Four Methods, One Problem: The No-Three-In-Line Problem

Pranav Ramanathan [1] **Thomas Prellberg** [1] **Prathamesh Dinesh Joshi** [2] **Raj Abhijit Dandekar** [2]
Rajat Dandekar [2] **Sreedath Panat** [2]

Abstract

The No-Three-In-Line problem asks for the maximum number of points that can be placed on an $n \times n$ grid with no three collinear, a famous problem in combinatorial geometry. We compare four approaches: (i) integer linear programming (ILP) using Gurobi, (ii) transformer-based pattern learning via PatternBoost, (iii) reinforcement learning using proximal policy optimization (PPO), and (iv) a Scientific Machine Learning (SciML) method that recasts the problem as continuous energy minimisation solved by gradient flow on Apple Metal. ILP achieves provably optimal solutions up to 19×19 grids. PatternBoost matches optimal performance up to 14×14 grids and finds 29 points on 15×15 grids (optimal is 30). PPO achieves perfect solutions on 10×10 grids but fails at 11×11 due to constraint violations. SciML finds optimal $2n$ configurations up to $n = 18$ in several cases on Apple M2 (Metal) hardware without symmetry pruning, but reliability is non-monotonic: $n = 17$ achieves 0/10 in a 10-run sweep (success reproduced only for one seed), while $n = 18$ succeeds 9/10 runs. Continuous-relaxation methods extend the reach of learning-based solvers well beyond the discrete-search ceilings of PatternBoost and PPO, suggesting a productive new direction for combinatorial geometry.

1. Introduction

We consider the No-Three-In-Line problem (Guy & Kelly, 1968): What is the maximum number of points that can be placed on an $n \times n$ grid such that no three are collinear? Figure 1 illustrates an optimal solution for a 10×10 grid, achieving the maximum of 20 points with no three collinear.

[1]School of Mathematical Sciences, Queen Mary University of London [2]Vizuara AI Labs. Correspondence to: Pranav Ramanathan <p.ramanathan@se23.qmul.ac.uk>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

To our knowledge, this is the first systematic comparison of classical and AI-driven search paradigms for the No-Three-In-Line problem.

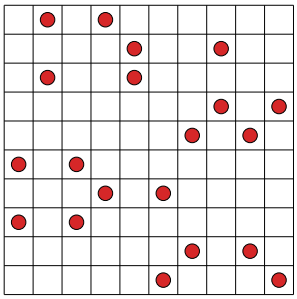

Figure 1. Optimal solution for the No-Three-In-Line problem on a 10×10 grid, achieving the maximum of 20 points.

A trivial upper bound of $2n$ points follows from the pigeonhole principle, as no row or column can contain more than two points. The best proven lower bound, due to Hall, Jackson, Sudbery, and Wild, places approximately $1.5n$ points using modular hyperbola constructions (1975). However, Guy and Kelly conjectured that for sufficiently large n, the maximum asymptotically approaches only cn points, where $c \approx 1.814$ (1968). This leaves a significant gap between the best-proven result and the conjectured limit, underscoring the problem's persistent difficulty. The problem has also been studied in three dimensions (Pór & Wood, 2007) and in an extensible formulation (Nagy et al., 2023).

Traditional computational approaches, such as recursive backtracking algorithms and integer linear programming (ILP), frame the No-Three-In-Line problem as a constraint satisfaction task by encoding collinearity constraints algebraically and using branch-and-bound optimization (Nemhauser & Wolsey, 1988; Eppstein, 2018). These methods grow exponentially in n. Recent CP-SAT and GPU enumeration push exact solutions to $n \geq 60$ (Flammenkamp, 2026), but both depend critically on symmetry pruning: with that restriction removed, unrestricted exact completeness in the public record extends to at most $n = 19$–20 (Flammenkamp, 2026). This gap highlights how strongly current exact methods still depend on symmetry reduction.

Tsuchiya and Takefuji applied classical neural networks

to the No-Three-In-Line problem in 1995 (1995); modern ML approaches have not revisited the problem until recently. Recent ML approaches to combinatorial optimization include reinforcement learning, graph neural networks, and transformer-based pattern search (Mazyavkina et al., 2021; Cappart et al., 2023; Vaswani et al., 2017). A recent breakthrough came with PatternBoost (Charton et al., 2024), which discovered a counterexample to Graham and Harary's conjecture on hypercube subgraphs (Graham & Harary, 1992). While these applications demonstrate PatternBoost's effectiveness on geometric point placement problems with distance-based constraints, its application to collinearity-based constraints remains unexplored.

We compare four approaches: integer linear programming using the Gurobi solver, transformer-based pattern learning via the PatternBoost framework, reinforcement learning using Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Scientific Machine Learning (SciML) (Rackauckas & Nie, 2017; Karniadakis et al., 2021).

Our contributions are:

- **A comparative study.** To our knowledge, the first systematic comparison of four search paradigms: ILP, learned pattern search, reinforcement learning, and SciML continuous relaxation, studying where each paradigm succeeds and how it fails.
- **A SciML method.** A novel SciML method that recasts the No-Three-In-Line problem as continuous energy minimisation and solves it via gradient flow on Apple M2 (Metal) GPU hardware without symmetry pruning.
- **PatternBoost on collinearity.** To our knowledge, the first application of PatternBoost (Charton et al., 2024) to the No-Three-In-Line problem.
- **PPO with action masking.** To our knowledge, the first application of a PPO reinforcement-learning agent (Schulman et al., 2017) with action masking to the No-Three-In-Line problem.

2. Methods

2.1. Problem Formulation

The No-Three-In-Line problem is defined on an $n \times n$ discrete grid, where each cell (i, j) with $0 \leq i, j < n$ represents a potential point placement. We represent a configuration as a binary matrix $\mathbf{X} \in \{0, 1\}^{n \times n}$, where $x_{i,j} = 1$ indicates a point is placed at position (i, j), and $x_{i,j} = 0$ indicates an empty cell.

