RADONBENCH: A BENCHMARK OF CLASSICAL AND LEARNING-GUIDED SEARCH TECHNIQUES FOR PROBLEMS IN COMBINATORIAL GEOMETRY

Anonymous authors
Paper under double-blind review

ABSTRACT

Classical optimization and learning-guided search offer complementary approaches to open problems in discrete mathematics, but their outputs are difficult to compare when search spaces, validity checks, and stopping rules differ. We present *RadonBench*, a benchmark for constructing maximum-cardinality Radon-free lattice configurations: finite subsets of $\mathbb{Z}^d$ admitting no integral Radon partition. It remains open whether Radon-free configurations with 11 points in $\mathbb{Z}^3$ or 21 points in $\mathbb{Z}^4$ are possible. If they exist, they would improve the corresponding lower bounds on the integer Radon numbers. We compare the following methods: exhaustive extension, greedy local search, a task-adapted PatternBoost solver, AlphaZero-Radon, a single-agent AlphaZero-style MCTS solver with a task-specific set-attention network, and AlphaEvolve-style program evolution. All outputs use one success predicate: a convex target-sized configuration succeeds exactly when its number of integral Radon points is zero. PatternBoost and AlphaZero-Radon construct verified Radon-free configurations of sizes 10 and 20, one point short of the respective targets; AlphaZero-Radon's best convex target terminals have two integral Radon points in both dimensions. For 50 fixed Radon-free seeds per dimension, exhaustive extension tests every added point in its declared boxes ($159{,}500$ 3D and $499{,}000$ 4D extensions) and also reaches a minimum of two integral Radon points. From 1,000 convex target-sized starts per dimension, greedy local search reaches one integral Radon point in 27 3D and 25 4D runs. One 20-round AlphaEvolve-style program-evolution run per dimension also reaches one integral Radon point. No audited run reaches zero. These results establish reproducible near-miss baselines over explicitly scoped search domains; they do not establish nonexistence.

1 INTRODUCTION

Mathematical discovery is rapidly changing with tools from AI. Classical enumeration and local optimization, neural construction, search with learned policies, and evaluator-guided program evolution have all proved useful for combinatorial discovery (Wagner, 2021; Fawzi et al., 2022; Charton et al., 2024; Novikov et al., 2025). Despite this progress, many basic combinatorial quantities, including numerous Ramsey numbers, remain beyond the reach of both conventional computation and current AI-assisted methods (Radziszowski, 1994; McKay & Radziszowski, 1995; Exoo, 1989). Here we investigate another such problem and use it to benchmark modern search techniques.

Radon's theorem is a fundamental result in convex and discrete geometry: any $d+2$ points in $\mathbb{R}^d$ admit a bipartition whose convex hulls intersect (Bárány, 2021; Barvinok, 2002; Gruber, 2007). Motivated by applications in integer programming and combinatorics, researchers have studied integer versions of Radon's theorem (Onn, 1991; Eckhoff, 1993; De Loera et al., 2017; 2019). Here the points have integer coordinates, and the intersection of the two convex hulls must contain a lattice point. This gives an *integral Radon partition*. A finite set in $\mathbb{Z}^d$ with no such partition is *Radon-free*, and the *integer Radon number* $r(\mathbb{Z}^d)$ is the smallest cardinality for which every lattice point configuration has an integral Radon partition. The exact value in the plane is $r(\mathbb{Z}^2) = 6$, but it remains unknown in higher dimensions (Onn, 1991; Bezdek & Blokhuis, 2003). We study whether there is a Radon-free configuration with 11 points in $\mathbb{Z}^3$ or one with 21 points in $\mathbb{Z}^4$.

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

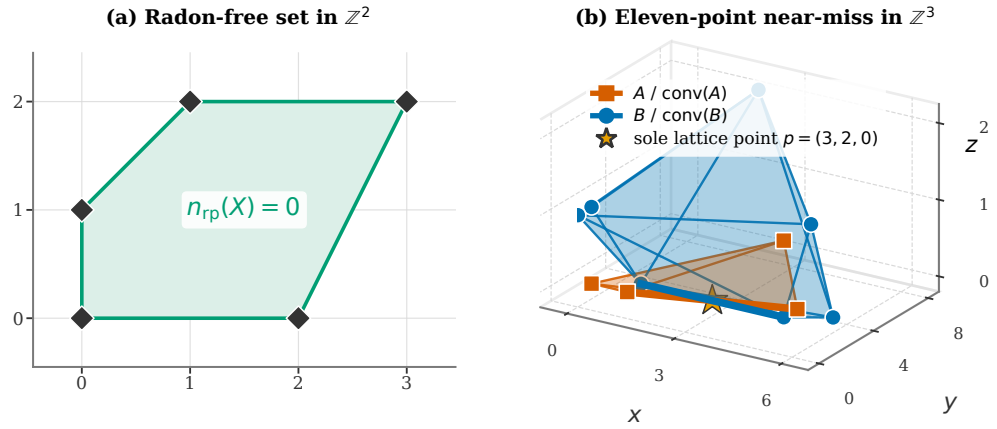

Figure 1: **The search problem.** (a) Onn's five-point configuration in $\mathbb{Z}^2$ is Radon-free (Onn, 1991). (b) An 11-point near-miss in $\mathbb{Z}^3$ found by the declared local-search experiment. The displayed split has the single shared lattice point $p = (3, 2, 0)$, hence $n_{\mathrm{rp}} = 1$; success requires $n_{\mathrm{rp}} = 0$.

The search for Radon-free configurations offers a compact comparison setting: candidate verification is practical, yet the configuration space is already too large for naive enumeration at the open targets. Prior work established structural bounds and recognition complexity for integral Radon partitions (Onn, 1991; Bezdek & Blokhuis, 2003), but those authors did not systematically study constructive search near the open lattice frontiers.

We introduce *RadonBench*, a benchmark of five method families: exhaustive extension, greedy local search, a task-adapted PatternBoost solver, AlphaZero-Radon (our AlphaZero-style solver), and AlphaEvolve-style program evolution. These methods instantiate two complementary search paradigms. *Cardinality constructors* grow configurations point by point while maintaining Radon-freeness at every proper prefix; PatternBoost and AlphaZero-Radon belong primarily to this group. *Fixed-target optimizers* instead evaluate complete N-point configurations and minimize their number of integral Radon points; this group comprises exhaustive extension, greedy local search, and AlphaEvolve-style program evolution. AlphaZero-Radon is construction-oriented but also scores convex N-point terminals, so it contributes to the fixed-target comparison whenever it reaches the horizon. The methods do not search one common finite domain and their computational units are not matched. We therefore compare verified outputs rather than claim a ranking of search efficiency. In particular, we separate two questions:

RQ1 Does any declared protocol produce a verifier-accepted configuration at an open target?

RQ2 If a run is unsuccessful, what does it nevertheless certify: the largest verified Radon-free cardinality $k_{\max}$, or the minimum target-sized Radon-point count $r_{\min}(N)$?

This separation prevents a legal $(N-1)$-point prefix from being treated as a scored N-point candidate. Explicit 10-point and 20-point witnesses provide finite-box positive controls immediately below the two frontiers; smaller cardinalities need not be benchmarked separately because Radon-freeness is hereditary.

Our contributions are twofold. First, we define a common success predicate, graded score, and reporting contract for search methods with different proposal spaces. Second, we give an auditable outcome comparison across five classical and learning-guided method families in $\mathbb{Z}^3$ and $\mathbb{Z}^4$, while making each method's reachable configurations and stopping conditions explicit. Verifier derivations, hyperparameter tables, search-space diagnostics, and additional experiments are provided in the appendices.

No audited run reported here resolves either open frontier. All negative observations are limited to the candidates or finite domains searched by the declared protocols; they are not non-existence results for $\mathbb{Z}^3$ or $\mathbb{Z}^4$.

Table 1: Open targets, known feasible configurations immediately below them, and the coordinate domain used by each method. Box dimensions are side lengths. "No finite box" means that the evaluator accepts arbitrary finite integer coordinates. The method-specific domains are not box-matched.

Instance	Known feasible configuration	Method	Coordinate domain
$\mathbb{Z}^3$, target 11	10-point Radon-free	Exhaustive extension	Added point: $20\times20\times8$ box
		Greedy local search	All points: $20\times20\times5$ box
		PatternBoost	All points: $20\times20\times5$ box
		AlphaZero-Radon	All points: $20\times20\times5$ box
		AlphaEvolve-style	No finite box
$\mathbb{Z}^4$, target 21	20-point Radon-free	Exhaustive extension	Added point: $20\times20\times5\times5$ box
		Greedy local search	All points: $20\times20\times5\times5$ box
		PatternBoost	All points: $6\times6\times6\times6$ box
		AlphaZero-Radon	All points: $6\times6\times6\times6$ box
		AlphaEvolve-style	No finite box

2 Problem and Evaluation

Let $X \subset \mathbb{Z}^d$ be a finite set of distinct points. An *integral Radon partition* is a split $X = A \sqcup B$ into nonempty parts satisfying $\mathrm{conv}(A) \cap \mathrm{conv}(B) \cap \mathbb{Z}^d \neq \varnothing$. We call every lattice point in such an intersection an integral Radon point and write

$$n_{\mathrm{rp}}(X) = \#\{p \in \mathbb{Z}^d : p \text{ is an integral Radon point of } X\}.$$

Thus X is Radon-free exactly when $n_{\mathrm{rp}}(X) = 0$. Every such configuration is in convex position: if $x \in \mathrm{conv}(X\backslash\{x\})$, then $\{x\}\sqcup(X\backslash\{x\})$ already has the shared lattice point x. Known global bounds establish the existence of Radon-free sets through sizes 10 in $\mathbb{Z}^3$ and 20 in $\mathbb{Z}^4$, while $N = 11$ and $N = 21$ are the respective open targets (Onn, 1991; Bezdek & Blokhuis, 2003). Separate explicit witnesses certify those two feasible cardinalities immediately below the open targets.

Shared verification. Every reported output is judged by the same two-stage evaluator: it first checks convex position and then enumerates lattice points in $\mathrm{conv}(X)$ and tests whether each supports an integral Radon partition. A candidate succeeds only if it has the target cardinality, is in convex position, and has $n_{\mathrm{rp}} = 0$. The full evaluator and audit procedure appear in Appendix C.1.

