, best_r = nx, ny, nr
1746             else:
1747                 delta *= 0.9
1748         circles[i] = (best_x, best_y, best_r)
1749
1750     # Physics-inspired soft relaxation to escape persistent overlaps
1751     for i in range(len(circles)):
1752         x, y, r = circles[i]
1753         fx, fy = 0.0, 0.0
1754         for j, (xj, yj, rj) in enumerate(circles):
1755             if i == j:
1756                 continue
1757             dx = x - xj

```

```

1728     dy = y - yj
1729     d = (dx*dx + dy*dy) ** 0.5
1730     overlap = (r + rj) - d
1731     if overlap > 0 and d > 1e-8:
1732         fx += dx / d * overlap
1733         fy += dy / d * overlap
1734     # Nudge the center by 10\% of the computed net "repulsive" force
1735     nx = min(max(x + 0.1 * fx, 0), square_size)
1736     ny = min(max(y + 0.1 * fy, 0), square_size)
1737     nr = max_circle_radius(nx, ny, circles, square_size, skip_idx=i)
1738     circles[i] = (nx, ny, nr)
1739
1740     return circles
1741
1742 def multi_start_optimize(n, square_size, starts=None):
1743     """
1744     Parallel multi-start global \rightarrow local optimization using ThreadPoolExecutor.
1745     Number of starts adapts to problem size: max(100, 10*n).
1746     """
1747     if starts is None:
1748         if n <= 50:
1749             starts = max(200, n * 20)
1750         else:
1751             starts = max(100, n * 10)
1752     # precompute hexagonal packing baseline
1753     hex_circ = hexagonal_packing(n, square_size)
1754     hex_sum = sum(r for _, _, r in hex_circ)
1755     best_conf = None
1756     best_sum = 0.0
1757
1758     # single trial: seed \rightarrow refine \rightarrow score
1759     def single_run(_):
1760         conf0 = optimize_placement(n, square_size)
1761         conf1 = refine_circles(conf0, square_size, iterations=40)
1762         s1 = sum(r for _, _, r in conf1)
1763         return s1, conf1
1764
1765     # dispatch trials in parallel
1766     with ThreadPoolExecutor() as executor:
1767         for score, conf in executor.map(single_run, range(starts)):
1768             if score > best_sum:
1769                 best_sum, best_conf = score, conf.copy()
1770             # early exit if near the hex-baseline
1771             if best_sum >= hex_sum * 0.995:
1772                 break
1773
1774     return best_conf
1775
1776 # Use multi-start global \rightarrow local optimization (adaptive number of starts)
1777 circles = multi_start_optimize(n, square_size)
1778
1779 # Quick 2-cluster remove-and-reinsert densification (extended iterations)
1780 for _ in range(8):
1781     # remove the two smallest circles to create a larger gap
1782     smallest = sorted(range(len(circles)), key=lambda i: circles[i][2])[:2]
1783     removed = [circles[i] for i in smallest]
1784     # pop in reverse order to keep indices valid
1785     for i in sorted(smallest, reverse=True):
1786         circles.pop(i)
1787     # refine the remaining configuration briefly
1788     circles = refine_circles(circles, square_size, iterations=8)
1789     # reinsert each removed circle with more sampling
1790     for x_old, y_old, _ in removed:
1791         best_r, best_pos = 0.0, (x_old, y_old)