Collinearity Constraint. Three distinct points (x_1, y_1), (x_2, y_2), and (x_3, y_3) are collinear if and only if they satisfy:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0 \tag{1}$$

This determinant evaluates to zero when the three points lie on a common line (horizontal, vertical, or diagonal). The constraint applies to *all* possible straight lines passing through the grid, including lines with arbitrary rational slopes.

Optimization Objective and Constraints. With decision variables $x_{i,j} \in \{0, 1\}$ indicating a point at (i, j), the ILP formulation is:

$$\max_{x_{i,j} \in \{0,1\}} \quad \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{i,j} \tag{2}$$

$$\text{s.t.} \quad \sum_{(i,j) \in \ell} x_{i,j} \leq 2 \quad \forall \text{ grid lines } \ell \text{ with } |\ell| \geq 3. \tag{3}$$

Equivalently, the line constraint can be written for every triple of distinct grid points whose determinant above is zero:

$$x_{i_1,j_1} + x_{i_2,j_2} + x_{i_3,j_3} \leq 2 \quad \forall \text{ distinct } (i_k, j_k) \quad \text{such that} \begin{vmatrix} i_1 & j_1 & 1 \\ i_2 & j_2 & 1 \\ i_3 & j_3 & 1 \end{vmatrix} = 0. \tag{4}$$

The two formulations are mathematically equivalent; however, the implementation uses the line formulation, as this yields fewer constraints overall and a tighter LP relaxation.

Constraint Enumeration. For an $n \times n$ grid, there are n^2 grid cells, yielding $\binom{n^2}{3} = O(n^6)$ possible triplets of points. However, not all triplets lie on a common line. The number of unordered collinear triples grows as $\Theta(n^4 \log n)$. If constraints were written separately for every collinear triple, this would be the relevant count. In our ILP we instead aggregate all collinear points on the same line into a single inequality $\sum_{p \in \ell \cap G_n} x_p \leq 2$, so the relevant quantity is the number $|L_n^{\geq 3}|$ of distinct lines containing at least three grid points. This satisfies $|L_n^{\geq 3}| = \Theta(n^4)$, giving $\Theta(n^4)$ line constraints, a reduction of $\Theta(n^6) \to \Theta(n^4 \log n) \to \Theta(n^4)$.

This formulation establishes the combinatorial and geometric foundation upon which all subsequent methods are built.

2.2. Integer Linear Programming

We solve the No-Three-In-Line problem using integer linear programming (ILP) (Eppstein, 2018) with the Gurobi

Optimizer (Gurobi Optimization, LLC, 2024). Using the formulation from Section 2.1, the implementation focuses on efficient constraint generation and solver configuration.

Constraint Generation. To enforce the no-three-in-line requirement, we enumerate collinear point sets as formalized in Algorithm 1.

Algorithm 1 ILP Constraint Generation

Input: Grid size n
Output: Set of linear constraints $\mathcal{C}$
Initialize $\mathcal{C} \leftarrow \emptyset$, line dictionary $\mathcal{L} \leftarrow \{\}$
for each pair of distinct grid points (i_1, j_1) and (i_2, j_2) **do**
  Compute line equation $ax + by + c = 0$ through both points
  Normalize: $(a, b, c) \leftarrow (a, b, c)/\gcd(|a|, |b|, |c|)$
  Apply consistent sign convention to obtain canonical form ℓ
  if $\ell \notin \mathcal{L}$ **then**
    $\mathcal{L}[\ell] \leftarrow \emptyset$
  end if
  $\mathcal{L}[\ell] \leftarrow \mathcal{L}[\ell] \cup \{(i_1, j_1), (i_2, j_2)\}$
end for
for each line $\ell \in \mathcal{L}$ with $|\mathcal{L}[\ell]| \geq 3$ **do**
  Add constraint: $\sum_{(i,j)\in\mathcal{L}[\ell]} x_{i,j} \leq 2$ to $\mathcal{C}$
end for
Output: $\mathcal{C}$

Implementation. The resulting ILP model contains n^2 binary variables and $\Theta(n^4)$ linear constraints, corresponding to the number of distinct lines passing through at least three grid points. We solve the model using Gurobi's branch-and-bound algorithm (Nemhauser & Wolsey, 1988; Huang et al., 2021) with default settings for cutting planes and primal heuristics.

2.3. PatternBoost Transformer Learning

We apply the PatternBoost methodology (Charton et al., 2024) to the No-Three-In-Line problem, using a transformer to learn patterns from high-quality configurations and guide the search toward better solutions.

Grid Representation and Tokenization. Each $n \times n$ grid configuration is encoded as a sequence of tokens for transformer input. We map each grid cell (i, j) to a unique integer token $t_{i,j} = i \cdot n + j \in \{0, 1, \ldots, n^2 - 1\}$. A configuration containing points at positions $\mathcal{P} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ is represented as the sequence of tokens $(t_{i_1,j_1}, t_{i_2,j_2}, \ldots, t_{i_k,j_k})$. Each token corresponds to a single grid cell, yielding a vocabulary of size n^2. We prepend a special start token to each sequence.

Transformer Architecture. We employ a GPT-2 style decoder-only transformer (Radford et al., 2019) based on Karpathy's Makemore implementation (Karpathy, 2024). The model consists of 4 transformer layers with 4 attention heads and a hidden dimension of 64. The feedforward network in each layer expands to dimension 256 ($4\times$ hidden dimension) with GELU activation (Hendrycks & Gimpel, 2016). Token embeddings and learned positional embeddings are summed before processing through the transformer layers. The model performs next-token prediction via cross-entropy loss over the vocabulary. Figure 2 illustrates the complete architecture with residual connections between layers.