For each lattice point $p \in (\mathrm{conv}(X) \cap \mathbb{Z}^d) \setminus X$, the evaluator tests whether there are weights $\lambda_i \geq 0$ and binary assignments $b_i \in \{0, 1\}$ satisfying

$$\textstyle\sum_i b_i\lambda_i x_i = p, \quad \sum_i (1-b_i)\lambda_i x_i = p, \quad \sum_i b_i\lambda_i = 1, \quad \sum_i \lambda_i = 2.$$

Here $b_i = 1$ assigns x_i to one side of the partition and $b_i = 0$ assigns it to the other. The first two equalities express the same point p as a convex combination of the two sides. The third equality normalizes the first combination; together with $\sum_i \lambda_i = 2$, it also makes the coefficients of the second combination sum to one. Thus both sides are nonempty, and feasibility holds exactly when p is an integral Radon point for some partition. PyNormaliz enumerates the candidate lattice points and Gurobi performs these split tests (Bruns & Ichim, 2010; Gurobi Optimization, LLC, 2026). Each feasible p is counted once, irrespective of how many partitions witness it, and their number is n_{rp}. The input points are omitted because the preceding convex-position check guarantees that none lies in the convex hull of the others.

Output semantics. A single leaderboard would conflate variable-length construction with fixed-target repair. For audited outputs $\mathcal{O}$ of a declared run, we therefore report

$$k_{\max} = \max\{|X| : X \in \mathcal{O},\ X \text{ convex},\ n_{\mathrm{rp}}(X) = 0\},$$
$$r_{\min}(N) = \min\{n_{\mathrm{rp}}(X) : X \in \mathcal{O},\ |X| = N,\ X \text{ convex}\}.$$

The second quantity is undefined if no convex target candidate is emitted. Method-native counts and layer profiles describe outputs but are not pooled into cross-method success rates. Search boxes, proposal domains, stopping rules, and budgets are stated per method; shared verification makes certificates and scores comparable, not search coverage or efficiency. Appendix B.1 gives the finite-box witnesses, and Appendix C records the per-method configurations.

3 METHODS

All five methods use the target cardinality and verifier from Section 2. For a method with an origin-anchored finite coordinate domain of side lengths $L = (L_1, \ldots, L_d)$, we write $\Lambda(L) = \prod_{j=1}^{d} \{0, \ldots, L_j - 1\}$. No single box is shared across methods: each protocol has the method-specific coordinate domain listed in Table 1, and the program-evolution method has no finite coordinate bound. Box membership is part of success only when the method itself enforces it.

Exhaustive extension and greedy local search are classical methods; PatternBoost, AlphaZero-Radon, and AlphaEvolve-style program evolution are learning-guided. By search objective, PatternBoost and AlphaZero-Radon are primarily cardinality constructors, while exhaustive extension, local search, and program evolution optimize complete N-point configurations. AlphaZero-Radon also scores the terminal configuration when it reaches N. The methods differ in their one-step proposals and reachable states, summarized in Table 2; these restrictions scope every negative result. In Table 2, "Starts from" denotes an externally supplied configuration; deterministic protocol anchors are specified in the method descriptions.

Table 2: High-level view of the five methods. Gray bands separate the two search objectives; implementation details appear in the subsections below.

Method	Starts from	Core idea
Cardinality construction: add points while keeping $n_{\mathrm{rp}} = 0$.		
PatternBoost	No points	Trains a Transformer ordering prior; keeps only convex, Radon-free prefixes.
AlphaZero-Radon	No points	Uses a set-attention policy and PUCT to add points; scores any convex configuration that reaches N points under its evaluator.
Fixed-target optimization: minimize n_{rp} over N-point configurations.		
Exhaustive extension	A Radon-free set of $N-1$ points	Tries every location for one final point in a finite box.
Greedy local search	A convex configuration of N points	Moves one point at a time whenever the number of integral Radon points decreases.
AlphaEvolve-style program evolution	16 Radon-free examples and a starter program	Uses an LLM to evolve code that proposes complete N-point configurations.

3.1 SHARED INTERFACE

Every method calls the shared convex-position and Radon-point evaluator described in Section 2. A proposal may emit a complete configuration or build/edit one sequentially. Reported outputs use the same success predicate, and every retained coordinate array is replayed with the reference evaluator. The exact implementation and the exhaustive driver's equivalent convex-position test are documented in Appendices C.1 and C.2.

3.2 EXHAUSTIVE ADD-ONE-POINT

For each of 50 fixed Radon-free $(N-1)$-point seeds S, this baseline tests every unoccupied position q in a declared finite box B_d and retains convex extensions. Its per-seed objective is

$$m_{B_d}(S) = \min_{\substack{q \in B_d \setminus S: \\ S \cup \{q\} \text{ in convex position}}} n_{\mathrm{rp}}(S \cup \{q\}) \,. \tag{1}$$

"Exhaustive" therefore means exhaustive over the final point after fixing S and B_d, not over all seeds or all N-point configurations. Exact boxes, coverage counts, and pseudocode are given in Table 1 and appendix C.2.

3.3 GREEDY LOCAL SEARCH

This deterministic-after-initialization repair baseline starts from a convex N-point extension of a stored Radon-free $(N-1)$-point configuration. A sweep visits points in stored order and evaluates all full-coordinate radius-one moves

$$x_i + \delta, \qquad \delta \in \{-1, 0, 1\}^d \setminus \{\mathbf{0}\},$$

that remain distinct, convex, and inside the method's box. It accepts the admissible move with the lowest strictly smaller n_{rp}, using a fixed direction order for ties, and stops at $n_{\mathrm{rp}} = 0$, after a non-improving sweep, or after 100 sweeps. Because it changes one point at a time and rejects plateaus and temporary increases, it cannot make coordinated moves or cross score barriers. Initialization and ordering details are in Appendix C.3.

3.4 PATTERNBOOST GENERATION

Our PatternBoost track is an iterative construction–training loop adapted from Charton et al. (2024). Configurations in $\Lambda(L)$ are encoded as mixed-radix point-token sequences. Generation 0 greedily grows empty states by accepting an addition only when the new prefix is convex and has $n_{\mathrm{rp}} = 0$. The largest terminal states are retained, symmetry-augmented, and used to train a decoder-only Transformer (Vaswani et al., 2017); sampled sequences seed the next generation and are completed by the same greedy legality test. The model is therefore trained as an ordering prior, while the verifier enforces every constraint. Here n_{rp} is a gate rather than a graded objective: an illegal addition is rejected, and construction stops when no tested point extends the prefix. Tokenization, selection, and training details are in Appendix C.4.

3.5 ALPHAZERO-STYLE CONSTRUCTION: ALPHAZERO-RADON

AlphaZero-Radon is a task-specific adaptation of the AlphaZero algorithmic template (Silver et al., 2018), not a reuse of its chess, shogi, or Go model. It retains policy–value-guided PUCT and an iterative search–training loop, but replaces the original game representation and residual convolutional network. Our single-agent construction task uses unordered point-set states, a permutation-invariant set-attention policy–value network (Lee et al., 2019), lattice-cell actions, and verifier-defined outcomes. Each trajectory starts at the fixed origin and adds the remaining points in $\Lambda(L)$. The network scores lattice locations, and Monte Carlo tree search plans with the shaped edge reward

$$r(s') = \begin{cases} 0 & \text{if } s' \text{ is not in convex position,} \\ \dfrac{C_{\mathrm{R}}}{N(1 + n_{\mathrm{rp}}(s'))} & \text{if } s' \text{ is convex and } n_{\mathrm{rp}}(s') > 0, \\ \dfrac{C_{\mathrm{S}}}{N} & \text{if } |s'| = N,\ s' \text{ is convex, and } n_{\mathrm{rp}}(s') = 0, \\ \dfrac{C_{\mathrm{step}}}{N} & \text{if } |s'| < N,\ s' \text{ is convex, and } n_{\mathrm{rp}}(s') = 0. \end{cases} \tag{2}$$

with $C_{\mathrm{step}} = 1$, $C_{\mathrm{R}} = 2$, and $C_{\mathrm{S}} = 25$. A non-convex or Radon-positive state terminates the simulated branch; convex Radon-positive states receive the displayed graded terminal reward at every cardinality. We report fixed-target quality only for convex terminals at the horizon.

To control branching, each expanded node retains the $K = 64$ unoccupied actions with the largest network probabilities. A child found terminal during search is marked and skipped by subsequent tree selections. Each decision uses 500 simulations, after which the trajectory generator samples from the root visit distribution at temperature one. Training, architecture, and the exact Plan-D search procedure are specified in Appendix C.6.

3.6 ALPHAEVOLVE-STYLE PROGRAM EVOLUTION

AlphaEvolve (Novikov et al., 2025) treats executable programs, rather than point sets, as the population under search. We instantiate this with OpenEvolve (Sharma, 2025): an LLM proposes exact search-and-replace edits to a Python block, a fixed evaluator executes the edited constructor, and an island-based MAP-Elites archive (Mouret & Clune, 2015) retains programs by both fitness and

behavioural features. Thus PatternBoost models a distribution over point sequences, whereas the AlphaEvolve-style method evolves the algorithm that constructs or repairs point sets.

The immutable wrapper gives each program sixteen fixed Radon-free $(N-1)$-point references and the verifier. One evaluation permits at most 1,000 `find_rps` calls and returns at most 32 integer $N \times d$ candidates. Only four constructor functions may evolve; the wrapper, resource accounting, geometry code, and external evaluator remain fixed. The initial program implements the interface but emits no candidate, so it contributes no hand-written search heuristic.

After the allowance closes, the evaluator validates shapes, finiteness, integrality and distinctness, then recomputes convexity and n_{rp} for every returned candidate. Program fitness is the maximum score of one candidate; statistics from different candidates are never mixed. A convex target with $k > 0$ Radon points scores $0.50 + 0.49/(1+k)$; a convex Radon-free target scores 1. The full fitness, evolution loop, and audit fields are given in Appendix C.5.

4 RESULTS

The five methods share one headline question: did the declared run produce a verifier-accepted configuration at the open target? Table 3 therefore reports target recovery first. When a run does not recover the target, it keeps two secondary outcomes in separate columns: $k_{\max}$, the largest fully Radon-free configuration emitted at any cardinality, and $r_{\min}(N)$, the smallest verifier score among emitted convex target-sized configurations. These quantities describe different output types and are not combined into a cross-method ranking.

Table 3: Verified outcomes at the two open targets. FOUND denotes a verifier-accepted target-sized output; a dash denotes an inapplicable output type.

