

# Machine Learning Course Project Report

Yanru Guan   Yuetian Wu   Yang Zhan   Zhi Zhang

School of Information Science and Technology

{2300013051, 2200013172, 2300013063, 2200013216}@stu.pku.edu.cn

**Abstract**—This report studies the application of large language model (LLM)-based program evolution to geometric extremal problems, focusing on optimizing the configuration of  $n = 18$  points in the plane by minimizing the ratio of the maximum pairwise distance to the minimum pairwise distance. This is a typical "easy-to-verify, hard-to-construct" optimization task, where traditional analytic or heuristic methods often struggle to surpass known optima. Recent works such as FunSearch and AlphaEvolve propose generating and evolving programs with LLMs to search in program space, providing a new paradigm for such problems. However, the current open-source implementation OpenEvolve still underutilizes historical experience and is prone to local optima caused by symmetric constructions.

To address this issue, we propose an Experience Replay Memory (ERM) mechanism on top of OpenEvolve, guiding the LLM to explicitly summarize modification experience between parent and child programs and reuse it in subsequent evolution, thereby improving efficiency and stability. Under a unified experimental setting, we compare the original OpenEvolve with the ERM-augmented method for  $n = 18$ . Results show that ERM significantly accelerates convergence and continues to approach the optimum. With further human feedback, our program achieves a construction numerically comparable to the known optimum and nearly identical in geometric structure. This work demonstrates the effectiveness of incorporating structured historical experience in program-evolution LLM frameworks and shows the practical potential of human-LLM collaborative evolution to reach optimal solutions, offering useful experience for larger-scale or more complex geometric optimization problems.

## I. Introduction

### A. Background

For "easy-to-verify, hard-to-construct" problems, traditional methods often rely on hand-designed heuristics or exhaustive search, with clearly limited efficiency and scalability. In recent

years, with the rise of LLMs and mature automated evaluation, these problems have shifted toward an "LLM-generate-and-verify" paradigm, where explicit verification rules are used to filter and optimize results.

However, directly asking an LLM to produce a final solution is often constrained by hallucinations, numerical instability, and difficulty with local modifications. To address this, researchers explore having LLMs generate programs that in turn generate solutions. FunSearch proposes a "program population + automated evaluation + iterative rewriting" framework, enabling evolutionary search in program space and achieving new results on multiple mathematical and algorithmic problems [1]. Building on this, AlphaEvolve systematizes and extends the paradigm and introduces several geometric and combinatorial optimization benchmarks, including the "minimize max/min pairwise distance ratio" problem studied here [2]. The open-source implementation OpenEvolve then provides a practical engineering framework for reproduction, analysis, and further improvements [3].

### B. Topic and Problem Definition

We study configurations of  $n = 18$  points in the plane that minimize the ratio between the maximum and minimum pairwise distances. Specifically, given a point set  $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^2$ , define

$$d_{\min}(X) = \min_{i < j} \|x_i - x_j\|_2, \quad d_{\max}(X) = \max_{i < j} \|x_i - x_j\|_2, \quad (1)$$

The objective is to minimize

$$r(X) = \frac{d_{\max}(X)}{d_{\min}(X)}. \quad (2)$$

This problem is a concrete instance of a classic geometric extremal question, with early results

dating back to Bateman–Erdős (1951)[4] and Bezdek–Fodor (1999)[5]. The current best-known results are compiled online [6]. For the 2D  $n = 18$  case, the best known result[6] is  $r^2 = 14.725+$  (Cantrell, 2009), which we use as the reference baseline.

Notably, for other scales, substantial progress has been driven by LLM-based code generation frameworks. AlphaEvolve improved existing results for 2D  $n = 16$  and 3D  $n = 14$ . For 2D  $n = 16$ , two nearly simultaneous works in October 2025, CodeEvolve [7] and FM Agent [8], further improved the result from  $\sqrt{12.88927}$  to  $\sqrt{12.88923}$  with consistent numeric values and constructions. This suggests that for this scale, program evolution and LLM-driven search are converging to the same high-quality solution.