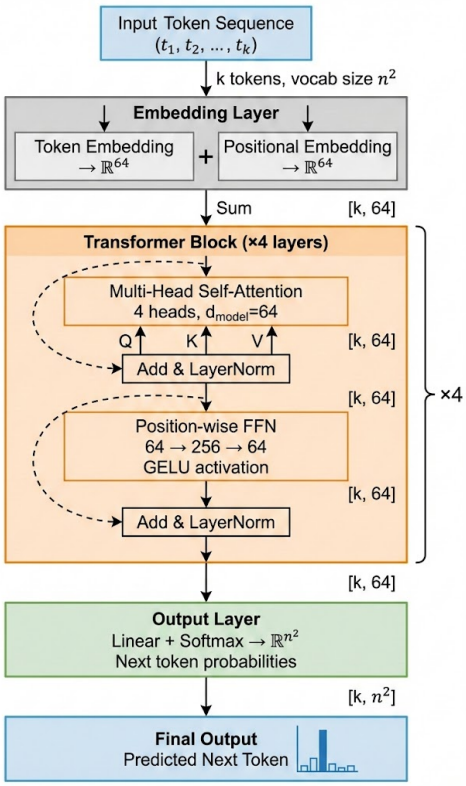

Figure 2. PatternBoost transformer: 4 layers, 4 heads, dimension 64, with residual connections.

Training Data Generation. Initial training data is generated via greedy saturation from the empty grid, formalized in Algorithm 2. This greedy lookahead strategy produces saturated configurations. We generate 20,000 such configurations in batches of 500, forming the initial training pool.

Priority Queue (TopPool). Best configurations are maintained in a fixed-capacity priority queue implemented as a min-heap of size $K = 20{,}000$. Each entry stores a tuple (s, σ) where s is the configuration score (number of points) and σ is the tokenized sequence. When the heap is full and a new configuration with score $s' > s_{\min}$ arrives, the minimum-score entry is evicted and the new entry is inserted. This ensures training always uses the top-performing

Algorithm 2 Greedy Saturation for Training Data Generation

Input: Empty $n \times n$ grid
Output: Saturated configuration with no valid moves remaining
Initialize grid $G \leftarrow \emptyset$ (all cells empty)
while valid placements exist **do**
  $\mathcal{C} \leftarrow \{(i,j) : G[i,j] = 0\}$ {All empty cells}
  for each candidate $(i,j) \in \mathcal{C}$ **do**
    Simulate placing point at (i,j)
    Update forbidden regions based on collinearity constraints
    score$[i,j] \leftarrow$ |remaining available cells|
  end for
  $(i^*, j^*) \leftarrow \arg\max_{(i,j)\in\mathcal{C}}$ score$[i,j]$ {Ties broken randomly}
  Place point at (i^*, j^*) in G: $G[i^*, j^*] \leftarrow 1$
end while
Output: G

configurations discovered.

Iterative Refinement Loop. PatternBoost alternates between local and global optimization, running until a valid $2n$-point configuration is found or a generation budget is exhausted, formalized in Algorithm 3.

Table 1 summarizes the complete hyperparameter configuration.

Table 1. PatternBoost Hyperparameters

PARAMETER	VALUE
Transformer Architecture	
LAYERS	4
ATTENTION HEADS	4
HIDDEN DIMENSION	64
FEEDFORWARD DIMENSION	256
ACTIVATION FUNCTION	GELU
LOSS FUNCTION	CROSS-ENTROPY
Training Configuration	
OPTIMIZER	ADAMW
LEARNING RATE	10^{-4}
WEIGHT DECAY	0.1
BATCH SIZE	64
STEPS PER GENERATION	5,000
GENERATION BUDGET $T_{\max}$	5–80 (GRID-SIZE DEPENDENT)

Symmetry Handling. To increase training data diversity and prevent redundant storage, we apply symmetry processing at two stages. During data augmentation, each configuration is expanded to all 8 symmetric versions (4 rotations and 2 reflection states), increasing the training dataset size by $8\times$. When inserting into TopPool, configurations are mapped to their lexicographically smallest canonical form

Algorithm 3 PatternBoost Iterative Refinement

Input: Initial pool $\mathcal{P}_0$ of 20,000 configurations, generation budget $T_{\max}$
Output: Optimized configuration pool $\mathcal{P}_T$
for generation $t = 1$ to $T_{\max}$ **do**
  Load Pool: Initialize TopPool with $\mathcal{P}_{t-1}$
  Data Augmentation: $\mathcal{D}_t \leftarrow \emptyset$
  for each configuration $c \in \mathcal{P}_{t-1}$ **do**
    Generate all 8 symmetries (4 rotations $\times$ 2 reflections)
    Apply canonical form deduplication
    Add augmented configurations to $\mathcal{D}_t$
  end for
  Train Transformer: Train model θ_t for 5,000 steps on $\mathcal{D}_t$ using AdamW (Loshchilov & Hutter, 2019) with cross-entropy loss
  Generate Candidates: Sample 20,000 token sequences $\{\sigma_1, \ldots, \sigma_{20000}\}$ from trained model π_{θ_t}
  for each sequence σ_i, $i = 1$ to 20000 **do**
    Decode: Convert token sequence σ_i to grid coordinates
    Greedy Saturation: Apply Algorithm 2 to complete partial configuration
    Evaluate: Compute score s_i (number of points placed)
    Attempt to insert (s_i, σ_i) into TopPool (min-heap with capacity 20,000)
  end for
  Save Generation: Persist updated pool $\mathcal{P}_t$ to disk
end for
Output: $\mathcal{P}_T$

under these symmetries to prevent storing duplicate equivalent solutions. This approach maximizes training signal while maintaining pool diversity.

2.4. Proximal Policy Optimization Reinforcement Learning

We formulate the No-Three-In-Line problem as a sequential decision-making task and solve it using deep reinforcement learning. We employ Proximal Policy Optimization (PPO) (Schulman et al., 2017) with action masking to train agents that learn to place points on the grid while satisfying the collinearity constraint.