Method	Found?	$k_{\max} \uparrow$	$r_{\min}(N) \downarrow$
$\mathbb{Z}^3$, *target* $N = 11$			
PatternBoost	No	10	–
AlphaZero-Radon	No	10	2
Exhaustive extension	No	–	2
Greedy local search	No	–	1
AlphaEvolve-style	No	–	1
$\mathbb{Z}^4$, *target* $N = 21$			
PatternBoost	No	20	–
AlphaZero-Radon	No	20	2
Exhaustive extension	No	–	2
Greedy local search	No	–	1
AlphaEvolve-style	No	–	1

RQ1: Did a protocol recover an open-target counterexample? No completed PatternBoost run, exhaustive add-one-point experiment, AlphaZero-Radon run, 1,000-start greedy-local-search run, or AlphaEvolve-style program-evolution run produces a verified open-target counterexample.

RQ2: What certified output did an unsuccessful run produce? PatternBoost reaches $k_{\max} = 10$ in $\mathbb{Z}^3$ and 20 in $\mathbb{Z}^4$. Across its complete 2,000-iteration runs, AlphaZero-Radon reaches $k_{\max} = 10$ in 3D and 20 in 4D; among its convex target terminals, the minimum score is 2 in both dimensions. Exhaustive add-one-point search also attains minimum score 2 in both dimensions after covering every extension position in each of its 50 fixed-seed boxes. Greedy local search attains score 1 in both dimensions (on 27 of 1,000 3D terminals and 25 of 1,000 4D terminals), but never reaches zero. The AlphaEvolve-style method likewise attains $r_{\min} = 1$ in both dimensions: its best programs return convex 11- and 21-point batches whose best individual candidates each have one Radon point. The two output types answer different questions: the former certifies a fully valid shorter configuration, whereas the latter measures a target-sized near miss. We therefore neither order them against one another nor pool method-native candidate counts into a success rate. Figure 2 shows within-run progress for the two cardinality constructors; Figure 3 gives the complementary

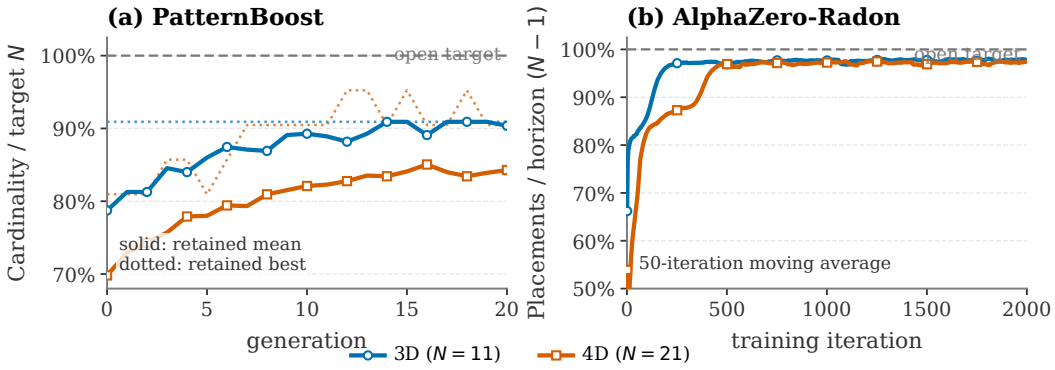

Figure 2: **Method-native construction progress.** (a) For PatternBoost, solid curves show the mean cardinality among the 50 retained configurations and dotted curves show the retained maximum at each generation. (b) For AlphaZero-Radon, curves show the mean number of post-origin placements per self-play trajectory, smoothed with a 50-iteration moving average. Values are normalized by the open target cardinality in (a) and by the post-origin horizon $N - 1$ in (b). The dashed line marks target completion. Horizontal axes are method-specific and do not represent matched compute; each curve summarizes one complete declared run.

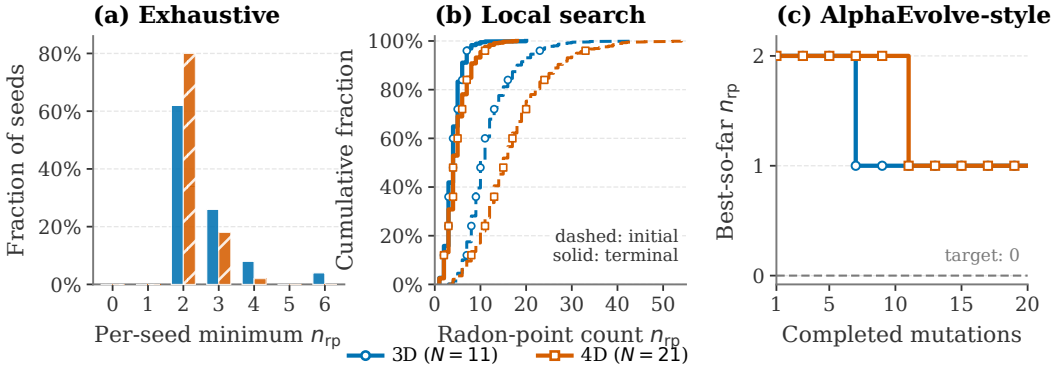

Figure 3: **Fixed-target search outcomes under the current declared protocols.** (a) For exhaustive extension, bars show the distribution of the minimum n_{rp} obtained after searching the full declared box for each of 50 fixed seeds. (b) For greedy local search, curves show empirical CDFs of the initial and terminal scores over 1,000 selected starts. (c) For AlphaEvolve-style program evolution, curves show the best-so-far score over 20 completed mutations; the first score-one program appears at mutation 7 in 3D and 11 in 4D. Colors identify the dimension. Seeds, starts, and mutations are protocol-native units and do not represent matched compute.

fixed-target outcomes for exhaustive extension, greedy local search, and AlphaEvolve-style program evolution. Both use method-native axes and do not compare computational efficiency. Appendix B.2 provides representative verified constructions.

The experiments are not compute- or verifier-query-matched. Method-native run lengths and budgets are reported for reproducibility, but Table 3 compares verified outputs rather than search efficiency. Appendix A records the completed runs and retained outputs.

5 Related Work

Onn formalized integral Radon partitions in the lattice setting, proved $r(\mathbb{Z}^2) = 6$, established general bounds, and studied recognition complexity (Onn, 1991). Bezdek and Blokhuis proved the upper bound $r(\mathbb{Z}^3) \leq 17$ (Bezdek & Blokhuis, 2003); integral Radon partitions are also integer 2-Tverberg

partitions (De Loera et al., 2017). These works motivate the open frontiers but do not provide a multi-method empirical study of constructive search.

Relevant ingredients in machine-assisted mathematical discovery include neural construction (Wagner, 2021), policy–value-guided tree search from AlphaZero (Silver et al., 2018), its task-specific adaptation to algorithm discovery in AlphaTensor (Fawzi et al., 2022), Transformer-guided search (Charton et al., 2024), and LLM program evolution (Romera-Paredes et al., 2024; Novikov et al., 2025). A recent comparative precedent studies classical, PatternBoost, and reinforcement-learning methods for the no-three-in-line problem (Ramanathan et al., 2025). RadonBench instead uses a lattice-convexity verifier and explicitly separates constructive from fixed-cardinality outputs. Complete computational resolutions of combinatorial existence problems require exhaustive search and checkable certificates (Heule et al., 2016; Heule, 2018; Brakensiek et al., 2022); our bounded searches do not provide such a non-existence certificate.

6 Discussion

Neither PatternBoost, exhaustive extension, nor AlphaZero-Radon reaches an open target; greedy local search likewise finds no target across 1,000 starts per dimension but reaches score-one near misses in both. Two explanations remain inseparable: the declared searches may cover too little of their method-specific domains, or a Radon-free configuration may not exist at that cardinality. These observations support only search-space diagnoses. A constructor's legal-prefix frontier reflects its proposal distribution and cutoff, not a global obstruction. After 20 mutations, the AlphaEvolve-style runs in both dimensions reach score-one near misses.

Reading the results through each search space. Exhaustive extension provides a finite-domain completeness statement only after fixing a legal $(N-1)$-point seed and a finite box for the last point. The completed experiment visits every position in each declared box, but its negative result concerns only those 50 fixed seeds per dimension, not configurations that require changing a seed or adding a point outside the box.

Greedy local search has the opposite search-space design. It can modify every coordinate of every point of an N-point configuration over successive sweeps, but each proposal moves only one point by Chebyshev radius one, remains inside the fixed box, and must immediately lower the integer-valued score. All 2,000 reported runs stall at positive score, including 27 of 1,000 in 3D and 25 of 1,000 in 4D at score one. A missing path may therefore require coordinated changes, a neutral or uphill intermediate state, or leaving the declared box.

PatternBoost and AlphaZero-Radon both build point sets sequentially. PatternBoost assigns no graded score to a failing prefix, whereas AlphaZero-Radon gives a larger terminal reward to a convex Radon-positive state with fewer integral Radon points. PatternBoost fits an ordering distribution from previously retained configurations and then uses the verifier to accept or skip sampled points. AlphaZero-Radon uses its policy to retain a top-K action set at each expanded node and spends MCTS simulations only within that restricted support. A frontier at $N-1$ therefore means that the learned proposal mechanism did not produce a verified continuation under the declared run; it is not evidence that no continuation exists elsewhere in the box.

AlphaEvolve-style program evolution has the broadest coordinate interface because an evolved program may emit arbitrary finite integer coordinates. Its practical search space is nevertheless the set of programs reached by its declared mutation lineage and the bounded batch of candidates each program returns. Its 3D and 4D lineages both attain one-Radon-point target configurations, but neither returns a Radon-free target. These distinctions are why we keep target recovery, valid-prefix cardinality, and target-sized score in separate table columns rather than treating the five methods as an efficiency leaderboard.

What defines a reproducible search-space claim. A finite-box exhaustive search must specify its fixed seeds and coordinate box; a repair method must specify its initialization, move radius, number of points changed together, and acceptance rule. A learned constructor additionally needs its action vocabulary, candidate truncation, and terminal handling, while program evolution needs its evaluator batch and number of mutations. Omitting any of these choices changes the reachable candidates. We expose them so future runs can vary one restriction at a time without changing the

success predicate. The same accounting explains why a grid position, a repair trajectory, an MCTS game, and a program-returned candidate are not interchangeable trials.

A complete negative result would require more than exhaustive evaluation inside one coordinate box. One would need a proved finite representative family, or an invariant coordinate bound, for lower-cardinality Radon-free configurations and a proved finite extension region for every representative. Affine unimodular transformations preserve Radon-freeness but need not preserve a box, so bounded-box exhaustion alone does not cover all equivalence classes. The appendices state the exact scope of every filled result.