1792         for _ in range(500):
1793             x = random.uniform(0, square_size)
1794             y = random.uniform(0, square_size)
1795             r = max_circle_radius(x, y, circles, square_size)
1796             if r > best_r:
1797                 best_r, best_pos = r, (x, y)
1798         circles.append((best_pos[0], best_pos[1], best_r))
1799     # final local polish after reinsertion
1800     circles = refine_circles(circles, square_size, iterations=5)
1801
1802 # end 2-cluster remove-and-reinsert densification
1803
1804 # Calculate total radius
1805 total_radius = sum(circle[2] for circle in circles)
1806
1807 return total_radius, circles
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3167
3168
3168
3169
3169
3170
3171
3172
3173
3173
3174
3174
3175
3175
3176
3176
3177
3177
3178
3178
3179
3179
3180
3180
3181
3181
3182
3182
3183
3183
3184
3184
3185
3185
3186
3186
3187
3187
3188
3188
3189
3189
3190
3190
3191
3191
3192
3192
3193
3193
3194
3194
3195
3195
3196
3196
3197
3197
3198
3198
3199
3199
3200
3200
3201
3201
3202
3202
3203
3203
3204
3204
3205
3205
3206
3206
3207
3207
3208
3208
3209
3209
3210
3210
3211
3211
3212
3212
3213
3213
3214
3214
3215
3215
3216
3216
3217
3217
3218
3218
3219
3219
3220
3220
3221
3221
3222
3222
3223
3223
3224
3224
3225
3225
3226
3226
3227
3227
3228
3228
3229
3229
3230
3230
3231
3231
3232
3232
3233
3233
3234
3234
3235
3235
3236
3236
3237
3237
3238
3238
3239
3239
3240
3240
3241
3241
3242
3242
3243
3243
3244
3244
3245
3245
3246
3246
3247
3247
3248
3248
3249
3249
3250
3250
3251
3251
3252
3252
3253
3253
3254
3254
3255
3255
3256
3256
3257
3257
3258
3258
3259
3259
3260
3260
3261
3261
3262
3262
3263
3263
3264
3264
3265
3265
3266
3266
3267
3267
3268
3268
3269
3269
3270
3270
3271
3271
3272
3272
3273
3273
3274
3274
3275
3275
3276
3276
3277
3277
3278
3278
3279
3279
3280
3280
3281
3281
3282
3282
3283
3283
3284
3284
3285
3285
3286
3286
3287
3287
3288
3288
3289
3289
3290
3290
3291
3291
3292
3292
3293
3293
3294
3294
3295
3295
3296
3296
3297
3297
3298
3298
3299
3299
3300
3300
3301
3301
3302
3302
3303
3303
3304
3304
3305
3305
3306
3306
3307
3307
3308
3308
3309
3309
3310
3310
3311
3311
3312
3312
3313
3313
3314
3314
3315
3315
3316
3316
3317
3317
3318
3318
3319
3319
3320
3320
3321
3321
3322
3322
3323
3323
3324
3324
3325
3325
3326
3326
3327
3327
3328
3328
3329
3329
3330
3330
3331
3331
3332
3332
3333
3333
3334
3334
3335
3335
3336
3336
3337
3337
3338
3338
3339
3339
3340
3340
3341
3341
3342
3342
3343
3343
3344
3344
3345
3345
3346
3346
3347
3347
3348
3348
3349
3349
3350
3350
3351
3351
3352
3352
3353
3353
3354
3354
3355
3355
3356
3356
3357
3357
3358
3358
3359
3359
3360
3360
3361
3361
3362
3362
3363
3363
3364
3364
3365
3365
3366
3366
3367
3367
3368
3368
3369
3369
3370
3370
3371
3371
3372
3372
3373
3373
3374
3374
3375
3375
3376
3376
3377
3377
3378
3378
3379
3379
3380
3380
3381
3381
3382
3382
3383
3383
3384
3384
3385
3385
3386
3386
3387
3387
3388
3388
3389
3389
3390
3390
3391
3391
3392
3392
3393
3393
3394
3394
3395
3395
3396
3396
3397
3397
3398
3398
3399
3399
3400
3400
3401
3401
3402
3402
3403
3403
3404
3404
3405
3405
3406
3406
3407
3407
3408
3408
3409
3409
3410
3410