2.4.1. Markov Decision Process Formulation

The problem is modeled as a Markov Decision Process (MDP) (Sutton & Barto, 2018; Towers et al., 2024) with tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. The state space $\mathcal{S}$ consists of $n \times n$ binary grids $s_t \in \{0,1\}^{n\times n}$, where $s_t(i,j) = 1$ indicates a placed point. The action space $\mathcal{A}$ is discrete with $|\mathcal{A}| = n^2$, and action $a = i \cdot n + j$ places a point at position (i,j). We use action masking to restrict the agent to empty cells only. The transition function $P(s_{t+1}|s_t, a_t)$ is deterministic: choosing action a_t sets $s_{t+1}(i,j) = 1$ at the selected location.

The reward function balances exploration with constraint satisfaction using sparse rewards and dense penalties:

$$R(s_t, a_t) = \begin{cases} +1.0 & \text{valid placement} \\ -10 & \text{collinearity violation} \\ -1 & \text{occupied cell} \\ +100 & k = 2n,\ \text{violations} = 0 \\ +10 & k = 2n,\ \text{violations} > 0 \end{cases} \tag{5}$$

where k is the current number of placed points. The final two cases correspond to perfect completion and imperfect completion, respectively. The 10:1 penalty-to-reward ratio encourages constraint satisfaction over rapid point accumulation, while the large perfect solution bonus creates a strong gradient toward zero-violation configurations. We use discount factor $\gamma = 0.99$.

Collinearity is detected using the determinant formula: three points (x_1, y_1), (x_2, y_2), (x_3, y_3) are collinear if $(x_2 - x_1)(y_3 - y_1) = (x_3 - x_1)(y_2 - y_1)$. For each new placement, we check all $\binom{k}{2}$ pairs of existing points. Episodes terminate when $k = 2n$ points are placed, with no early termination on violations to allow recovery from mistakes.

2.4.2. PPO Algorithm and Network Architecture

We employ Proximal Policy Optimization with action masking (MaskablePPO) from Stable-Baselines3 Contrib (Raffin et al., 2021). PPO optimizes a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\hat{A}_t, \\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big] \tag{6}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\epsilon = 0.2$ is the clipping parameter. The total loss combines the clipped objective with value function loss $L^{VF}(\theta) = (V_\theta(s_t) - V_t^{\text{target}})^2$ and entropy bonus $S[\pi_\theta](s_t)$:

$$L_{\text{total}}(\theta) = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \tag{7}$$

with coefficients $c_1 = 0.5$ and $c_2 = 0.01$. Advantage estimates $\hat{A}_t$ are computed using Generalized Advantage Estimation (GAE) (Schulman et al., 2016) with $\lambda = 0.95$ and $\gamma = 0.99$.

The policy network uses a multi-layer perceptron (MLP) with two hidden layers of 512 units each and ReLU activations. The architecture processes the flattened n^2-dimensional observation vector to produce action probabilities over the n^2 action space. The policy network (actor) $\pi_\theta(a|s)$ and value function (critic) $V_\theta(s)$ share feature extraction layers but have separate output heads, as shown in Figure 3.

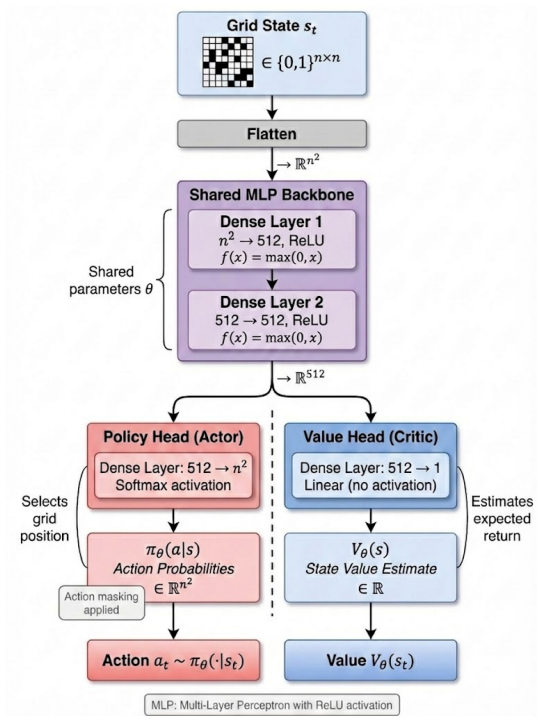

Figure 3. PPO actor-critic with shared 512×2 MLP backbone splitting into policy and value heads.

2.4.3. Training Configuration

We use the Adam optimizer (Kingma & Ba, 2015) with learning rate $\alpha_0 = 3 \times 10^{-4}$ and exponential decay: $\alpha_t = \alpha_0 \cdot (0.8)^{\lfloor t/10^6 \rfloor}$. Training employs 30 parallel environments, collecting 2048 steps per environment (61,440 total samples) before each policy update using batches of 1024 samples. The complete training runs for 10 epochs of 1,000,000 steps each (10,000,000 total steps). Table 2 summarizes the complete hyperparameter configuration.

Table 2. PPO Hyperparameters

Parameter	Value
Training epochs	10
Network architecture	[512, 512] MLP
Activation function	ReLU
Learning rate	3×10^{-4}
Optimizer	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
GAE λ	0.95
Discount γ	0.99
PPO clip ϵ	0.2
Value coefficient c_1	0.5
Entropy coefficient c_2	0.01

Model performance is evaluated every epoch over 20 episodes using success rate (fraction achieving $2n$ points with zero violations), average violations, and a composite score that balances points placed, efficiency, and constraint satisfaction. The best model is selected based on maximum composite score across all training epochs.