In contrast, for  $n = 18$ , current LLM code generation frameworks cannot match the known optimum; most of them stagnate at local optima under symmetric constructions. Figure 1 shows the known optimum and the solution produced by vanilla OpenEvolve.

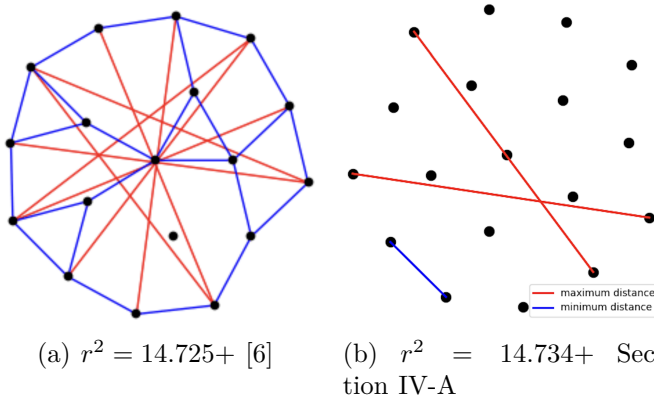


Fig. 1: Comparison for  $n = 18$ : (a) the current best-known solution, (b) the solution from vanilla OpenEvolve.

### C. Contributions

For the problem of minimizing the max/min pairwise distance ratio of 18 points in 2D, our main contributions are:

- 1) Introduce an Experience Replay Memory (ERM) mechanism in OpenEvolve, prompting the LLM to explicitly summarize program modification experience and condition

future evolution, improving the use of historical information.

- 2) Compare methods with and without ERM under a unified setup; results show ERM yields stable and significant gains in convergence speed and proximity to the optimum.
- 3) Build a human-in-the-loop iteration process on top of the method, achieving a configuration nearly identical to the best-known solution, with a max/min distance ratio comparable to the optimum.

## II. Problem Setup

We require the LLM to output Python code implementing the function `min_max_dist_dim2_18()`, which returns the coordinate array of  $n = 18$  2D points  $X = \{x_i\}_{i=1}^{18} \subset \mathbb{R}^2$ . The output must be a `numpy.ndarray` with shape  $(18, 2)$ ; if the shape is incorrect or execution fails, the evaluation is marked as failed and returns a score of 0 (corresponding to `combined_score=0`).

The evaluator computes all pairwise Euclidean distances for the output point set, yielding  $d_{\min}(X)$  and  $d_{\max}(X)$  as defined in (1). Our original optimization target is to minimize the squared ratio

$$r^2(X) = \left( \frac{d_{\max}(X)}{d_{\min}(X)} \right)^2. \quad (3)$$

But to be compatible with OpenEvolve’s ”maximize fitness” form, the evaluator returns the reciprocal

$$\text{min\_max\_ratio}(X) = \left( \frac{d_{\min}(X)}{d_{\max}(X)} \right)^2 = \frac{1}{r^2(X)}, \quad (4)$$

Larger values are better.

## III. Method

### A. Warm-start: vanilla OpenEvolve

In practice, we find that using a reasonably good non-trivial program as the initial program improves iteration efficiency. Therefore, we first run OpenEvolve without any changes to obtain an effective warm-start solution. Based on this warm-start, we use it as the initial program and apply our modified search framework for further optimization.

## B. Experience Replay Memory (ERM)

We observe that OpenEvolve underutilizes historical records: it only shows the LLM some top and diverse solutions, without leveraging modification experience between parent and child programs. Based on this observation, we propose an Experience Replay Memory mechanism, which asks the LLM to summarize its modification ideas in short natural-language text and stores them in an experience bank along with the score difference between parent and child. When a parent program is selected again, we retrieve its historical experience and provide it with the current prompt to guide subsequent evolution. The pipeline is shown in Fig. 2.