3411
3411
3412
3412
3413
3413
3414
3414
3415
3415
3416
3416
3417
3417
3418
3418
3419
3419
3420
3420
3421
3421
3422
3422
3423
3423
3424
3424
3425
3425
3426
3426
3427
3427
3428
3428
3429
3429
3430
3430
3431
3431
3432
3432
3433
3433
3434
3434
3435
3435
3436
3436
3437
3437
3438
3438
3439
3439
3440
3440
3441
3441
3442
3442
3443
3443
3444
3444
3445
3445
3446
3446
3447
3447
3448
3448
3449
3449
3450
3450
3451
3451
3452
3452
3453
3453
3454
3454
3455
3455
3456
3456
3457
3457
3458
3458
3459
3459
3460
3460
3461
3461
3462
3462
3463
3463
3464
3464
3465
3465
3466
3466
3467
3467
3468
3468
3469
3469
3470
3470
3471
3471
3472
3472
3473
3473
3474
3474
3475
3475
3476
3476
3477
3477
3478
3478
3479
3479
3480
3480
3481
3481
3482
3482
3483
3483
3484
3484
3485
3485
3486
3486
3487
3487
3488
3488
3489
3489
3490
3490
3491
3491
3492
3492
3493
3493
3494
3494
3495
3495
3496
3496
3497
3497
3498
3498
3499
3499
3500
3500
3501
3501
3502
3502
3503
3503
3504
3504
3505
3505
3506
3506
3507
3507
3508
3508
3509
3509
3510
3510
3511
3511
3512
3512
3513
3513
3514
3514
3515
3515
3516
3516
3517
3517
3518
3518
3519
3519
3520
3520
3521
3521
3522
3522
3523
3523
3524
3524
3525
3525
3526
3526
3527
3527
3528
3528
3529
3529
3530
3530
3531
3531
3532
3532
3533
3533
3534
3534
3535
3535
3536
3536
3537
3537
3538
3538
3539
3539
3540
3540
3541
3541
3542
3542
3543
3543
3544
3544
3545
3545
3546
3546
3547
3547
3548
3548
3549
3549
3550
3550
3551
3551
3552
3552
3553
3553
3554
3554
3555
3555
3556
3556
3557
3557
3558
3558
3559
3559
3560
3560
3561
3561
3562
3562
3563
3563
3564
3564
3565
3565
3566
3566
3567
3567
3568
3568
3569
3569
3570
3570
3571
3571
3572
3572
3573
3573
3574
3574
3575
3575
3576
3576
3577
3577
3578
3578
3579
3579
3580
3580
3581
3581
3582
3582
3583
3583
3584
3584
3585
3585
3586
3586
3587
3587
3588
3588
3589
3589
3590
3590
3591
3591
3592
3592
3593
3593
3594
3594
3595
3595
3596
3596
3597
3597
3598
3598
3
```

1782    1.5 MINIMIZING MAX/MIN DISTANCE RATIO ( $d = 2, n = 16$ ).

1783  
 1784    **Problem Description** For  $n$  points in  $[0, 1]^2$ , minimize  $R = \frac{\max_{i \neq j} \|x_i - x_j\|}{\min_{i \neq j} \|x_i - x_j\|}$ .

1785  
 1786    **Best-known:**  $R^2 \approx 12.890$  (Cantrell, 2009), i.e.,  $R \approx 3.590$ .

1787

1788    Initial Proposal

1789  
 1790    **Problem.** Arrange  $n$  points in  $[0, 1]^d$  to optimize the dispersion / packing–covering trade-  
 1791    off. The benchmark metric is

1792  
 1793    
$$\text{ratio} = \frac{\min \text{ pairwise distance}}{\max \text{ pairwise distance}},$$

1794  
 1795    so that larger ratio is better (values in  $(0, 1]$ ).

1796    **Evaluator.** Given a program exposing `max_min_dis_ratio(n, d)`, we obtain config-  
 1797    urations for  $(n, d) = (16, 2)$  and  $(14, 3)$ , then report ratio for each case.

1798    **Baseline algorithm.** The initial program employs:

- Enhanced simulated annealing with adaptive cooling,
- Neighbor-repulsion moves,
- Periodic smoothing via  $k$ -NN weighted averages,
- A local refinement stage.

1800  
 1801    KD-tree acceleration is used for nearest-neighbor queries; hyperparameters adapt to dimen-  
 1802    sion.

1803

1804    Initial Program:

```
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
from scipy.spatial.distance import pdist
from scipy.spatial import cKDTree

# (Removed) smooth_points smoothing logic is now inlined to reduce indirection

def calculate_distances(points):
    """Calculates min, max, and ratio of pairwise Euclidean distances using scipy pdist."""
    if points.shape[0] < 2:
        return 0.0, 0.0, 0.0
    distances = pdist(points, metric='euclidean')
    eps = 1e-8
    min_dist = max(np.min(distances), eps)
    max_dist = np.max(distances)
    ratio = max_dist / min_dist
    return min_dist, max_dist, ratio

# (Removed) perturb_point now inlined directly where used

def update_temperature(temperature, cooling_rate, accept_history, iteration, total_iters,
                      initial_temperature, window_size=100):
    """
    Adaptive cooling with acceptancerate feedback and periodic reheating.
    """
    window = accept_history[-min(len(accept_history), window_size):]
    rate = sum(window) / len(window)
    # gentler correction: slow/fast cooling factors reduced
    if rate < 0.2:
        adj = 1.02
    elif rate > 0.8:
        adj = 0.98
    else:
        adj = 1.0
    temperature *= cooling_rate * adj
    # removed periodic reheating to maintain smoother cooling schedule
    # if (iteration + 1) % (total_iters // 4) == 0:
    #     temperature = initial_temperature
    return temperature

def max_min_dis_ratio(n: int, d: int, seed=None):
    """
    Finds n points in d-dimensional space to minimize the max/min distance ratio
  
```

```

1836     using simulated annealing.
1837
1838     Args:
1839         n (int): Number of points.
1840         d (int): Dimensionality of the space.
1841
1842     Returns:
1843         tuple: (best_points, best_ratio)
1844
1845     """
1846
1847     # Adaptive hyperparameters based on dimensionality
1848     iterations = 3000 if d <= 2 else 6000 # increased sweeps for improved convergence
1849     initial_temperature = 10.0
1850     cooling_rate = 0.998 if d <= 2 else 0.996 # slower cooling for extended exploration
1851     perturbation_factor = 0.15 if d <= 2 else 0.12 # tuned smaller steps in 3D for better
1852         local refinement
1853     # relaxation factor for post-acceptance repulsive adjustment
1854     # relaxation_factor removed; using inline 0.1 * perturbation_factor below
1855
1856     # 1. Initial State: reproducible random generator
1857     rng = np.random.default_rng(seed)
1858     # uniform random initialization in [0,1]^d for simplicity
1859     current_points = rng.random((n, d))
1860
1861     #_, current_ratio = calculate_distances(current_points)
1862
1863     best_points = np.copy(current_points)
1864     best_ratio = current_ratio
1865
1866     temperature = initial_temperature
1867     accept_history = []
1868     window_size = 50 # window for stagnation detection and adaptive injection
1869     # smoothing_interval remains, but smoothing_strength is fixed inlined above
1870     smoothing_interval = max(10, iterations // (20 if d <= 2 else 30)) # more frequent
1871         smoothing in 3D for improved uniformity
1872
1873     for i in range(iterations):
1874         # Build KD-tree once per iteration for neighbor queries
1875         tree = cKDTree(current_points)
1876         # optional smoothing step using distance-weighted neighbor smoothing
1877         if (i + 1) % smoothing_interval == 0:
1878             # choose neighbor count based on dimension
1879             k_smooth = 6 if d > 2 else 4
1880             _, idxs = tree.query(current_points, k=k_smooth+1)
1881             neighbors = current_points[idxs[:,1:]] # exclude self
1882             # compute inverse-distance weights
1883             diffs = neighbors - current_points[:, None, :]
1884             dists = np.linalg.norm(diffs, axis=2) + 1e-6
1885             weights = 1.0 / dists
1886             weights /= weights.sum(axis=1, keepdims=True)
1887             neighbor_means = (neighbors * weights[... , None]).sum(axis=1)
1888             blend = 0.6 if d > 2 else 0.7
1889             current_points = np.clip(current_points * blend + neighbor_means * (1 - blend), 0.0,
1890             1.0)
1891             #_, current_ratio = calculate_distances(current_points)
1892             if current_ratio < best_ratio:
1893                 best_points = current_points.copy()
1894                 best_ratio = current_ratio
1895
1896     # 2. Generate Neighboring State: Perturb a random point
1897     # Simplify scaling: rely on temperature to adjust step-size instead of best_ratio
1898     # dynamic perturbation decays sublinearly with temperature for finer local moves
1899     perturbation_strength = perturbation_factor * ((temperature / initial_temperature)**0.6
1900         + 0.15)
1901
1902     # Choose a random point to perturb
1903     point_to_perturb_index = rng.integers(0, n)
1904
1905     old_point = current_points[point_to_perturb_index].copy()
1906     # Increase repulsive move frequency in low dimensions
1907     # dynamic repulsion probability: stronger at high temperature, tapering off as we cool
1908     if d > 2:
1909         # reduce repulsion frequency in 3D for finer refinement
1910         repulsion_prob = float(np.clip(temperature / initial_temperature, 0.2, 0.8))
1911     else:
1912         repulsion_prob = float(np.clip(temperature / initial_temperature + 0.1, 0.5, 0.95))
1913
1914     # start with a random jitter
1915     # random jitter inlined for readability
1916     candidate = old_point + rng.uniform(-perturbation_strength, perturbation_strength,
1917         size=old_point.shape)
1918     if n > 1 and rng.random() < repulsion_prob:

```