2.5. SciML Gradient Flow

Discrete enumeration scales exponentially in n, and the methods of Sections 2.2–2.4 collectively cover only up to $n = 19$, well below the symmetry-reduced records cited in Section 1. We therefore reformulate the problem in continuous form: instead of choosing a $0/1$ placement, we evolve a real-valued density on the grid under a smooth energy whose landscape biases trajectories toward valid $2n$-point configurations (Schuetz et al., 2022). Solving the resulting ordinary differential equation by gradient flow turns combinatorial search into trajectory integration, a setting in which the Scientific Machine Learning toolchain (Rackauckas & Nie, 2017; Karniadakis et al., 2021) provides mature, GPU-accelerated solvers.

2.5.1. Continuous Relaxation and Energy Function

Let $\mathbf{x} \in [0,1]^{n^2}$ be a continuous relaxation of the binary indicator matrix $\mathbf{X}$ of Section 2.1, with $x_{i,j}$ now interpreted as a probability of a point at cell (i,j). We optimise an energy with three terms,

$$E(\mathbf{x}) = \alpha\, E_{\text{coll}}(\mathbf{x}) + E_{\text{count}}(\mathbf{x}) + \mu\, E_{\text{bin}}(\mathbf{x}), \tag{8}$$

where $E_{\text{count}} = -\sum_{i,j} x_{i,j}$ rewards point count, and $E_{\text{bin}} = \sum_{i,j} 4x_{i,j}^2(1-x_{i,j})^2$ is a double-well term that pushes each $x_{i,j}$ toward $\{0,1\}$. The collinearity penalty is

$$E_{\text{coll}}(\mathbf{x}) = \sum_{\ell} \frac{1}{c_\ell} \sum_{\substack{i<j<k \\ i,j,k \in \ell}} x_i\, x_j\, x_k, \tag{9}$$

where the outer sum runs over all lines ℓ through at least three grid points, c_ℓ is the mean incidence of points on line ℓ (*mean-incidence* normalisation), and the inner sum is the elementary symmetric polynomial e_3 of the mass values on ℓ. This term is zero whenever at most two points on any line carry nonzero mass, and grows as soon as a third point is activated. The collinearity term reuses the line-incidence structure of Section 2.1, replacing $O(n^6)$ triple enumeration with $\Theta(n^4)$ per-line sums; the mean-incidence normalisation keeps α within a narrow range as n grows.

2.5.2. Gradient Flow Dynamics

From a uniform random initialisation $\mathbf{x}(0) \in [0,1]^{n^2}$, we evolve the relaxed configuration by gradient flow,

$$\frac{d\mathbf{x}}{dt} = -\nabla E(\mathbf{x}), \qquad t \in [0,T], \tag{10}$$

integrated with a fixed-step RK4 scheme on the GPU. The horizon T is grid-size dependent (Section 3.5).

2.5.3. Parallel Trajectory Search on Apple Metal

We implement Equation (10) as a custom Metal compute kernel that integrates R independent trajectories in parallel, one per GPU thread, with no inter-thread communication (Besard et al., 2022). Each thread holds private scratch for an $n \times n$ board (capped at $n \leq 32$ by kernel-local memory) and reads a precomputed sparse line table that encodes the constraint structure once per run. We use $R = 5000$ trajectories per run. After integration, each trajectory is converted to a binary configuration by retaining the top $2n$ entries of $\mathbf{x}(T)$ above a threshold τ, and the best valid configuration found along each trajectory is recorded. Most successful runs reach their final placement within the first 10–20% of the integration horizon, motivating an early-best heuristic that returns once any thread reports a zero-violation $2n$-point configuration.

2.5.4. Hyperparameters and Parameter Scaling

Table 3 summarises the SciML configuration. Energy and solver parameters are grid-size aware: α and T are tuned per n from the schedule used in our experiments, and we discuss their empirical scaling behaviour in Section 3.5.

Table 3. SciML Gradient Flow Hyperparameters

PARAMETER	VALUE
PARALLEL TRAJECTORIES R	5000
TIME HORIZON T	10 ($n \leq 15$), 25 ($n = 16$), 30 ($n = 17, 18$)
COLLINEARITY PENALTY α	PER-n SCHEDULE FOR $n \leq 18$ (FIG. 5)
GPU BACKEND	APPLE METAL GPU

3. Results

3.1. Experimental Setup

We evaluated each method across multiple grid sizes to assess solution quality and scalability. ILP was tested on grids up to $n = 19$, PatternBoost on grids up to $n = 15$, and PPO on grids up to $n = 11$. All experiments were conducted on an Apple M2 with 32 GB RAM. Software implementations used Python with PyTorch for neural network training, Gurobi for integer programming, and Stable-Baselines3 for reinforcement learning.

3.2. Solution Quality and Scalability

Table 4 presents the solution quality achieved by each method across different grid sizes. We report the maximum number of points successfully placed while satisfying the no-three-in-line constraint (zero violations). For reference, we include known optimal values $2n$ for grid size $n \times n$.

Integer Linear Programming. ILP achieved provably optimal solutions up to $n = 19$ (38 points), providing the strongest baseline across all tested grid sizes.

PatternBoost. PatternBoost achieved optimal solutions up to $n = 14$ (28 points) and discovered 29-point configurations on $n = 15$ grids, one short of the known optimum, but did not scale beyond $n = 15$. Training ran for between 5 and 80 generations (5,000 steps each) per grid size.

Proximal Policy Optimization. PPO achieved perfect solutions on $n = 10$ grids (20 points, zero violations) but incurred a collinearity violation on $n = 11$ grids, revealing difficulty maintaining constraint satisfaction as problem complexity increases. Each model was trained in a single run ($\sim$66–73 min for $n = 10$; $\sim$69 min for $n = 11$).

SciML. The gradient flow recovered optimal $2n$-point configurations for every $n \in \{10, \ldots, 18\} \setminus \{17\}$, with per-run success rates from $4/10$ at $n = 14$ to $10/10$ at $n = 10$ and $n = 12$. For $n = 18$, the fastest successful run reached 36 points in $\sim$218 s on Apple M2. Two failure modes are visible: at $n = 17$, we reproduced a valid 34-point solution for a specific seed, but the original 10-run ablation yielded 0/10 success (failed runs bottomed out at two residual violations); at $n = 19$ the best attempt reached 35 points with three violations.