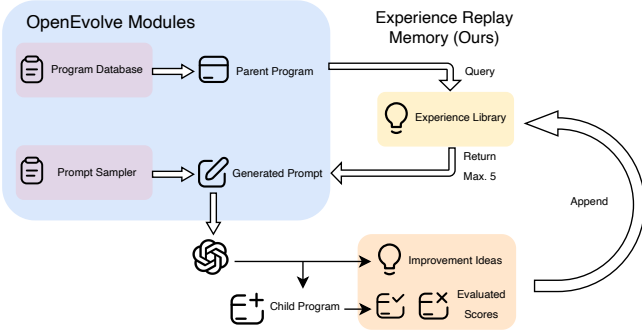


Fig. 2: Method pipeline

## IV. Experiment

### A. Warm-start: vanilla OpenEvolve

**Training procedure.** In this experiment, we use the DeepSeek-V3.2 model [9]. We start with a program that randomly samples 18 points and normalizes them, and train for 10 iterations to obtain a reasonably good non-random symmetric construction. However, since symmetry leads to local optima, we select a random algorithmic solution with a score slightly below the symmetric construction as the initial program for the second stage. The second-stage hyperparameters are: `max_iterations=80`; `num_islands=5`; `migration_interval=10`; `exploitation_ratio=0.6`; and `population_size=70`. Each round allows the LLM to fully rewrite the program to encourage exploration. The prompt encourages OpenEvolve to break symmetry and use randomized sampling, physical simulation, gradient optimization, and other methods; see Appendix C.

**Results.** The 2D visualization is shown in Fig. 3, and the coordinates of the solution are listed in Appendix A.

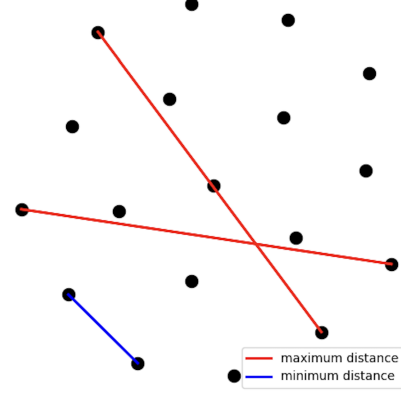


Fig. 3:  $r^2 = 14.734+$

### B. Experience Replay Memory

**Experimental setup.** For all baselines, we use identical experimental settings to ensure fairness. Specifically, all experiments use the best program from Section IV-A as the initial program; the maximum iterations are `max_iterations=40`, with `num_islands=5`, `migration_interval=10`, `exploitation_ratio=0.6`, and `population_size=70`. Each iteration allows the LLM to fully rewrite the program to maximize exploration. All experiments use the same DeepSeek-V3.2 model [9] and consistent sampling parameters.

With ERM enabled, we retain up to 5 historical experience items for each parent program and provide them along with the current prompt to the LLM. The corresponding prompt template is given in Appendix C.

Under this unified setup, we compare two methods: OpenEvolve w.o. ERM, the original OpenEvolve without experience replay; and OpenEvolve w. ERM, OpenEvolve augmented with Experience Replay Memory. The only difference between the two experiments is whether ERM is enabled.

**Results.** The trajectory of best historical scores is shown in Fig. 4. As shown, adding ERM significantly improves both the speed and magnitude of program evolution.

We further visualize the final best solutions from both methods in Fig. 5. After introducing ERM, the search framework effectively ab-

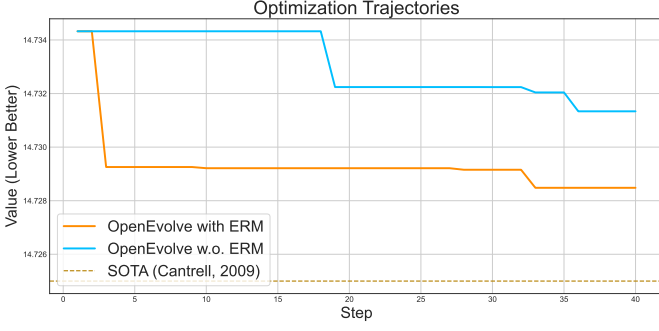


Fig. 4: Best historical scores for OpenEvolve w./w.o. ERM (here the score is  $r^2(X) = d_{\max}(X)^2/d_{\min}(X)^2$ , smaller is better)

sorbs and reuses historical modification experience, converging faster to high-quality extremal configurations. The best program generated by OpenEvolve w. ERM and the approximate coordinates of its solution are given in Appendix D and B.