```

1890 # compute nearest neighbor via KD-tree for efficiency (reusing prebuilt tree)
1891 _, nn_idx = tree.query(old_point, k=2)
1892 nn_idx = nn_idx[1]
1893 vec = old_point - current_points[nn_idx]
1894 norm = np.linalg.norm(vec)
1895 if norm > 1e-8:
1896     dir_vec = vec / norm
1897     candidate = old_point + perturbation_strength * dir_vec
1898 # keep the point in [0,1]^d
1899 current_points[point_to_perturb_index] = np.clip(candidate, 0.0, 1.0)
1900 _, _, candidate_ratio = calculate_distances(current_points)
1901
1902 # Acceptance criterion
1903 delta = candidate_ratio - current_ratio
1904 accept = (delta < 0) or (rng.random() < np.exp(-delta / temperature))
1905
1906 if accept:
1907     current_ratio = candidate_ratio
1908     # Post-acceptance repulsive relaxation to improve local spacing
1909     # reuse prebuilt KD-tree for repulsive relaxation
1910     dists, idxs_nn = tree.query(current_points[point_to_perturb_index], k=2)
1911     dir_vec = current_points[point_to_perturb_index] - current_points[idxs_nn[1]]
1912     norm = np.linalg.norm(dir_vec)
1913     if norm > 1e-8:
1914         # push away from nearest neighbor
1915         adjustment = 0.1 * perturbation_factor * dir_vec / norm
1916         current_points[point_to_perturb_index] = np.clip(
1917             current_points[point_to_perturb_index] + adjustment, 0.0, 1.0
1918         )
1919         # update ratio and best points after relaxation
1920         _, _, relaxed_ratio = calculate_distances(current_points)
1921         current_ratio = relaxed_ratio
1922         if relaxed_ratio < best_ratio:
1923             best_points = current_points.copy()
1924             best_ratio = relaxed_ratio
1925         # also keep the standard bestcheck for the candidate move
1926         if current_ratio < best_ratio:
1927             best_points = current_points.copy()
1928             best_ratio = current_ratio
1929     else:
1930         current_points[point_to_perturb_index] = old_point
1931
1932     # Update temperature with adaptive schedule
1933     accept_history.append(accept)
1934     temperature = update_temperature(temperature, cooling_rate, accept_history, i,
1935         iterations, initial_temperature)
1936     # periodic mild reheating for 3D to escape deep minima
1937     if d > 2 and (i + 1) % (iterations // 3) == 0:
1938         temperature = max(temperature, initial_temperature * 0.3)
1939
1940     # random injection to escape plateaus: reinitialize one point every 20% of iterations
1941     # random injection only if we've stagnated (low acceptance in recent window)
1942     if (i + 1) % max(1, iterations // 5) == 0 and len(accept_history) >= window_size \
1943         and sum(accept_history[-window_size:]) / window_size < 0.1:
1944         j = rng.integers(0, n)
1945         current_points[j] = rng.random(d)
1946         _, _, current_ratio = calculate_distances(current_points)
1947
1948     # Local refinement stage: fine-tune best solution with small Gaussian perturbations
1949     refine_iters = max(100, iterations // 20)
1950     for _ in range(refine_iters):
1951         idx = rng.integers(0, n)