Figure 10 shows the best SciML solution on an 18×18 grid achieving 36 points.

3.3. Training Dynamics

Figure 4 shows PatternBoost's loss over 15,000 training steps (the first 3 of 20 generations; each generation runs 5,000 steps). Both training and test loss drop from above 5.0 to near 0.1, with a minimal train-test gap throughout, indicating strong generalization without overfitting.

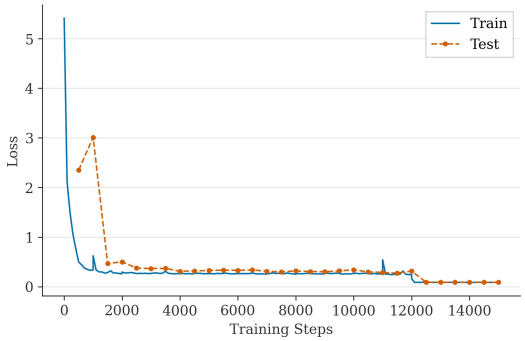

Figure 4. PatternBoost training and test loss over 15,000 steps.

3.4. Parameter Scaling for SciML

Mean-incidence normalisation is used throughout all grid sizes. α is hand-tuned per n: higher for small grids ($\alpha = 3.3$ at $n = 2$, rising to $\alpha = 16.7$ at $n = 10$) and held in a narrower range for larger grids ($\alpha \approx 10$ for $n = 11$–16, decreasing to 7 at $n = 18$). The integration horizon T stays at 10 for $n \leq 15$, 25 at $n = 16$, and 30 for $n = 17, 18$. The reported $n = 19$ exploratory sweep used a separate $T = 10$, $\alpha = 8$ setting. The failure modes at $n = 17$ and $n = 19$ coincide with the schedule's tightest margin: raising α further tends to freeze trajectories in bad local minima before the gradient flow can converge.

Figure 5 shows the full $\alpha(n)$ and $T(n)$ schedules.

3.5. Limitations

Figures 7 and 8 illustrate PPO's performance transition between successful and failed constraint satisfaction. On 10×10 grids (Figure 8), the agent consistently achieves perfect solutions with 20 points and zero violations, demonstrating successful policy learning for problems of this scale. However, on 11×11 grids (Figure 7), despite placing 22 points, the agent incurs a single collinearity violation (highlighted point in red box), revealing the brittleness of learned policies as problem complexity increases. The violation occurs in a densely populated region where multiple collinearity constraints intersect, suggesting that the policy network struggles to maintain global constraint awareness when the number of placed points exceeds approximately 20. This

Table 4. Solution Quality Comparison Across Methods and Grid Sizes

GRID SIZE	ILP	PATTERNBOOST	PPO	SCIML	KNOWN OPTIMAL
10×10	20	20	20	20 (10/10)	20
11×11	22	22	–	22 (7/10)	22
12×12	24	24	–	24 (10/10)	24
13×13	26	26	–	26 (8/10)	26
14×14	28	28	–	28 (4/10)	28
15×15	30	29	–	30 (8/10)	30
16×16	32	–	–	32 (8/10)	32
17×17	34	–	–	34*	34
18×18	36	–	–	36 (9/10)	36
19×19	38	–	–	–	38

Note: – indicates either a grid size not tested for that method or a run that did not yield a valid zero-violation configuration. * reproduced for one seed; original 10-run sweep: 0/10.

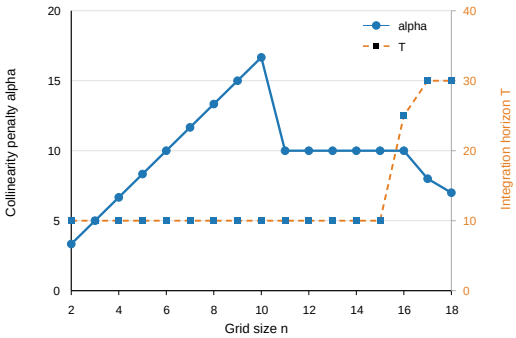

Figure 5. SciML collinearity-penalty schedule $\alpha(n)$ (blue, left axis) and integration horizon $T(n)$ (orange, right axis). Mean-incidence normalisation is used throughout; α is hand-tuned per n.

represents a sharp transition from reliable performance to systematic failure as grid size increases by just one row and column.

PatternBoost's failure at $n = 15$ follows a different pattern. While achieving 29 points on 15×15 grids, one point short of the known optimum of 30, the method demonstrates graceful degradation rather than catastrophic failure. The configurations discovered remain valid (zero violations) but fail to reach optimality, suggesting that the transformer's learned patterns generalize approximately but not perfectly to larger grids. Figure 6 shows PatternBoost's optimal 14×14 solution achieving 28 points, demonstrating the geometric patterns successfully learned through iterative refinement. Figure 9 shows ILP's optimal 19×19 solution for comparison, illustrating the increased complexity that AI methods must learn to replicate at larger scales.

4. Discussion

Across the four paradigms, exact search dominates wherever it applies, while learning-based discrete methods plateau at $n \leq 15$. The SciML continuous-relaxation approach reaches $n = 18$ without symmetry pruning; at $n = 19$ the method does not yet reliably converge, a limitation we attribute to compute rather than a fundamental barrier.

ILP's branch-and-bound approach guarantees optimality but faces exponential growth in problem complexity. PatternBoost's greedy data generation creates a quality-speed trade-off that constrains learned pattern quality at larger n. Reinforcement learning consistently produces constraint violations beyond $n = 10$, revealing difficulty in satisfying geometric constraints through reward-based exploration. The SciML method's continuous-discrete gap at $n = 17$ and $n = 19$ traps trajectories in configurations with residual collinearities that τ cannot reliably round to a valid solution.