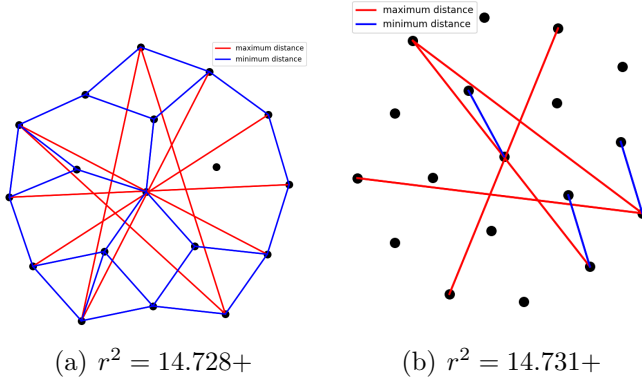


Fig. 5: Comparison of best solutions: (a) with ERM, (b) vanilla OpenEvolve.

### C. Case study

Escaping symmetric solutions. In the early training phase starting from a random-sampling initial program, we found that OpenEvolve obtained a fairly good symmetric solution in the second iteration ( $d_{\max}/d_{\min} = \sqrt{14.928+}$ ), far better than later solutions that used randomness and optimization. OpenEvolve then became stuck in a local optimum, repeatedly making meaningless tweaks to the symmetric solution. We therefore intervened manually, selected a better-performing asymmetric construction as the initial program, and launched a new stage of training. We also

encouraged breaking symmetry and exploring more diverse algorithms in the `system_message` of the prompt.

Human-in-the-loop optimization. In addition to the automated evolution method above, we introduce human feedback to combine the LLM’s search breadth with human geometric intuition. We construct the following iterative process:

- 1) Run multiple evolution rounds with the improved framework to converge to the current best program;
- 2) Manually fine-tune the code based on geometric symmetry or gradient information for the configuration generated by the best program;
- 3) Use the manually corrected program as a new initial population (warm-start) and run the next evolution round.

After multiple iterations, we obtain  $r^2 = 14.7259776+$ , extremely close to the known optimum  $r^2 = 14.725+$ .

We also visualize the human-refined solution. Notably, after rotation, the human solution is nearly identical to the known optimum; see Fig. 6.

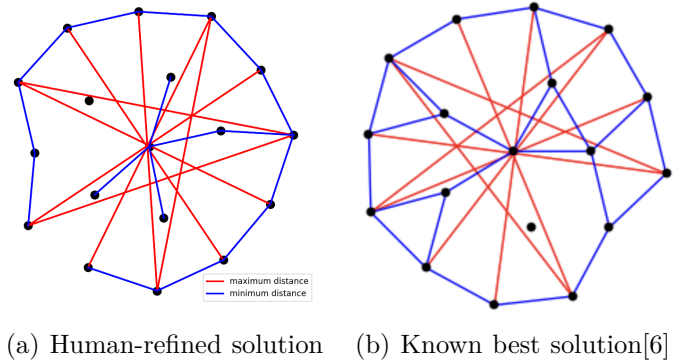


Fig. 6: Rotating the human solution counterclockwise by  $135^\circ$  yields a configuration similar to the known optimum

### V. Conclusion

This work studies the performance and limitations of LLM-based program evolution on the extremal geometry problem of minimizing the max/min distance ratio for  $n = 18$  points in the plane. Through reproduction experiments, we verify that the existing OpenEvolve framework easily falls into local optima caused by symmetric

constructions at this scale, making it hard to reach the known optimum.