1952         old_point = best_points[idx].copy()
1953         perturb = rng.normal(0, perturbation_factor * 0.05, size=d)
1954         best_points[idx] = np.clip(old_point + perturb, 0.0, 1.0)
1955         _, _, refined_ratio = calculate_distances(best_points)
1956         if refined_ratio < best_ratio:
1957             best_ratio = refined_ratio
1958         else:
1959             best_points[idx] = old_point
1960     return best_points, best_ratio

```

## I.6 AUTOCONVOLUTION PEAK MINIMIZATION ( $L^\infty$ ).

**Problem Description** For nonnegative densities  $f$  supported on  $[-\frac{1}{2}, \frac{1}{2}]$  with  $\int f = 1$ , define

$$\mu_\infty = \sup_t (f * f)(t).$$

1944 The exact optimum is unknown.  
 1945

1946 **Human Best:**

1947  $0.64 \leq \mu_\infty \leq 0.75496.$

1948 The lower bound is due to Cloninger–Steinerberger, and the upper bound comes from explicit step-  
 1949 function constructions of Matolcsi–Vinuesa (rescaled to unit support).

1950  
 1951 Initial Proposal

1952 **Problem definition.** Let

1954 
$$\mathcal{F} = \left\{ f \in L^1\left([- \frac{1}{2}, \frac{1}{2}]\right) : f \geq 0, \int_{-1/2}^{1/2} f(x) dx = 1 \right\},$$
  
 1955  
 1956

1957 and define

1958 
$$(f * f)(t) = \int_{\mathbb{R}} f(x) f(t - x) dx.$$
  
 1959

1960 We seek to minimize the peak value of the autoconvolution:

1961 
$$\mu_\infty = \inf_{f \in \mathcal{F}} \|f * f\|_\infty.$$
  
 1962

1963 **Constraints.**

- 1964 • Nonnegative density.
- 1965 • Unit mass ( $L^1 = 1$ ).
- 1966 • Support length 1 (here taken as  $[-\frac{1}{2}, \frac{1}{2}]$ ).

1968 In the implementation,  $f$  is represented by nonnegative step heights on a uniform grid and  
 1969 normalized to unit integral.

1970 **Optimization goal.** Minimize

1972 
$$\mu_\infty = \max_t (f * f)(t).$$
  
 1973

1974 Smaller values are better.

1975 **Best-known human results.** In this standard setup, the best currently published bounds  
 1976 are

1977 
$$0.64 \leq \mu_\infty \leq 0.75496.$$

1978 The upper bound traces to work of Matolcsi–Vinuesa (after normalizing support length to  
 1979 1), and the lower bound to Cloninger–Steinerberger.

1980 **Algorithmic goal.** Create an algorithm that constructs feasible densities with progres-  
 1981 sively smaller  $\mu_\infty$ . The baseline program generates simple analytical candidates (box, tri-  
 1982 angle, cosine-squared, Gaussian) on a uniform grid, normalizes to unit mass, and computes  
 1983 autoconvolution via FFT to measure  $\mu_\infty$ . It serves as a starting point for more advanced  
 1984 search/optimization methods.

1985 **References.**

- 1987 • E. P. White, *An optimal  $L^2$  autoconvolution inequality*, Canadian Mathematical  
 1988 Bulletin (2024).
- 1989 • M. Matolcsi and C. Vinuesa, *Improved bounds on the supremum of autoconvolu-  
 1990 tions*, J. Math. Anal. Appl. 372 (2010), 439–447.
- 1991 • A. Cloninger and S. Steinerberger, *On suprema of autoconvolutions with an appli-  
 1992 cation to Sidon sets*, Proc. Amer. Math. Soc. 145 (2017), 3191–3200.

1993 **Initial Program:**

```
1996 # -*- coding: utf-8 -*-
1997 """
1998 Autoconvolution Peak Minimization
1999 =====
```