Unlike SciML, whose shorter run times made 10-seed sweeps feasible on the M2 MacBook, PPO and PatternBoost were evaluated from a single training run due to hardware budget constraints, so wall-clock times are not directly comparable across methods. The SciML pipeline was evaluated only on Apple Metal, the per-grid hyperparameter schedule was tuned manually, and we did not exploit the rotational and reflective symmetries that classical solvers rely on at larger n.

Hybrid training combining PatternBoost's supervised pre-training with PPO's reinforcement fine-tuning is a natural next direction. Integrating symmetry pruning into the SciML continuous relaxation could extend reliable convergence beyond $n = 18$. Extending this energy-minimisation framework to other forbidden-configuration problems in combinatorial geometry (Schuetz et al., 2022) offers a broader test of the paradigm.

Software and Data

Code for all four methods is available in the public GitHub repository.

Acknowledgements

This work was conducted as part of the STRIDE programme at Queen Mary University of London. We thank the STRIDE team and the School of Mathematical Sciences at Queen Mary University of London for their support, training, development opportunities, resources, and mentorship.

Impact Statement

This work studies optimization strategies for a classical combinatorial geometry problem. The methods explored are computational and mathematical in nature, and we do not anticipate direct negative societal impacts beyond standard concerns related to compute usage and energy consumption associated with machine learning experimentation.

References

Besard, T., Hawkins, M., et al. Metal.jl: Metal programming in Julia, 2022. URL https://github.com/JuliaGPU/Metal.jl.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24:1–61, 2023. URL https://www.jmlr.org/papers/v24/21-0449.html.

Charton, F., Ellenberg, J. S., Wagner, A. Z., and Williamson, G. Patternboost: Constructions in mathematics with a little help from AI. *arXiv preprint arXiv:2411.00566*, 2024.

Eppstein, D. Gurobi versus the no-three-in-line problem. Blog post. https://11011110.github.io/blog/2018/11/12/gurobi-vs-no.html, 2018.

Flammenkamp, A. No-three-in-line: current records and remarks, 2026. URL https://wwwhomes.uni-bielefeld.de/achim/no3in/readme.html. Accessed 2026-04-11.

Graham, N. and Harary, F. Changing and unchanging the diameter of a hypercube. *Discrete Applied Mathematics*, 37-38:265–274, 1992.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2024. https://www.gurobi.com.

Guy, R. K. and Kelly, P. A. The no-three-in-line problem. *Canadian Mathematical Bulletin*, 11:527–531, 1968.

Hall, R. R., Jackson, T. H., Sudbery, A., and Wild, K. Some advances in the no-three-in-line problem. *Journal of Combinatorial Theory, Series A*, 18:336–341, 1975. doi: 10.1016/0097-3165(75)90043-6.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Huang, L., Chen, X., Huo, W., Wang, J., Zhang, F., Bai, B., and Shi, L. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. *arXiv preprint arXiv:2111.06257*, 2021.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021. doi: 10.1038/s42254-021-00314-5.

Karpathy, A. *makemore*, 2024. https://github.com/karpathy/makemore.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980. ICLR 2015.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101. ICLR 2019.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134: 105400, 2021.

Nagy, D. T., Nagy, Z. L., and Woodroofe, R. The extensible no-three-in-line problem. *European Journal of Combinatorics*, 114:103796, 2023.

Nemhauser, G. L. and Wolsey, L. A. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

Pór, A. and Wood, D. R. No-three-in-line-in-3D. *Algorithmica*, 47:481–488, 2007. doi: 10.1007/s00453-006-0158-9.

Rackauckas, C. and Nie, Q. DifferentialEquations.jl – A performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, 5(1):15, 2017. doi: 10.5334/jors.151.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Schuetz, M. J. A., Brubaker, J. K., and Katzgraber, H. G. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4:367–377, 2022. doi: 10.1038/s42256-022-00468-6.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1506.02438. ICLR 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. ISBN 9780262039246.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Tsuchiya, K. and Takefuji, Y. A neural network algorithm for the no-three-in-line problem. *Neurocomputing*, 8: 43–49, 1995. doi: 10.1016/0925-2312(94)00003-4.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017. doi: 10.65215/ctdc8e75.

A. Solution Visualisations

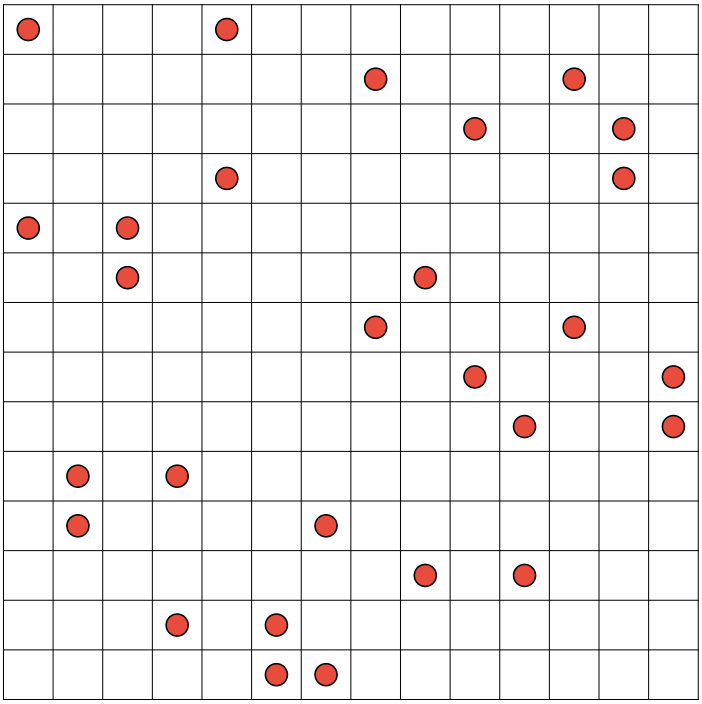

Figure 6. PatternBoost optimal solution on 14×14 grid achieving 28 points.