To mitigate this, we introduce an Experience Replay Memory (ERM) mechanism during program evolution. By explicitly recording and reusing modification experience between parent and child programs, the LLM can leverage historical information more effectively in later search. Experiments show that, with the same model, hyperparameters, and initial program, ERM significantly improves convergence speed and repeatedly pushes solutions toward the current known optimum, validating its effectiveness in program-evolution LLM frameworks.

Although our method has not yet surpassed the current optimum for  $n = 18$ , experiments and case analyses show that reasonable modeling of historical experience is crucial for escaping local optima and improving search stability. Further, when combined with human feedback in a collaborative evolution process, the framework can produce configurations nearly identical to the known optimum at this scale. Overall, this study provides a feasible path for explicitly introducing "memory and experience" into LLM-driven program evolution, and offers useful practical guidance for future work.

## VI. Future Directions

Our study shows that in LLM-driven program evolution frameworks, effective modeling and use of historical search experience significantly impact optimization performance. Future research can proceed along the following directions.

- 1) First, at the experience modeling level, explore higher-level abstractions, such as summarizing repeated successes or failures into reusable strategy patterns rather than only local experience tied to a single parent program, to improve generalization.
- 2) Second, at the evolution control level, study how to dynamically adjust experience usage across search stages, encouraging diversity early and emphasizing high-value experience later to achieve a better exploration-exploitation balance.
- 3) Third, at the reasoning paradigm level, we observe that appropriate human intervention (e.g., providing heuristics when generated programs are too similar over many

steps) can further improve program quality and find solutions comparable to SOTA. Future work can explore human-in-the-loop reasoning paradigms.

- 4) Finally, extending the proposed experience replay mechanism to higher dimensions, larger scales, or different types of geometric and combinatorial optimization problems is a natural and challenging direction.

## References

- [1] B. Romera-Paredes, M. Barekatin, A. Novikov, M. Balog et al., “Mathematical discoveries from program search with large language models,” *Nature*, vol. 625, pp. 468–475, 2023.
- [2] A. Novikov et al., “Alphaevolve: A coding agent for scientific and algorithmic discovery,” *arXiv preprint arXiv:2506.13131*, 2025.
- [3] A. Sharma, “Openevolve: an open-source evolutionary coding agent,” 2025. [Online]. Available: <https://github.com/algorithmicsuperintelligence/openevolve>
- [4] P. Bateman and P. Erdős, “Geometrical extrema suggested by a lemma of besicovitch,” *The American Mathematical Monthly*, vol. 58, no. 5, pp. 306–314, 1951.
- [5] A. Bezdek and F. Fodor, “Minimal diameter of certain sets in the plane,” *Journal of Combinatorial Theory, Series A*, vol. 85, no. 1, pp. 105–111, 1999.
- [6] “Minimizing the ratio of maximum to minimum distance,” Webpage, <https://erich-friedman.github.io/packing/maxmin/>.
- [7] H. Assumpção, D. Ferreira, L. Campos, and F. Murai, “Codeevolve: an open source evolutionary coding agent for algorithm discovery and optimization,” 2026. [Online]. Available: <https://arxiv.org/abs/2510.14150>
- [8] A. Li et al., “The fm agent,” 2025. [Online]. Available: <https://arxiv.org/abs/2510.26144>
- [9] DeepSeek-AI et al., “Deepseek-v3.2: Pushing the frontier of open large language models,” 2025. [Online]. Available: <https://arxiv.org/abs/2512.02556>

## Appendix A

### Coordinates for the Warm-start Solution

(0.1094142263, 0.2108139917), (0.2700115844, 0.0516945663),  
 (0.4942253501, 0.0225873470), (0.6964263647, 0.1239094508),  
 (0.2273797922, 0.4036979489), (0.3956427368, 0.2412136928),  
 (0.6362636728, 0.3418531817), (0.8586453069, 0.2815345212),  
 (0.0002296726, 0.4088099459), (0.4454821839, 0.4632571120),  
 (0.6076506307, 0.6208951318), (0.7983729788, 0.4994676267),  
 (0.1181325282, 0.6017486981), (0.3424201955, 0.6645605860),  
 (0.6175381717, 0.8468122799), (0.8082194226, 0.7253345767),  
 (0.1780101843, 0.8198429154), (0.3946354019, 0.8846425612)