Implications for search design. The framework suggests three protocol-level choices for future runs. First, repair methods should test coordinated moves or permit neutral and uphill transitions instead of enlarging the coordinate radius alone. Second, initialization and box constraints deserve the same care as the stopping rule because both restrict reachable basins. Third, evaluation should retain the objective each algorithm actually optimizes. Constructors should report the largest verified legal prefix, whereas fixed-target methods should report the best verified target score; neither replaces the common binary target-recovery outcome.

Limitations. The methods use different proposal domains, move sets, and run lengths; hardware and verifier calls are not matched. PatternBoost has one 20-generation run per dimension, and AlphaZero-Radon has one complete 2,000-iteration cold-start run per dimension. AlphaEvolve-style program evolution has one 20-mutation lineage per dimension rather than repeated independent trials. The greedy-local-search results are limited to 1,000 fixed starts per dimension, strict single-point descent, and dimension-specific fixed boxes. The exhaustive experiment fixes only 50 catalog seeds per dimension and cannot exclude extensions of other seeds or points outside its declared boxes. Consequently, Table 3 compares verified outputs, not sample or computational efficiency. Although the two constructive methods share the $6 \times 6 \times 6 \times 6$ box in 4D, their search distributions and budgets are not matched.

7 Conclusion

RadonBench organizes five classical and learning-guided searches around a common target-recovery criterion while preserving their different output semantics. PatternBoost reaches the known calibration cardinalities 10 and 20; across its complete 2,000-iteration runs, AlphaZero-Radon reaches the same cardinalities and target score 2 in both dimensions, while exhaustive add-one-point search obtains target score 2 but no target recovery after completely visiting the declared box for each of 50 fixed seeds in both dimensions. Full-coordinate greedy local search reaches score 1 but no target across 1,000 starts per dimension. AlphaEvolve-style program evolution also reaches score 1 in both dimensions after 20 mutations, without recovering either target. These results do not resolve the integer Radon numbers, but the success-first reporting scheme provides a reproducible framework for adding future searches without conflating solution quality with compute efficiency.

Reproducibility Statement

The appendix and supplementary material specify the verifier, method configurations, seeds, budgets, reported aggregates, representative outputs, and data used for the figures.

AI Use Statement

An LLM is an explicit experimental component of the AlphaEvolve-style method. Generative AI tools also assisted with implementation review and code editing, feedback on experimental methodology, artifact inspection and result interpretation, figure generation, and manuscript drafting and language editing. The authors reviewed all AI-assisted code, figures, analyses, and text, verified the reported claims and results, and take responsibility for the final content of the paper.

REFERENCES

Imre Bárány. *Combinatorial Convexity*, volume 77 of *University Lecture Series*. American Mathematical Society, Providence, RI, 2021.

Alexander Barvinok. *A Course in Convexity*, volume 54 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002.

Károly Bezdek and Aart Blokhuis. The radon number of the three-dimensional integer lattice. *Discrete & Computational Geometry*, 30(2):181–184, 2003.

Joshua Brakensiek, Marijn Heule, John Mackey, and David Narváez. The resolution of Keller's conjecture. *Journal of Automated Reasoning*, 66:277–300, 2022.

Winfried Bruns and Bogdan Ichim. Normaliz: algorithms for affine monoids and rational cones. *Journal of Algebra*, 324(5):1098–1113, 2010.

François Charton, Jordan S Ellenberg, Adam Zsolt Wagner, and Geordie Williamson. Patternboost: Constructions in mathematics with a little help from ai. *arXiv preprint arXiv:2411.00566*, 2024.

Jesús A. De Loera, Thomas A. Hogan, Frédéric Meunier, and Nabil H. Mustafa. Tverberg theorems over discrete sets of points. *arXiv preprint arXiv:1706.01178*, 2017.

Jesús A. De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil H. Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bulletin of the American Mathematical Society*, 56(3):415–511, 2019. doi: 10.1090/bull/1653.

Jürgen Eckhoff. Helly, Radon, and Carathéodory type theorems. In P. M. Gruber and J. M. Wills (eds.), *Handbook of Convex Geometry*, volume A, pp. 389–448. North-Holland, Amsterdam, 1993.

Geoffrey Exoo. A lower bound for r(5,5). *Journal of Graph Theory*, 13(1):97–98, 1989.

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

Peter M. Gruber. *Convex and Discrete Geometry*, volume 336 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, 2007.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL `https://www.gurobi.com`.

Marijn Heule. Schur number five. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing (SAT)*, pp. 228–245, 2016.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 2019. URL `https://proceedings.mlr.press/v97/lee19d.html`.

Brendan D. McKay and Stanisław P. Radziszowski. R(4,5) = 25. *Journal of Graph Theory*, 19(3): 309–322, 1995.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. URL `https://arxiv.org/abs/1504.04909`.

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

Shmuel Onn. On the geometry and computational complexity of Radon partitions in the integer lattice. *SIAM Journal on Discrete Mathematics*, 4(3):436–447, 1991. doi: 10.1137/0404039.

Stanisław P. Radziszowski. Small ramsey numbers. *Electronic Journal of Combinatorics*, 1(DS1), 1994. doi: 10.37236/21. Dynamic Survey.

Pranav Ramanathan, Thomas Prellberg, Matthew Lewis, Prathamesh Dinesh Joshi, Raj Abhijit Dandekar, Rajat Dandekar, and Sreedath Panat. Three methods, one problem: Classical and ai approaches to no-three-in-line. *arXiv preprint arXiv:2512.11469*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

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

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017. URL `https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need`.

Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*, 2021. URL `https://arxiv.org/abs/2104.14516`.

A Detailed Experimental Outcomes

All declared runs use the stopping rules and sample counts specified in the method appendix. The headline outcomes appear in Table 3; this section records only method-native evidence that is not shown there.

A.1 Constructive methods

PatternBoost. Both 20-generation loops completed. In the $20 \times 20 \times 5$ box, the largest verified Radon-free cardinality is 10; generation 20 retains 47 ten-point and three nine-point states. In the $6 \times 6 \times 6 \times 6$ box, the largest cardinality is 20, reached in generations 12, 13, 15, and 18; generation 20 retains eight nineteen-point, 19 eighteen-point, and 23 seventeen-point states. Neither run reaches its open target.

AlphaZero-Radon. Each 2,000-iteration run generates 200,000 trajectories. In 3D, 157,485 trajectories reach the 11-point horizon and 8,651 of those terminals are convex; their minimum score is $r_{\min}(11) = 2$. In 4D, 120,973 trajectories reach the 21-point horizon and 2,302 are convex; their minimum score is $r_{\min}(21) = 2$. The largest verified Radon-free prefixes have 10 and 20 points, respectively, and neither run reaches its open target.

Table 4: Coverage of the declared exhaustive add-one-point domains. Tested excludes positions occupied by the fixed seed; convex targets are the extensions scored by the verifier.

Instance	Seeds	Tested	Convex targets
$\mathbb{Z}^3,\ N = 11$	50	159,500	85,913
$\mathbb{Z}^4,\ N = 21$	50	499,000	173,355

A.2 Fixed-target methods

Exhaustive extension. For each dimension, the experiment fixes 50 Radon-free seeds and tests every unoccupied lattice position in the declared box. The 3D run tests 159,500 additions and scores 85,913 convex targets; the 4D run tests 499,000 additions and scores 173,355 convex targets. The minimum score is two in both dimensions. This is exhaustive only for the fixed seed–box pairs.

Greedy local search. Both dimensions use fixed 1,000-start evaluation sets. In 3D, 990 starts improve and 27 terminate at score one; the median score falls from 10 to 4. In 4D, 991 starts improve and 25 terminate at score one; the median falls from 15 to 4. Neither run reaches zero. The reported 4D aggregate is fixed to indices 0–999.

AlphaEvolve-style program evolution. Each dimension has one 20-mutation lineage. Every completed mutation returns 32 convex target-sized configurations under a shared allowance of 1,000 internal `find_rps` calls, for 640 externally scored candidates per dimension. Neither lineage reaches zero. The first score-one program appears at mutation 7 in 3D and mutation 11 in 4D.

B Additional Experiment Configurations

Software. Python 3.11.15, NumPy 2.4.6, PyNormaliz 2.23, Gurobi 13.0.2. The structural analyses use no GPU; all analysis runs on CPU on a shared machine, so wall-clock figures are indicative rather than compute-matched.

B.1 Finite-box calibration witnesses

The following ordered coordinate arrays are finite-box positive controls. The 3D witness has been translated so that its coordinate-wise minima are zero; the 4D witness is a verified 20-point prefix emitted by the reported AlphaZero-Radon run at iteration 190. The shared verifier confirms row distinctness, convex position, and $n_{\mathrm{rp}} = 0$ for both arrays.

Three-dimensional witness.

$$W_3 = \begin{bmatrix} 3 & 5 & 0 \\ 8 & 4 & 0 \\ 6 & 2 & 0 \\ 5 & 2 & 0 \\ 7 & 5 & 0 \\ 6 & 6 & 1 \\ 6 & 8 & 1 \\ 2 & 1 & 1 \\ 3 & 0 & 1 \\ 0 & 5 & 2 \end{bmatrix} \subset \Lambda(20, 20, 5).$$

This array is the $\mathbb{Z}^3$ positive-control witness used at $n = 10$.

Four-dimensional witness.

$$W_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 5 \\ 2 & 0 & 1 & 0 \\ 2 & 0 & 1 & 5 \\ 1 & 0 & 0 & 5 \\ 0 & 5 & 0 & 0 \\ 2 & 5 & 1 & 3 \\ 1 & 3 & 1 & 0 \\ 1 & 5 & 0 & 5 \\ 0 & 5 & 0 & 5 \\ 1 & 2 & 1 & 5 \\ 1 & 5 & 1 & 4 \\ 2 & 5 & 1 & 1 \\ 2 & 2 & 1 & 5 \\ 1 & 5 & 1 & 3 \\ 0 & 1 & 0 & 5 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 1 & 5 & 0 & 1 \end{bmatrix} \subset \Lambda(6,6,6,6).$$

This array is the $\mathbb{Z}^4$ positive-control witness used at $n = 20$.