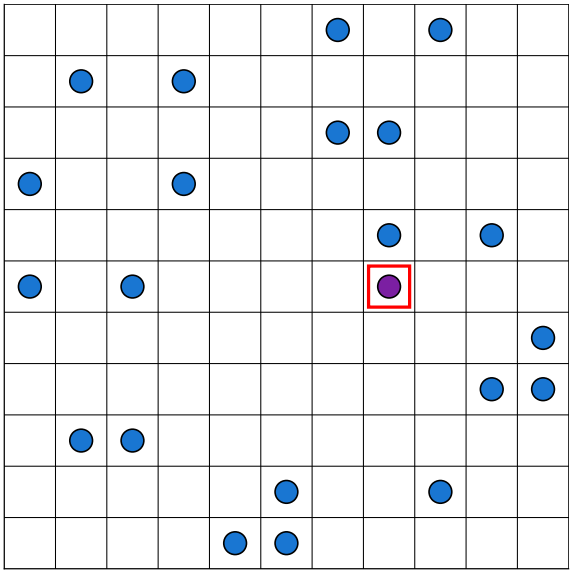

Figure 7. PPO failure on 11×11 grid: 22 points with 1 violation.

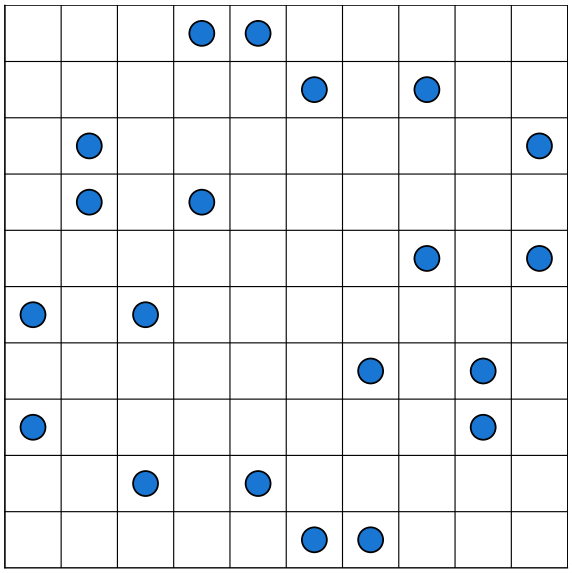

Figure 8. PPO success on 10×10 grid: 20 points with 0 violations.

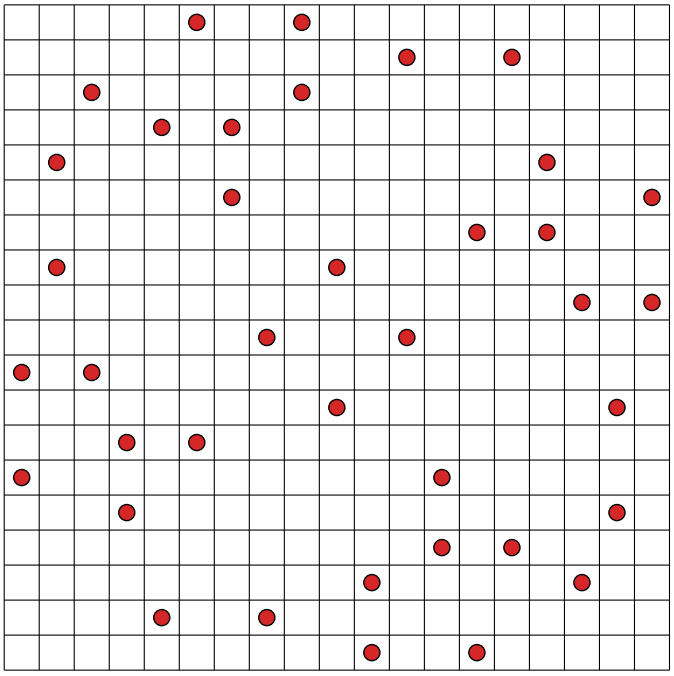

Figure 9. ILP optimal solution on 19×19 grid achieving 38 points.

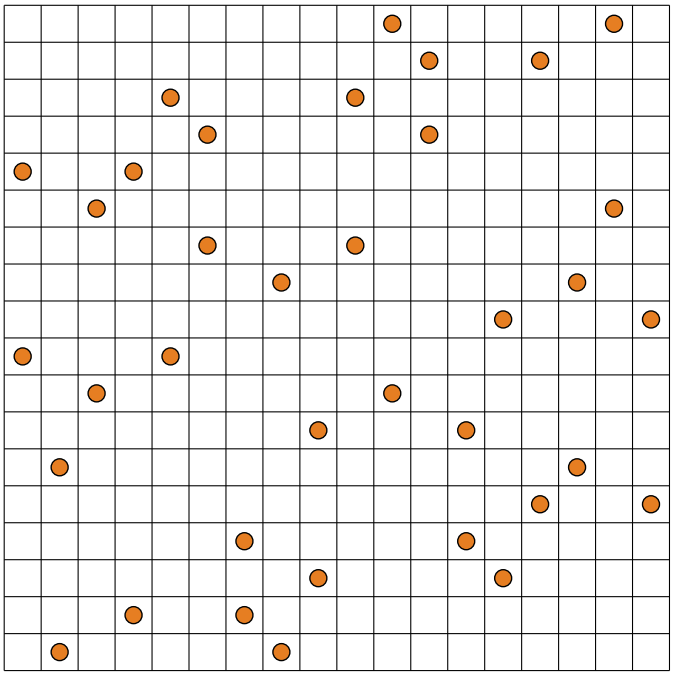

Figure 10. SciML solution on an 18×18 grid achieving 36 points without symmetry pruning.