## Appendix B

### Coordinates for the Experience Replay Memory Solution

(0.7205003052, 0.5835684296), (0.4997217542, 0.5049012391),  
 (0.2812993472, 0.5755498658), (0.3680717307, 0.3168376786),  
 (0.9493426197, 0.5273803499), (0.5241407725, 0.7331635330),  
 (0.3080091775, 0.8105375853), (0.1432607565, 0.2703624830),  
 (0.5207253701, 0.1453836429), (0.6523754945, 0.3334469426),  
 (0.8839310912, 0.7474277540), (0.4822617682, 0.9599886505),  
 (0.0692254227, 0.4876608476), (0.2959150220, 0.0989085718),  
 (0.8805649382, 0.3083615519), (0.6983936385, 0.8826144688),  
 (0.0991479534, 0.7152661895), (0.7489145925, 0.1202980874)

## Appendix C

### Prompts Used

#### Listing 1: System message in the OpenEvolve prompt configuration

SETTING:

You are an expert computational geometer and optimization specialist focusing on point dispersion problems. Your task is to evolve a constructor function that generates an optimal arrangement of exactly 18 points in 2D space, maximizing the ratio of minimum distance to maximum distance between all point pairs.

PROBLEM CONTEXT:

- Target: Beat the SOTA found by mathematician David Cantrell of min/max ratio =  $\sqrt{1/14.725}$  0.2605987

- Constraint: Points must be placed in 2D Euclidean space (typically normalized to unit square  $[0,1] \times [0,1]$ )
- Mathematical formulation: For points  $P_i = (x_i, y_i)$ ,  $i = 1, \dots, 16$ :
  - \* Distance matrix:  $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$  for all  $ij$
  - \* Minimum distance:  $d_{\min} = \min\{d_{ij} : ij\}$
  - \* Maximum distance:  $d_{\max} = \max\{d_{ij} : ij\}$
  - \* Objective: maximize  $d_{\min}/d_{\max}$  subject to spatial constraints

#### PERFORMANCE METRICS:

1. **\*\*min\_max\_ratio\*\***:  $d_{\min}/d_{\max}$  ratio (PRIMARY OBJECTIVE - maximize)
2. **\*\*combined\_score\*\***:  $\min\_max\_ratio / \text{SOTA}$  (progress toward beating SOTA)

#### TECHNICAL REQUIREMENTS:

- **\*\*Reproducibility\*\***: Fixed random seeds for all stochastic components

#### HINTS:

- Symmetry doesn't always give the best result. Current SOTA is not a symmetric construction. Don't be afraid to be explore different structures.
- Running time is not a big deal, feel free to write code that runs longer, as long as it isn't unacceptably slow (e.g. takes 20+ seconds to finish).
- Maybe try various randomized optimization methods e.g. simulated annealing or physics-inspired methods such as force-based simulation. For example, add a force between two points if they are too slow or too far part to make the distance tend to a same value.

Focus on breaking through the plateau by trying fundamentally different approaches - don't just tweak parameters.