```
1998
1999 This program generates step heights for a probability density function
2000 that minimizes the maximum value of its autoconvolution.
2001 """
2002
2003
2004 def evaluate_C1_upper_std(step_heights: np.ndarray) -> Dict[str, float]:
2005     """
2006     Standard-normalized C1 (support [-1/2,1/2], dx=1/K).
2007     - Project to feasible set: h >= 0 and f = 1 (L1 normalization).
2008     - Objective: mu_inf = max_t (f*f)(t) (smaller is better).
2009     Returns: {"valid", "mu_inf", "ratio"(=mu_inf), "integral"(=1.0), "K"}
2010     """
2011     h = np.asarray(step_heights, dtype=float)
2012     if h.size == 0 or np.any(h < 0):
2013         return {"valid": 0.0, "mu_inf": float("inf"), "ratio": float("inf")}
2014     K = int(len(h))
2015     dx = 1.0 / K
2016
2017     integral = float(np.sum(h) * dx)
2018     if integral <= 0:
2019         return {"valid": 0.0, "mu_inf": float("inf"), "ratio": float("inf")}
2020     h = h / integral # f = 1
2021
2022     F = np.fft.fft(h, 2*K - 1) # linear autoconvolution via padding
2023     conv = np.fft.ifft(F * F).real
2024     conv = np.maximum(conv, 0.0) # clamp tiny negatives
2025
2026     mu_inf = float(np.max(conv) * dx)
2027     return {"valid": 1.0, "mu_inf": mu_inf, "ratio": mu_inf, "integral": 1.0, "K": float(K)}
2028
2029 def make_candidate(K: int, kind: str = "cos2") -> np.ndarray:
2030     """
2031     Simple candidate builder on [-1/2,1/2] (NOT normalized here).
2032
2033     Args:
2034         K: Number of discretization points
2035         kind: Type of candidate function ("box", "triangle", "cos2", "gauss")
2036
2037     Returns:
2038         Step heights array
2039     """
2040     x = np.linspace(-1.0, 1.0, K)
2041     if kind == "box":
2042         h = np.ones(K)
2043     elif kind == "triangle":
2044         h = 1.0 - np.abs(x)
2045         h[h < 0] = 0.0
2046     elif kind == "cos2":
2047         h = np.cos(np.pi * x / 2.0) ** 2
2048     elif kind == "gauss":
2049         h = np.exp(-4.0 * x**2)
2050     else:
2051         raise ValueError(f"unknown kind={kind}")
2052     return h
2053
2054 def main():
2055     """
2056     Main function that generates step heights for autoconvolution minimization.
2057
2058     Returns:
2059         numpy.ndarray: Step heights array
2060     """
2061     K = 128
2062     kind = "cos2" # Change this to try different candidates (box/triangle/cos2/gauss)
2063     step_heights = make_candidate(K, kind)
2064
2065     # Evaluate the result to verify it's valid
2066     result = evaluate_C1_upper_std(step_heights)
2067     print(f"Generated {kind} candidate with K={K}, mu_inf={result['mu_inf']:.6f}")
2068
2069     return step_heights
```

2052  
2053

## I.7 THIRD AUTOCORRELATION INEQUALITY.

2054  
2055  
2056

**Problem Description** Let  $C_3$  be the largest constant such that  $\max_{|t| \leq 1/2} |(f * f)(t)| \geq C_3 \left( \int_{-1/4}^{1/4} f \right)^2$  for all (signed)  $f$ .

2057  
2058

**Best-known:** classical 1.4581 upper bound.

2059  
2060I.8 THIRD-ORDER AUTOCORRELATION INEQUALITY ( $C_3$  UPPER BOUND)2061  
2062

## Initial Proposal

2063  
2064  
2065

**Problem.** For piecewise-constant nonnegative functions on a fixed support with unit mass, we evaluate an upper bound  $C_{\text{upper\_bound}}$  derived from the maximum of the autoconvolution (normalized by squared  $L^1$  mass). The benchmark score is

2066  
2067  
2068

$$\text{score} = \frac{1}{C_{\text{upper\_bound}}},$$

2069

so that larger score indicates a smaller upper bound and hence a better result.

2070  
2071  
2072

**Evaluator.** The evaluator calls `find_better_c3_upper_bound()` from the target program to obtain step heights, computes the normalized autoconvolution maximum, and returns  $1/C_{\text{upper\_bound}}$ .

2073

**Baseline algorithm.** A simple genetic algorithm over height sequences serves as the baseline search method. The algorithm includes:

2074

- Tournament selection,
- One-point crossover,
- Gaussian mutation.

2075  
2076  
2077

## Initial Program:

2078