Every subset of a convex-position configuration remains in convex position, and Radon-freeness is downward closed under taking subsets. Consequently, auditing the two displayed arrays also certifies feasibility at every smaller cardinality in the same boxes. The main experiments nevertheless use only $n = 10$ and $n = 20$ as positive controls, because smaller subsets add no independent feasibility question. The witnesses certify feasibility only; they do not imply that a particular solver can reach them from its initialization.

B.2 Three-dimensional diversity of learning-guided 10-point configurations

Figure 4 gives eight verified 10-point configurations in $\Lambda(20, 20, 5)$. PB-A–PB-D are retained outputs from PatternBoost generation 20; AZ-A–AZ-D are retained AlphaZero-Radon construction prefixes from iterations 1961, 1962, and 1999. The panel headings report the layer profiles as counts on $z = 0, \ldots, 4$. All eight profiles are distinct: $(0, 0, 5, 5, 0)$, $(0, 5, 5, 0, 0)$, $(0, 1, 4, 5, 0)$, $(0, 5, 4, 0, 1)$, $(4, 2, 0, 4, 0)$, $(3, 3, 0, 4, 0)$, $(4, 2, 0, 1, 3)$, and $(4, 2, 1, 1, 2)$. The examples were deliberately selected to expose different layerwise structures, not to estimate how frequently either method produces a profile. The reference verifier independently confirms convex position and $n_{\mathrm{rp}} = 0$ for every displayed configuration.

B.3 Result aggregation and audit

For constructive protocols, we replay retained serialized terminal states and take the largest cardinality passing convexity and $n_{\mathrm{rp}} = 0$; this produces $k_{\max}$ in Table 3. For fixed-target protocols, we discard repeated-row or non-convex arrays and take the minimum n_{rp} among the remaining N-point candidates; this produces $r_{\min}(N)$ in Table 3. Ties may be resolved by lexicographic order of the sorted integer rows when an exact representative is needed. Each reported result records its source run, iteration or generation, verifier output, and artifact path. AlphaZero-Radon aggregation includes all 2,000 completed iterations in each dimension.

C Implementation details

This appendix provides pseudocode for the reference verifier and every evaluated search method. The complete algorithmic specifications appear in Algorithms 1 to 7.

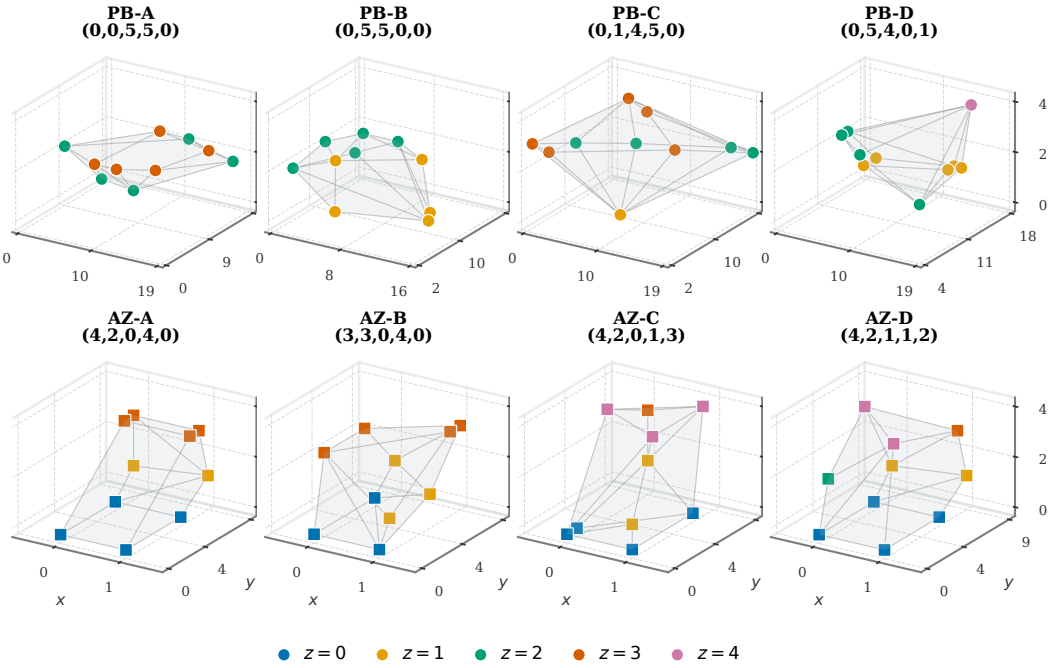

Figure 4: Representative learning-guided 10-point configurations in 3D. PB-A–PB-D (circles) are PatternBoost outputs; AZ-A–AZ-D (squares) are AlphaZero-Radon construction prefixes. Color encodes height z, and each heading gives the layer-count vector from $z = 0$ to $z = 4$. Gray edges and translucent faces show the convex-hull scaffold. To keep both compact AlphaZero-Radon configurations and box-spanning PatternBoost configurations legible, panels use their own displayed x–y limits, with the true limits shown by the ticks; the z axis is visually expanded. Every panel is a distinct Radon-free configuration. The examples illustrate structural diversity rather than empirical profile frequencies.

C.1 Reference verifier

The benchmark verifier consists of the functions `is_conv_position_all()` and `find_rps()`. For $X = \{x_1, \ldots, x_n\} \subset \mathbb{Z}^d$, the first routine performs one leave-one-out linear-program feasibility test per input point. For each i, it asks whether coefficients $\alpha_j \geq 0$ exist such that

$$\sum_{j \neq i} \alpha_j = 1, \qquad \sum_{j \neq i} \alpha_j x_j = x_i. \tag{3}$$

Feasibility means $x_i \in \mathrm{conv}(X \setminus \{x_i\})$ and the candidate is rejected; `is_conv_position_all()` returns `True` exactly when every leave-one-out model is infeasible.

The function `find_rps()` first passes the homogenized polytope to PyNormaliz (Bruns & Ichim, 2010) and reads `DegIElements`, which enumerates the lattice points in $\mathrm{conv}(X)$. Input vertices are removed. For every remaining lattice point p, the verifier asks whether there are $\lambda_i \geq 0$ and binary $b_i \in \{0, 1\}$ such that

$$\sum_i b_i \lambda_i x_i = p, \qquad \sum_i (1 - b_i) \lambda_i x_i = p,$$
$$\sum_i b_i \lambda_i = 1, \qquad \sum_i \lambda_i = 2. \tag{4}$$

The variable b_i assigns x_i to the first part when $b_i = 1$ and to the second part when $b_i = 0$; the corresponding products select the weights used by each convex combination. The first coefficient sum is constrained to one, and the second also sums to one because $\sum_i (1 - b_i)\lambda_i = \sum_i \lambda_i - \sum_i b_i \lambda_i = 2 - 1$. Thus a feasible solution places p in the convex hulls of two nonempty

disjoint parts. Conversely, any integral Radon partition and an integral Radon point give a feasible assignment. The implementation solves this model with Gurobi (Gurobi Optimization, LLC, 2026), sets `NonConvex=2` for the binary–continuous products, and tests each candidate p. It returns each feasible lattice point once, irrespective of the number of partitions that witness it; hence the benchmark score is $n_{\mathrm{rp}}(X) = |\texttt{find_rps}(X)|$. A candidate is a verified Radon-free configuration exactly when `is_conv_position_all()` returns `True` and `find_rps()` is empty.

Algorithm 1 Reference convexity and integral-Radon-point evaluator.

Require: finite set of distinct lattice points $X = \{x_1, \ldots, x_n\} \subset \mathbb{Z}^d$
Ensure: convex-position flag c and set R of integral Radon points

for $i = 1, \ldots, n$ **do**
  if $x_i \in \mathrm{conv}(X \setminus \{x_i\})$ according to Equation (3) **then**
    return (FALSE, $\varnothing$)
  end if
end for
$P \leftarrow (\mathrm{conv}(X) \cap \mathbb{Z}^d) \setminus X$ using PyNormaliz
$R \leftarrow \varnothing$
for $p \in P$ **do**
  if the feasibility system in Equation (4) accepts p **then**
    $R \leftarrow R \cup \{p\}$
  end if
end for
return (TRUE, R)

Following the canonical implementation, INFEASIBLE rejects a candidate Radon point and every other returned status counts it. All complete Radon-count evaluations use this same rule.

Cross-track accounting. The five tracks are not compute-matched. Method-native stopping rules are part of each protocol, but raw verifier-call totals are not used as a common budget or efficiency metric. Reported local-search aggregates are computed from retained JSON/NPZ artifacts for both the 3D and 4D instances; retained outputs likewise support both dimensional instances of the other tracks. A stochastic run is successful only if a serialized candidate passes the common verifier.

The available implementations use the software versions listed below and in Appendix C.3. Their evaluator implementations share the formulation and status rule above. Retained configurations and aggregate logs support the reported audits.

PatternBoost and AlphaZero-Radon environment. PatternBoost and AlphaZero-Radon use Python 3.11.15, PyTorch 2.12.0 with CUDA 13.0, NumPy 2.4.6, SciPy 1.17.1 (including Qhull), PyNormaliz 2.23, and Gurobi 13.0.2. Neural-network training and generation use one NVIDIA A100-SXM4 GPU with 80 GB memory; CPU processes execute the geometric oracle. We do not request PyTorch's bitwise-deterministic GPU kernels.

Local and exhaustive environment. These CPU-only searches use Python 3.12.11, NumPy 1.26.4, SciPy 1.16.2, PyNormaliz 2.19, and Gurobi 12.0.3; the local-search environment also contains PyTorch 2.8.0 and trimesh 4.8.3. They run on an Intel Xeon Gold 5320 node (two 26-core sockets, SMT2, 503 GiB RAM). Local tasks request 32 CPUs and 64 GB with at most three concurrent tasks; exhaustive tasks request 8 CPUs and 24 GB and run one at a time. Neither requests a GPU.

Program-evolution environment. OpenEvolve uses Python 3.12.11, OpenEvolve 0.2.27, NumPy 1.26.4, SciPy 1.16.2, PyNormaliz 2.19, and Gurobi 12.0.3 on the same CPU node. Each job requests four CPUs, 64 GB RAM, and no GPU. The method-specific subsections give the remaining seeds, stopping rules, provenance, and run settings.