### Listing 2: Template example for instructing the LLM to output improvement ideas and the program

```
# Current Program Information
- Fitness: {fitness_score}
- Feature coordinates: {feature_coords}
- Focus areas: {improvement_areas}

{artifacts}

# Program Evolution History
{evolution_history}

# Current Program
““{language}
{current_program}
““

# Task
Rewrite the program to improve its FITNESS SCORE.
The system maintains diversity across these dimensions: {feature_dimensions}
Different solutions with similar fitness but different features are valuable.
Provide the complete new program code.
Also include a '## Experience' section with 1-3 SHORT bullet points describing the
change rationale and expected outcome (<= 200 chars total).

IMPORTANT: Make sure your rewritten program maintains the same inputs and outputs
as the original program, but with improved internal implementation.

““{language}
# Your rewritten program here
““
```

### Listing 3: Template example for providing historical experience to the LLM

```
## Recent Experience (Parent)

### Experience {idx} (delta: {delta_str}, child: {child_id})
- Changes: {changes}
Notes:
{notes}
```

## Appendix D

### Code for Experience Replay Memory

```
import numpy as np

def min_max_dist_dim2_18() -> np.ndarray:
    """
    Returns 18 points in 2D maximizing the ratio of minimum to maximum pairwise distance.

    This version implements a focused approach:
    1. Single high-quality asymmetric 4-6-5-3 initialization (proven best)
    2. Simplified gradient optimization targeting only exact extremal pairs
    3. Extended refinement with small step sizes
    4. Strategic perturbations and optimal scaling
    """
    np.random.seed(20240711)
    n = 18
    idx_i, idx_j = np.triu_indices(n, k=1)

    def distances(points):
        return np.linalg.norm(points[idx_i] - points[idx_j], axis=1)

    def ratio(points):
        d = distances(points)
        dmin = np.min(d)
        dmax = np.max(d)
        return dmin / dmax, dmin, dmax

    # Use only the proven best asymmetric 4-6-5-3 initialization
    center = np.array([0.5, 0.5])
    pts = np.zeros((n, 2))
    counts = [4, 6, 5, 3]
    radii = [0.13, 0.29, 0.44, 0.58]
    offset = 0
    for ring in range(4):
        cnt = counts[ring]
        for j in range(cnt):
            angle = 2*np.pi*j/cnt + 0.3*ring + np.random.uniform(-0.04, 0.04)
            pts[offset] = center + radii[ring] * np.array([np.cos(angle), np.sin(angle)])
            offset += 1
            if offset >= n:
                break
        if offset >= n:
            break
    pts = np.clip(pts, 0.05, 0.95)

    best_pts = pts.copy()
    best_score, best_dmin, best_dmax = ratio(pts)

    # Phase 1: Extended gradient optimization focusing only on exact extremal pairs
    n_iter1 = 200000
    step_init = 0.001

    for it in range(n_iter1):
        d = distances(pts)
        dmin = np.min(d)
        dmax = np.max(d)

        # Find all extremal pairs with tolerance
        min_indices = np.where(np.abs(d - dmin) < 1e-12)[0]
        max_indices = np.where(np.abs(d - dmax) < 1e-12)[0]
```

```

grad = np.zeros_like(pts)

if len(min_indices) > 0:
    for idx in min_indices:
        i, j = idx_i[idx], idx_j[idx]
        vec = pts[i] - pts[j]
        dist = d[idx] + 1e-12
        dir_vec = vec / dist
        grad[i] += dir_vec / (dmax * len(min_indices))
        grad[j] -= dir_vec / (dmax * len(min_indices))

if len(max_indices) > 0:
    for idx in max_indices:
        i, j = idx_i[idx], idx_j[idx]
        vec = pts[i] - pts[j]
        dist = d[idx] + 1e-12
        dir_vec = vec / dist
        factor = dmin / (dmax ** 2 * len(max_indices))
        grad[i] -= factor * dir_vec
        grad[j] += factor * dir_vec

step = step_init * (1 - it / n_iter1) ** 1.5 + 1e-6
pts += step * grad
pts = np.clip(pts, 0, 1)

curr_score, curr_dmin, curr_dmax = ratio(pts)
if curr_score > best_score + 1e-12:
    best_score, best_dmin, best_dmax = curr_score, curr_dmin, curr_dmax
    best_pts = pts.copy()

# Strategic perturbation
if it % 5000 == 0 and it > 10000:
    perturbation = np.random.normal(0, 0.002 * (1 - it / n_iter1), pts.shape)
    pts += perturbation
    pts = np.clip(pts, 0, 1)

pts = best_pts.copy()

# Phase 2: Intensive refinement with even smaller steps
n_iter2 = 200000

for it in range(n_iter2):
    d = distances(pts)
    dmin = np.min(d)
    dmax = np.max(d)

    min_indices = np.where(np.abs(d - dmin) < 1e-12)[0]
    max_indices = np.where(np.abs(d - dmax) < 1e-12)[0]

    grad = np.zeros_like(pts)

    if len(min_indices) > 0:
        for idx in min_indices:
            i, j = idx_i[idx], idx_j[idx]
            vec = pts[i] - pts[j]
            dist = d[idx] + 1e-12
            dir_vec = vec / dist
            grad[i] += dir_vec / (dmax * len(min_indices))
            grad[j] -= dir_vec / (dmax * len(min_indices))

    if len(max_indices) > 0:
        for idx in max_indices:
            i, j = idx_i[idx], idx_j[idx]
            vec = pts[i] - pts[j]