```

import scipy.integrate

def calculate_c3_upper_bound(height_sequence):
    N = len(height_sequence)
    delta_x = 1 / (2 * N)

    def f(x):
        if -0.25 <= x <= 0.25:
            index = int((x - (-0.25)) / delta_x)
            if index == N:
                index -= 1
            return height_sequence[index]
        else:
            return 0.0

    integral_f = np.sum(height_sequence) * delta_x
    integral_sq = integral_f**2

    if integral_sq < 1e-18:
        return 0.0

    t_points = np.linspace(-0.5, 0.5, 2 * N + 1)
    max_conv_val = 0.0
    for t_val in t_points:
        lower_bound = max(-0.25, t_val - 0.25)
        upper_bound = min(0.25, t_val + 0.25)

        if upper_bound <= lower_bound:
            convolution_val = 0.0
        else:
            def integrand(x):
                return f(x) * f(t_val - x)

            convolution_val, _ = scipy.integrate.quad(integrand, lower_bound, upper_bound,
                                                       limit=100)
    return 1 / convolution_val

```

```

2106     if abs(convolution_val) > max_conv_val:
2107         max_conv_val = abs(convolution_val)
2108
2109     return max_conv_val / integral_sq
2110
2111 def genetic_algorithm(population_size, num_intervals, generations, mutation_rate,
2112                       crossover_rate):
2113
2114     population = np.random.rand(population_size, num_intervals) * 2 - 1
2115
2116     best_solution = None
2117     best_fitness = 0.0
2118
2119     for gen in range(generations):
2120
2121         fitness_scores = np.array([calculate_c3_upper_bound(individual) for individual in
2122                                     population])
2123
2124         current_best_idx = np.argmax(fitness_scores)
2125         if fitness_scores[current_best_idx] > best_fitness:
2126             best_fitness = fitness_scores[current_best_idx]
2127             best_solution = population[current_best_idx].copy()
2128             # print(f"Generation {gen}: New best fitness = {best_fitness}")
2129
2130
2131         new_population = np.zeros_like(population)
2132         for i in range(population_size):
2133
2134             competitors_indices = np.random.choice(population_size, 2, replace=False)
2135             winner_idx = competitors_indices[np.argmax(fitness_scores[competitors_indices])]
2136             new_population[i] = population[winner_idx].copy()
2137
2138             for i in range(0, population_size, 2):
2139                 if np.random.rand() < crossover_rate:
2140                     parent1 = new_population[i]
2141                     parent2 = new_population[i+1]
2142                     crossover_point = np.random.randint(1, num_intervals - 1)
2143                     new_population[i] = np.concatenate((parent1[:crossover_point],
2144                                                       parent2[crossover_point:]))
2145                     new_population[i+1] = np.concatenate((parent2[:crossover_point],
2146                                                       parent1[crossover_point:]))
2147
2148             for i in range(population_size):
2149                 if np.random.rand() < mutation_rate:
2150                     mutation_point = np.random.randint(num_intervals)
2151                     new_population[i, mutation_point] += np.random.normal(0, 0.1)
2152
2153                     new_population[i, mutation_point] = np.clip(new_population[i, mutation_point],
2154                                                       -2, 2)
2155
2156         population = new_population
2157
2158     return best_solution
2159
2160 def find_better_c3_upper_bound():
2161
2162     NUM_INTERVALS = 4
2163     POPULATION_SIZE = 2
2164     GENERATIONS = 10
2165     MUTATION_RATE = 0.1
2166     CROSSOVER_RATE = 0.8
2167
2168     height_sequence_3 = genetic_algorithm(POPULATION_SIZE, NUM_INTERVALS, GENERATIONS,
2169                                           MUTATION_RATE, CROSSOVER_RATE)
2170
2171     return height_sequence_3

```