Algorithm 2 Exhaustive add-one-point search within a fixed box

Require: verified Radon-free seed $S \subset \mathbb{Z}^d$ with $|S| = N - 1$; bounds $(\ell_j, u_j)_{j=1}^d$
Ensure: minimum score m, minimizers Q, score histogram H, coverage counts

$B \leftarrow \prod_{j=1}^d \{\ell_j, \dots, u_j\}$; $m \leftarrow +\infty$; $Q \leftarrow \varnothing$; $H \leftarrow \varnothing$; $c_{\text{convex}} \leftarrow 0$
for $q \in B$ in Cartesian-product order **do**
  if $q \in S$ **then**
    continue
  end if
  $X \leftarrow S \cup \{q\}$
  if X is not in convex position **then**
    continue
  end if
  $c_{\text{convex}} \leftarrow c_{\text{convex}} + 1$
  $r \leftarrow n_{\text{rp}}(X)$; $H[r] \leftarrow H[r] + 1$
  if $r < m$ **then**
    $m \leftarrow r$; $Q \leftarrow \{q\}$
  else if $r = m$ **then**
    $Q \leftarrow Q \cup \{q\}$
  end if
end for
return $(m, Q, H, |B| - |S|, c_{\text{convex}})$

C.2 Exhaustive add-one-point experiment configuration

Dimension-general procedure. For a fixed seed $S \subset \mathbb{Z}^d$ and integer box $B = \prod_{j=1}^d \{\ell_j, \dots, u_j\}$, the driver visits every $q \in B$ in deterministic Cartesian-product order. It skips $q \in S$, rejects extensions outside convex position, and evaluates $n_{\text{rp}}(S \cup \{q\})$ for every remaining extension. It does not stop when a zero or a new minimum is found. Therefore every unoccupied position in B is either rejected by the convex-position predicate or assigned a Radon-point count. This is the sense in which the search is exhaustive. The seed is never modified, and no claim is made about other seeds or positions outside B.

Frozen seeds and sampling. Before evaluation, the experiment fixes 50verified Radon-free seeds per dimension using NumPy seed 20260819. The 3D seeds lie in $\{-10, \dots, 9\}^2 \times \{0, \dots, 7\}$; the 4D seeds lie in $\{0, \dots, 19\}^2 \times \{0, \dots, 4\}^2$. Seed selection is independent of extension scores, and no sampling occurs during a task.

Boxes and coverage. The 3D box is $B_3 = \{-10, \dots, 9\} \times \{-10, \dots, 9\} \times \{0, \dots, 7\}$, with 3,200 positions. Each 10-point seed is contained in B_3, leaving 3,190 attempted additions per seed and 159,500 overall. The 4D box is $B_4 = \{0, \dots, 19\} \times \{0, \dots, 19\} \times \{0, \dots, 4\}^2$, with 10,000 positions. Each 20-point seed is contained in B_4, leaving 9,980 attempted additions per seed and 499,000 overall.

Convex-position and Radon tests. Before enumeration, the driver checks that the seed is row-distinct, in convex position, and has $n_{\text{rp}}(S) = 0$. For each extension X, it constructs the homogenized polytope with PyNormaliz and reads `ExtremeRays`. The extension passes exactly when the returned rays equal $\{(x, 1) : x \in X\}$, so every input point is a vertex. This exact rational extreme-ray test is mathematically equivalent to the reference leave-one-out convex-position predicate, although it is a different implementation. For every passing extension, `find_rps_nd()` uses PyNormaliz `Deg1Elements` and the same Gurobi partition-feasibility test in Appendix C.1. Gurobi uses one thread per model; up to eight lattice points are tested concurrently within a task.

The supplementary manifest records the fixed seed coordinates, declared boxes, and coverage counts needed to reproduce the reported aggregate.

Algorithm 3 Dimension-general ordered greedy local search.

Require: ordered convex configuration $X = (x_1, \ldots, x_N) \subset \mathbb{Z}^d$, sweep cap $T = 100$, ordered directions $\mathcal{D}_d$, score n_{rp}

for $t = 1, \ldots, T$ **do**
  $s_{\mathrm{start}} \leftarrow n_{\mathrm{rp}}(X)$
  for $i = 1, \ldots, N$ **do**
    $s^\star \leftarrow n_{\mathrm{rp}}(X)$; $X^\star \leftarrow X$
    for $\delta \in \mathcal{D}_d$ in lexicographic order **do**
      $Y \leftarrow X$ with $x_i \leftarrow x_i + \delta$
      if $Y \subseteq \Lambda(L)$, has distinct points, and is in convex position **then**
        $s \leftarrow n_{\mathrm{rp}}(Y)$
        if $s < s^\star$ **then**
          $(X^\star, s^\star) \leftarrow (Y, s)$
        end if
        if $s^\star = 0$ **then**
          break
        end if
      end if
    end for
    $X \leftarrow X^\star$
  end for
  if $n_{\mathrm{rp}}(X) = 0$ or $n_{\mathrm{rp}}(X) \geq s_{\mathrm{start}}$ **then**
    break
  end if
end for
return X

C.3 Greedy local-search experiment configuration

Dimension-general procedure. For an instance (d, N, L), the descent moves all coordinates. Its ordered neighbourhood is

$$\mathcal{D}_d = \{-1, 0, 1\}^d \setminus \{\mathbf{0}\}.$$

For a box $\Lambda(L)$, proposals outside the box are rejected. The evaluated 3D and 4D neighbourhoods contain 26 and 80 offsets, respectively, before boundary and geometry filtering. The method uses a strict-improvement rule, stored point order, lexicographic direction order, a 100-sweep cap, and early stopping at score zero or after a full non-improving sweep.

Initial configurations. In 3D, initialization adds one uniformly sampled point in $\{-10, \ldots, 9\}^2 \times \{0, \ldots, 4\}$ to a verified 10-point seed and accepts the first distinct, convex extension within 100 draws. From the eligible pool, 1,000starts are sampled with NumPy seed 20260819 and translated into the $20 \times 20 \times 5$ box. In 4D, one uniformly sampled in-box point is added to each verified 20-point seed, again requiring distinctness and convex position; the reported set is the fixed first 1,000starts generated with NumPy seed 20260815. Initialization does not filter or rank candidates by n_{rp}. Algorithm 3 gives the subsequent deterministic descent. The point order is the stored array order. Directions are the lexicographically ordered Cartesian product $\{-1, 0, 1\}^d$ with the zero vector removed. Only strict improvement is admissible, so the first direction attaining a given best score wins a tie and equal-score plateaus are never traversed. No anchoring translation occurs during descent. All accepted 3D configurations remain inside $\{0, \ldots, 19\}^2 \times \{0, \ldots, 4\}$, and all accepted 4D configurations remain inside $\{0, \ldots, 19\}^2 \times \{0, \ldots, 4\}^2$.

Convexity and score implementation. For every candidate, convex position is checked point by point. To test point x_i, a Gurobi feasibility LP asks whether nonnegative coefficients on $X \setminus \{x_i\}$ sum to one and reproduce x_i; feasibility rejects the candidate. For the objective, PyNormaliz enumerates the degree-one lattice points of the homogenized polytope and the implementation removes the input

vertices. For each remaining lattice point p, Gurobi solves the feasibility model

$$\sum_i b_i \lambda_i x_i = p, \qquad \sum_i (1 - b_i) \lambda_i x_i = p,$$
$$\sum_i b_i \lambda_i = 1, \qquad \sum_i \lambda_i = 2,$$

with $\lambda_i \geq 0$ continuous and $b_i \in \{0, 1\}$; the model uses `NonConvex=2`. The number of feasible p is the descent score; any solver status other than INFEASIBLE is treated as feasible. No Gurobi feasibility, integrality, or optimality parameters are overridden. Radon points are memoized within a configuration after its first score evaluation. The fixed selection manifests identify the 1,000 reported starts in each dimension, and the retained trajectories reproduce the aggregate outcomes in Appendix A. The CPU implementation uses Python 3.12.11, PyNormaliz 2.19, Gurobi 12.0.3, and SciPy 1.16.2.

Table 5: Greedy local-search hyperparameters shared across dimensions.

Parameter	Value
Initialization draw cap	100 per extension request
Evaluated starts / seeds	1,000 per dimension / 3D 20260819, 4D 20260815
Neighbourhood	$3^d - 1$ full-coordinate radius-1 moves (3D: 26; 4D: 80)
Point / direction order	stored index / lexicographic
Acceptance / plateaus	strict n_{rp} decrease / disallowed
Coordinate bounds during descent	3D: $[0, 19]^2 \times [0, 4]$; 4D: $[0, 19]^2 \times [0, 4]^2$
Maximum sweeps / early stop	100 / full non-improving sweep or $n_{\mathrm{rp}} = 0$

C.4 PATTERNBOOST EXPERIMENT CONFIGURATION

For an instance (d, N, L), PatternBoost uses variable-length add-point sequences in $\Lambda(L)$. Its vocabulary has $1 + \prod_{j=1}^{d} L_j$ entries: one shared START/STOP token and one token per lattice point. A sequence has the form $[0, \tau_L(p_1), \ldots, \tau_L(p_k), 0]$, where the points are sorted lexicographically before supervised training; losses after the first stop token are masked. The reference incremental legality test accepts a prefix only when it is in convex position and has no integral Radon point. This is stricter than merely filtering complete model outputs: every training configuration and every saturated output is legal at each accepted prefix. All dimension-independent architectural, optimization, selection, generation, and saturation hyperparameters are shared across dimensions.

The completed three-dimensional run instantiates $L = (20, 20, 5)$ and $N = 11$, giving a 2,001-token vocabulary. The completed four-dimensional run uses $L = (6, 6, 6, 6)$ and $N = 21$, giving a 1,297-token vocabulary.

Table 6: PatternBoost hyperparameters.

Parameter	Value
Transformer architecture	
Causal blocks / attention heads	4 / 4
Embedding / FFN width	128 / 512
Activation / normalization	GELU / pre-LN
Dropout	0
Training	
Training rounds	20
Retained configurations / round	50
Optimizer	AdamW
Learning rate / schedule	5×10^{-4} / fixed
Weight decay	0.01
Adam β_1, β_2 / ϵ	$(0.9, 0.99)$ / 10^{-8}
Maximum updates / round	500
Minibatch size	32
Validation-orbit fraction	0.1
Proposal and saturation	
Generated states / round	250
Selection batch / keep fraction	250 / 0.2
Saturation workers	16
Sampling temperature / top-k	1 / none

Algorithm 4 PatternBoost with verifier-gated incremental saturation.