```

```

        dist = d[idx] + 1e-12
        dir_vec = vec / dist
        factor = dmin / (dmax ** 2 * len(max_indices))
        grad[i] -= factor * dir_vec
        grad[j] += factor * dir_vec

    step = 0.0004 * (1 - it / n_iter2) ** 1.5 + 5e-7
    pts += step * grad
    pts = np.clip(pts, 0, 1)

    curr_score, curr_dmin, curr_dmax = ratio(pts)
    if curr_score > best_score + 1e-12:
        best_score, best_dmin, best_dmax = curr_score, curr_dmin, curr_dmax
        best_pts = pts.copy()

pts = best_pts.copy()

# Optimal scaling and translation
min_coords = np.min(pts, axis=0)
max_coords = np.max(pts, axis=0)
span = max_coords - min_coords
max_span = np.max(span)

if max_span > 1e-8:
    center_orig = (min_coords + max_coords) / 2
    best_scaled = pts.copy()

    for scale in [0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98]:
        scaled = center_orig + (pts - center_orig) * (scale / max_span)
        scaled = np.clip(scaled, 0, 1)
        scaled_score, _, _ = ratio(scaled)
        if scaled_score > best_score:
            best_score = scaled_score
            best_scaled = scaled.copy()

# Fine translation search
offsets = np.linspace(-0.003, 0.003, 7)
for dx in offsets:
    for dy in offsets:
        translated = best_scaled + np.array([dx, dy])
        translated = np.clip(translated, 0, 1)
        trans_score, _, _ = ratio(translated)
        if trans_score > best_score:
            best_score = trans_score
            best_scaled = translated.copy()

pts = best_scaled

# Final micro-optimization (1000 very small steps)
for _ in range(1000):
    d = distances(pts)
    dmin = np.min(d)
    dmax = np.max(d)

    min_idx = np.argmin(d)
    max_idx = np.argmax(d)
    i_min, j_min = idx_i[min_idx], idx_j[min_idx]
    i_max, j_max = idx_i[max_idx], idx_j[max_idx]

    step = 1e-6

    # Adjust minimum pair
    vec_min = pts[i_min] - pts[j_min]
    if np.linalg.norm(vec_min) > 1e-12:
        dir_min = vec_min / np.linalg.norm(vec_min)

```

```

    pts[i_min] += dir_min * step / dmax
    pts[j_min] -= dir_min * step / dmax

    # Adjust maximum pair
    vec_max = pts[i_max] - pts[j_max]
    if np.linalg.norm(vec_max) > 1e-12:
        dir_max = vec_max / np.linalg.norm(vec_max)
        factor = dmin / (dmax ** 2)
        pts[i_max] -= dir_max * step * factor
        pts[j_max] += dir_max * step * factor

    pts = np.clip(pts, 0, 1)

    # Final check
    final_score, __, __ = ratio(pts)
    if final_score < best_score - 1e-12:
        pts = best_pts.copy()

    return pts

```