Require: box L, maximum size N, generations G, state-batch size B, keep fraction q, target retained-set size T, reference predicate LEGAL(S)

form T/q independent empty seeds; saturate each seed with LEGAL
retain the top qB terminal states within each batch to obtain $\mathcal{D}_0$
for $i = 0, \ldots, G-1$ **do**
    split $\mathcal{D}_i$ by complete symmetry orbit into train/validation sets
    augment both splits by box-preserving reflections and axis permutations
    train a next-token Transformer; retain the lowest-validation-loss checkpoint
    sample T/q unconstrained token sequences from START
    decode each sequence in order, skipping a token unless it denotes an unused lattice point whose addition satisfies LEGAL
    for each decoded seed state S **do**
        while $|S| < N$ and untested lattice points remain **do**
            traverse the remaining points in a fresh random order
            append the first p with LEGAL($(S \cup \{p\})$); permanently remove every tested point, including p, from the remaining pool
            stop if no tested point is legal
        end while
    end for
    in each batch of B terminal states, retain the top qB states by cardinality
    $\mathcal{D}_{i+1} \leftarrow$ retained states; save their unaugmented and augmented forms
end for

Teacher forcing pads every sequence to context length $N+1$ (12 in 3D and 22 in 4D). For an ordered point-token sequence $(\tau_1, \ldots, \tau_k)$, the input is $[0, \tau_1, \ldots, \tau_k]$ and the target is $[\tau_1, \ldots, \tau_k, 0]$; target positions after the first stop are set to -1 and ignored by cross-entropy. Minibatches are sampled uniformly with replacement.

Let m_ℓ be the multiplicity of side length ℓ in L. The augmentation group has

$$|\mathcal{H}(L)| = 2^d \prod_\ell m_\ell!$$

box-preserving transformations: independent reflections and permutations among axes with equal side lengths. All transforms are emitted without deduplicating coincident arrays. For $L = (20, 20, 5)$ this gives 16 transforms; for $L = (6, 6, 6, 6)$ it gives $2^4 4! = 384$. With 50 retained configurations, the nominal $45/5$ orbit split yields $720/80$ augmented train/validation sequences in 3D and $17{,}280/1{,}920$ in 4D when all orbits are distinct. Complete symmetry orbits are assigned wholly to one split before augmentation. Validation evaluates at most ten batches of 100. Training runs for 500 updates and evaluates every 25 updates; the minimum validation-loss checkpoint among steps $25, 50, \ldots, 500$ is used both for generation and to initialize the next round. Exact split sizes and selected steps are recorded per generation.

The Transformer uses learned token and absolute-position embeddings, untied output weights, causal self-attention, and PyTorch's default parameter initialization. Its dropout rate is zero. At the start of a new round, model weights come from the preceding round's selected checkpoint but AdamW is reinitialized; optimizer state is restored only when resuming an interruption within the same round. AdamW uses the fixed learning rate in Table 6.

Generation always draws N tokens after the initial START, without constraining the vocabulary, in minibatches of 10. Decoding stops at the first sampled STOP; before that stop, duplicate or illegal point tokens are removed from the remaining-point pool and skipped while later tokens are still attempted. Saturation repeatedly draws a fresh permutation of the remaining pool, accepts the first legal point, and permanently removes all points tested in that pass. The 250 terminal states form one selection batch; the largest 20%, namely 50 states, survive. Stable input order breaks cardinality ties. Generation and saturation use 16 CPU workers. A master seed of 20260809 initializes Python, NumPy, and PyTorch randomness; per-state seeds are assigned before parallel dispatch.

C.5 ALPHAEVOLVE-STYLE PROGRAM-EVOLUTION EXPERIMENT CONFIGURATION

Dimension-general task interface. For every instance (d, N), the immutable wrapper receives the same typed inputs: sixteen cataloged $(N-1) \times d$ arrays encoding Radon-free configurations, the dimension-independent geometry routines, target size N, output cap, and remaining oracle allowance. Returned candidates must have shape $N \times d$. The mutable four-function interface, validation order, fitness, oracle allowance, output cap, and MAP-Elites mechanism are shared across dimensions. The dimension, target, reference catalog, feature-bin count for `best_size`, and number of evolution mutations are recorded separately below. No lattice side lengths are passed to the evolved program or enforced by the evaluator.

Dimension-specific inputs. The 3D run samples eight references from each of two catalogs of verified 10-point configurations; the 4D run samples sixteen references from a catalog of verified 20-point configurations. Both use selection seed 2026, run for 20mutations, and accept arbitrary finite integer output coordinates. The completed runs take 4.88 and 47.58 hours in 3D and 4D, respectively.

Each call to `generate_candidates` installs process-local wrappers around the geometry functions. Calls to `find_rps` while the candidate program is active increment a counter and are limited jointly to 1,000; after exhaustion, a further call returns a nonempty sentinel rather than permitting a false Radon-free result. Convexity calls are uncharged. The internal allowance closes before the external evaluator begins, so rechecking the returned batch neither consumes nor enlarges the program's search budget. Every program receives a newly initialized NumPy `default_rng(42)` and may return at most 32 arrays.

Candidate validation and fitness. The evaluator imports the generated module and invokes `generate_candidates(N,32)`; the declared three-dimensional instance instantiates this call with $N = 11$. Exceptions receive score 0 and an error flag. Each returned object is converted to a NumPy array and rejected unless it is finite, two-dimensional with d columns, integer-valued, and row-distinct. More than 32 objects are truncated; arrays with a row count other than N receive candidate score 0. For a target-sized array $X = (x_1, \ldots, x_N)$, let $P = (x_1, \ldots, x_{N-1})$ denote its

ordered prefix. The exact candidate score is

$$s(X) = \begin{cases} 1, & X \text{ is convex and } n_{\mathrm{rp}}(X) = 0, \\ 0.50 + 0.49/(1 + n_{\mathrm{rp}}(X)), & X \text{ is convex and } n_{\mathrm{rp}}(X) > 0, \\ 0.40 + 0.09((N-1)/N), & X \text{ is not convex, but } P \text{ is convex and } n_{\mathrm{rp}}(P) = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The program fitness `combined_score` is $\max_X s(X)$ over the returned batch. The evaluator records the best candidate and summary batch statistics; no metric combines properties from different candidates.

Algorithm 5 Dimension-general AlphaEvolve-style program-evolution search.

Require: instance (d, N), initial program p_0, fixed evaluator E, islands $\mathcal{I}$, number of mutations T, target fitness 1

evaluate p_0 with E; insert its source, metrics, and features into the database
for $t = 1, \ldots, T$ **do**
    choose the active island and sample a parent from its population/archive
    assemble system and user messages from the fixed task prompt, parent source and metrics, execution artifacts, lineage, and selected top/diverse programs
    request an exact `SEARCH/REPLACE` diff from `gpt-5.6-sol`
    apply valid edits only inside the mutable `EVOLVE-BLOCK`; reject a full rewrite
    execute the child with a fresh RNG seed 42 and a shared allowance of 1,000 internal `find_rps` calls
    E independently validates and scores at most 32 returned candidates
    store source, parent, prompt, response, metrics, features, and candidate artifact
    update the island and global archive; migrate after every ten island generations
    checkpoint; **stop** if `combined_score`$= 1$
end for

Table 7: AlphaEvolve-style program-evolution settings.

Parameter	Value
Output cap	32 candidates
Reference seeds	16 cataloged $(N-1)$-point records
Reference-selection seed / program RNG	2026 / NumPy PCG64 seed 42
Internal oracle allowance	1,000 `find_rps` calls per program
Convexity calls / evaluator replay	uncharged / outside internal allowance
Evolution mutations / target score	20 per dimension / 1
Population / archive / islands	24 / 24 / 3
Migration interval	10 island generations
MAP-Elites features / bins	`best_size`: 3D 12, 4D 22; `convex_fraction`: 10
Evolution mode	exact diffs; full rewrites disabled
LLM / weight	`gpt-5.6-sol` / 1.0
Reasoning effort / output limit	medium / 8,000 tokens

Outcomes. Each AlphaEvolve-style program-evolution experiment completes 20 mutations.

Every child in both runs returns 32 convex target configurations, uses exactly 1,000 internal oracle calls, and has evaluator error flag zero. Each lineage therefore returns $20 \times 32 = 640$ candidates. Neither returns a Radon-free target. The best 3D program (first seen at mutation 7) and best 4D program (first seen at mutation 11) each attain one Radon point and fitness 0.745.

C.6 AlphaZero-Radon experiment configuration

The reported runs use the Plan-D search and training implementation. The 3D instance has $L = (20, 20, 5)$, $N = 11$, and 2,000 actions. The 4D instance has $L = (6, 6, 6, 6)$, $N = 21$, and 1,296

actions. Both complete 2,000 iterations, comprising 200,000 generated trajectories per dimension; all completed iterations enter the reported outcomes. The implementation calls this data-generation phase "self-play," but the adapted task is single-agent and has no opponent.

States, actions, and terminal outcomes. A state is the unordered set of placed lattice points, serialized internally by mixed-radix tokens

$$a(x) = \sum_{j=1}^{d} x_j \prod_{k=1}^{j-1} L_k,$$

so the first axis varies fastest. Every game starts from the fixed origin $(0, \dots, 0)$ and the agent selects the remaining $N-1$ points. The origin is an explicit finite-box anchor, not a without-loss-of-generality reduction; it contributes neither a policy target nor an edge reward. At an expanded node, occupied actions are zeroed before candidate ranking. Applying an action terminates the branch when the new set is non-convex, has positive n_{rp}, or reaches N. Rewards follow Equation (2) and are attached to the edge entering the successor. The value head estimates the undiscounted future reward-to-go.

Plan-D top-K PUCT. Let $P_\theta(\cdot \mid s)$ be the full-vocabulary softmax policy. Every newly expanded node zeros occupied actions, retains the $K = 64$ largest remaining probabilities, and renormalizes them. Writing the retained support as $C(s)$, the tree prior is

$$\widehat{P}(a \mid s) = \frac{P_\theta(a \mid s)\mathbf{1}\{a \in C(s)\}}{\sum_{b \in C(s)} P_\theta(b \mid s)}.$$

At the root only, self-play replaces this prior by $0.75\widehat{P} + 0.25\eta$, where $\eta \sim \mathrm{Dir}(0.1\mathbf{1})$. There is no eager root probe: a transition is checked only when tree selection reaches it. Once a child is found terminal, Plan D marks that edge and excludes it from later tree selections. Terminal means any non-convex or Radon-positive set as well as a target success, so a scored terminal receives one backup before it is masked. Search does not stop early upon discovering a success.

Algorithm 6 AlphaZero-Radon single-agent trajectory-generation and training loop.

Require: lattice sides L, target N, iterations I, games G, reads M, candidate count K, c_{puct}, Dirichlet parameter α

initialize policy–value parameters θ, Adam state, and FIFO buffer $\mathcal{B}$
for $i = 1, \dots, I$ **do**
  $\mathcal{D}_i \leftarrow \emptyset$
  for $g = 1, \dots, G$ **do**
    initialize $s_0 = \{(0, \dots, 0)\}$ and empty state, policy, and reward lists
    while s_t is not terminal **do**
      $\pi_t \leftarrow \mathrm{PUCTSEARCH}(s_t, \theta, M, K, c_{\mathrm{puct}}, \alpha)$
      store (s_t, π_t); sample $a_t \sim \pi_t$; apply a_t
      append the realized edge reward and continue unless terminal
    end while
    compute $G_t = \sum_{j=t}^{T_{\mathrm{end}}-1} r_j$ backwards
    add all $(s_t, \pi_t, G_t, n_{\mathrm{rp}}^{\mathrm{final}})$ to $\mathcal{D}_i$
  end for
  append $\mathcal{D}_i$ to $\mathcal{B}$; retain its latest 40 iterations
  for $h = 1, \dots, 10$ **do**
    sample a set-length bucket proportional to its buffered count
    draw at most 128 examples from that bucket without replacement
    update θ with Adam on Equation (6)
  end for
  every 10 iterations, save the model weights
end for

Table 8: AlphaZero-Radon hyperparameters shared across dimensions.

Parameter	Value
Search and trajectory generation	
Self-play worker processes	32
Training iterations	2,000
Self-play games / iteration	100
MCTS reads / move	500
PUCT constant c_{puct}	1.0
Candidate-set size K	64
Dirichlet α / noise fraction	0.1 / 0.25
Self-play temperature T	1
Network architecture	
Model width	256
Nyquist-safe Fourier bands / axis	3 in both dimensions
SAB layers / attention heads	4 / 4
Attention activation / normalization	ReLU / LayerNorm
Policy/value hidden width	256 / 256
Head activation / dropout	GELU / 0
Optimization	
Optimizer	Adam
Learning rate / schedule	10^{-4} / fixed
Weight decay	10^{-4}
Adam β_1, β_2 / ϵ	$(0.9, 0.999)$ / 10^{-8}
Maximum minibatch / updates per iteration	128 / 10
Replay-window length	40 iterations
Value-loss weight λ_V	20
Discount γ	1
Reward shaping	
Step / Radon / success constants	1 / 2 / 25
Convexity-break constant	0

Algorithm 7 Plan-D fixed-candidate PUCT search used by the reported runs.

Require: state s, network θ, reads M, candidate count K, c_{puct}, Dirichlet parameter α

function PUCTSEARCH($s, \theta, M, K, c_{\mathrm{puct}}, \alpha$)
  create a fresh root; retain and renormalize its occupancy-legal policy top-K
  mix Dirichlet noise over those root candidates
  for $m = 1, \ldots, M$ **do**
    descend over unmarked retained edges by maximizing $Q(u,a) + c_{\mathrm{puct}} U(u,a)$
    evaluate the selected successor with STEPOUTCOME
    if terminal **then**
      mark its parent edge terminal; back up its edge reward with leaf value 0
    else
      retain the policy top-K candidate set at the leaf; set leaf value to V_θ
      back up intervening edge rewards plus the leaf value
    end if
  end for
  return the normalized full-vocabulary root visit counts π
end function

PUCT and training targets. PUCT uses

$$Q(s,a) = \frac{W(s,a)}{1 + N(s,a)}, \qquad U(s,a) = \widehat{P}(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)}.$$

The root visit count starts at zero. Each simulation accumulates normalized per-edge rewards between the leaf and every ancestor and adds V_θ only at a nonterminal leaf. After 500 simulations, the

root visit counts are normalized over the full vocabulary and the played action is sampled from them with $T = 1$.

For a minibatch of size B, define the full-vocabulary policy by

$$p_\theta(a \mid s_b) = \frac{\exp \ell_\theta(s_b, a)}{\sum_{c \in \Lambda(L)} \exp \ell_\theta(s_b, c)}.$$

The loss is

$$\mathcal{L} = -\frac{1}{B} \sum_{b=1}^{B} \sum_{a \in \Lambda(L)} \pi_b(a) \log p_\theta(a \mid s_b) + \lambda_V \frac{1}{B} \sum_{b=1}^{B} \big(V_\theta(s_b) - G_b\big)^2. \tag{6}$$

Here $\lambda_V = 20$, π_b is the normalized MCTS visit vector, and G_b is the undiscounted reward-to-go. Although π_b is sparse because search expands top-K actions, the softmax denominator is the full action vocabulary.

Network. Each coordinate is normalized as $u_j = x_j/(L_j - 1)$ and mapped to $[\sin(2^k \pi u_j), \cos(2^k \pi u_j)]_{k=0}^{B_d - 1}$, where the Nyquist-safe $B_d = 3$ in both runs, followed by a learned linear projection to width 256. Four layer-normalized set-attention blocks with four heads encode the placed-point set. Each multihead-attention block uses scaled dot-product attention, a query residual, a ReLU output projection with a second residual, and LayerNorm after each residual. Every candidate cell uses the same coordinate encoder to form a cross-attention query over the set latents; a $256 \to 256 \to 1$ GELU MLP produces its policy logit. The value head mean-pools the set latents and applies an independent $256 \to 256 \to 1$ GELU MLP, whose scalar output is used directly. All linear layers use PyTorch defaults, and the dropout rate is zero.

Replay and optimization. One randomly initialized network and one Adam optimizer persist across all iterations. At each iteration, every self-play game receives the latest evaluation-mode weights, and every move constructs a fresh tree. Deterministic verifier outcomes may be memoized within a trajectory; this does not alter transitions, terminal labels, rewards, or the search budget.

For each decision after the origin, self-play stores the pre-move token set, the normalized visit policy, the final Radon count, and the reward-to-go.

The learner appends the current iteration before training. For each of ten updates, it selects a set-length bucket with probability proportional to its buffered sample count and draws $\min(128, |\text{bucket}|)$ examples uniformly without replacement. Thus each iteration makes at most 1,280 replay draws, independent of the current FIFO population; samples may reappear in later updates. Adam uses the fixed learning rate 10^{-4}, PyTorch defaults for $(\beta_1, \beta_2, \epsilon)$, and coupled ℓ_2 weight decay 10^{-4}.

Both jobs run uninterrupted from a clean initialization to iteration 2,000. The global initialization seed is 42; per-game and replay-sampling seeds are deterministic functions of the iteration and game indices. End-to-end wall times are approximately 85.9 hours in 3D and 561.4 hours in 4D.

Aggregation over the complete runs. In 3D, 157,485 of 200,000 generated trajectories place an eleventh point. Of those terminals, 8,651 are convex and receive a complete Radon count; the minimum is 2. In 4D, 120,973 games place a twenty-first point, 2,302 of those terminals are convex, and the minimum Radon count is 2; a trajectory with terminal score two first appears at iteration 702. No target terminal has score zero. The remaining target-cardinality terminals are non-convex and are not included in the "target candidates" column. The largest fully verified prefixes have sizes 10 and 20, respectively. Figure 5 reports the corresponding optimization and trajectory-length diagnostics over the same complete runs.

D LLM USAGE

An LLM is an explicit component of the AlphaEvolve-style experiment: OpenEvolve prompts the model identified in Table 7 to propose edits to the mutable program block. The experiment records

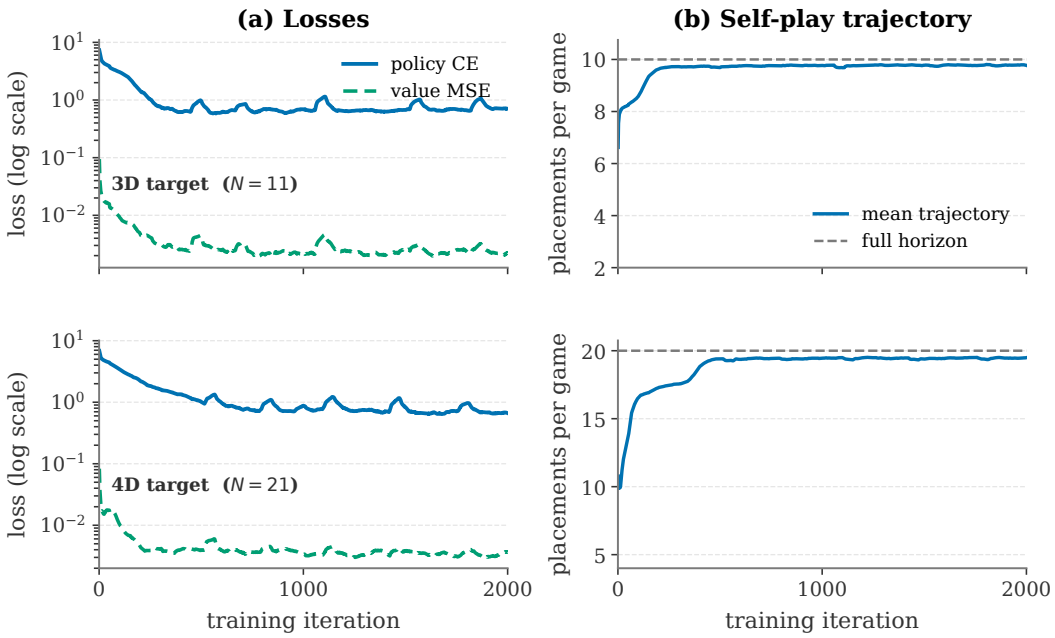

Figure 5: Training diagnostics for the complete 2,000-iteration AlphaZero-Radon runs. The top and bottom rows show the 3D ($N = 11$) and 4D ($N = 21$) instances, respectively. Curves are 50-iteration moving averages. The trajectory panels show only the mean trajectory length; the dashed line is the full horizon of 10 or 20 policy-selected placements after the fixed origin.

the realized prompts, responses, edited programs, parent links, and evaluator outputs. The LLM does not alter the verifier, reference data, resource accounting, or reported metrics.

General-purpose LLM tools also assisted with manuscript organization, language editing, code and artifact inspection, and consistency checks. The authors selected the scientific claims and experiments, verified all numerical values against stored artifacts, checked all citations against external sources, and remain responsible for the complete contents of the paper. No unverified LLM-generated numerical result is reported.