Learning to Complete Extremal Mathematical Structures

Mengzhou Sun
University of Warsaw
m.sun3@uw.edu.pl

Abstract

Extremal combinatorics seeks rare mathematical configurations that satisfy constraints while maximizing a global objective. Existing neural systems for mathematical discovery train a model to generate complete configurations from scratch. We argue that this framing mismatches the dynamics of effective search: high-quality solutions are more effectively found by preserving a feasible partial core and conditionally regenerating the remaining uncertain objects. We formulate extremal structure discovery as *object-level active-core completion*: the model is trained to conditionally complete masked objects given a feasible partial core, capturing inter-object relations rather than fitting a distribution over full configurations. We instantiate this with a Transformer over object-aligned slots and propose *Inpainting PatternBoost*: in each iteration, elite constructions from a top pool are partially destroyed by random or structured remasking. The model learns to reconstruct the missing objects, and their completions are reinserted into the pool. We evaluate on the No-Three-in-Line problem and maximum sum-of-radii circle packing in the unit square, under a single shared abstraction. On No-Three-in-Line, our framework recovers the optimal $2n$-point configuration for n up to 18, and reaches $2n-1$ points for $n \in \{19, 20, 25\}$, to our knowledge the best results among learning-based methods. On circle packing in a square, where FlowBoost combines neural generation with gradient-based local search, our approach matches or exceeds FlowBoost when paired with the same local search; even without local search, it reaches an average of 88.5% of FlowBoost's reported score across instances.[1]

1 Introduction

Recent AI systems have accelerated mathematical discovery through several distinct interfaces: guiding formal proof search and generating natural-language proofs [Trinh et al., 2024, Hubert et al., 2026, Huang and Yang, 2025], discovering faster algorithms via program evolution [Fawzi et al., 2022, Romera-Paredes et al., 2024, Novikov et al., 2025], and finding explicit constructions or counterexamples in concrete problems. This paper concerns the last interface, focusing on **extremal combinatorics**, a setting in which the goal is to construct combinatorial objects that satisfy constraints while optimizing a global objective.

Machine learning has begun to serve as an effective search partner for such problems. Wagner showed that reinforcement-learning search can discover non-trivial constructions and counterexamples in extremal graph theory [Wagner, 2021, Mehrabian et al., 2024]. PatternBoost [Charton et al., 2024] and its continuous extension FlowBoost [Bérczi et al., 2026] formalise a template that alternates two phases: a classical local search refines candidate constructions toward higher scores, while a generative model trained on the improved samples learns global patterns, and produces seeds for the

[1] Code repository: https://github.com/annoymous-1234/Inpainting-PatternBoost

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

next round of local search. Together, these works show that rare high-quality constructions contain learnable structure that guides search toward the near-extremal tail.

A common thread runs through these systems: the model is trained on, and generates complete constructions from scratch. We view this unconditional generation interface as a potential mismatch with the dynamics of searching for extremal structures for two reasons. First, feasibility in extremal problems is inherently relational: how an object should be placed depends on the objects already committed, so a model trained to imitate complete constructions captures these inter-object relations only implicitly, through the joint that links all objects at once. Second, productive mathematical search rarely builds a solution from scratch; it more often preserves a feasible partial structure, identifies the uncertain parts, and refines them in place. An interface that requires a fresh complete sample at every step is ill-suited to this dynamic.

A secondary issue, present in some of this line of work, is to serialise the construction into a token sequence, which forces an arbitrary ordering on what is essentially an unordered object collection and inflates the multimodal landscape the model must fit. We address it through our object-level representation.

We propose to make *conditional completion* the central learning problem. Rather than fitting a marginal $p_\theta(x)$ over complete configurations, we train a *conditional completion operator*

$$p_\theta(x_M \mid x_{\bar{M}}, M),$$

where M specifies which objects are masked out, $x_{\bar{M}}$ is the preserved partial construction (the *active core*), and x_M denotes the masked objects to be regenerated. The model's role shifts from imitating complete solutions to learning object-level compatibility relations: which objects conflict, coexist, or can replace one another within a given core.

We realize this principle with an object-level Transformer in which each token corresponds to a mathematical object (a grid-point slot or a circle), an architectural choice already used by FlowBoost [Bérczi et al., 2026]; our contribution is to pair it with *active-core conditioning*, so that the preserved core remains feasible throughout search. Algorithmically, we instantiate this as *Inpainting PatternBoost*, which retains the local/global alternation of PatternBoost but replaces complete generation with conditional reconstruction of remasked elite constructions.

We demonstrate the framework on two complementary domains: the **No-Three-in-Line** problem (discrete, grid-based) and **maximum sum-of-radii circle packing** in the unit square (continuous, geometric). Despite their structural differences, both fit the same active-core completion abstraction without domain-specific architectural changes. Empirically, our method matches or improves on prior learning-based results across both problems, supporting the claim that conditional completion is a more effective search interface than unconditional generation. Example constructions found by our framework are shown in Appendix A.

Contributions.

- **Formulation.** We formulate extremal structure discovery as object-level active-core completion, a principled alternative to generating complete configurations from scratch.
- **Method.** We introduce *Inpainting PatternBoost*, which modifies PatternBoost's global learning phase to train on remasked elite constructions and to sample conditional completions conditioned on a preserved active core, rather than generating full configurations.
- **Empirical results.** We achieve state-of-the-art results among learning-based methods on both problems, with ablations isolating the contributions of representation, active-core conditioning, and remasking policies.

2 Related Work

2.1 AI for mathematical discovery

Of the three AI-for-math interfaces listed in the introduction, we focus here on the third. Wagner [Wagner, 2021] introduced RL-search for extremal graph constructions and counterexamples; Mehrabian et al. [Mehrabian et al., 2024] scaled this line via AlphaZero with tabu search to find

increasingly large extremal graphs. PatternBoost [Charton et al., 2024] introduced the local-global iteration template; its continuous extension FlowBoost [Bérczi et al., 2026] applied this template to geometric extremal problems, including the circle-packing problem we study here. We share the local-global structure of PatternBoost and FlowBoost but train a conditional completion operator over partial constructions instead. Closest to our experiments, Ramanathan et al. [Ramanathan et al., 2025] apply PatternBoost and PPO reinforcement learning to No-Three-in-Line; their PatternBoost variant matches the optimal $2n$-point solution at grids up to $n = 14$ but not at the larger grid sizes they evaluate. Beyond the construction line, recent work has begun systematic mathematical exploration at scale [Georgiev et al., 2025].

2.2 Neural combinatorial optimization and masked reconstruction

Extremal combinatorics overlaps in problem form with combinatorial optimization, and several design choices in our method, particularly the diffusion-based generator for circle packing in a square, are borrowed from neural combinatorial optimization (NCO). NCO has produced learned construction and improvement operators for routing and integer programming [Kool et al., 2019, Nair et al., 2020], with diffusion-based variants such as DIFUSCO adapting denoising generative models to this setting [Sun and Yang, 2023]. Most directly, our masked-reconstruction interface shares its core with MaskCO [Chen et al., 2026], which reformulates NCO around masked solution reconstruction; we adapt the same idea to extremal structure discovery, with a mathematically certified active core and object-aligned mathematical slots in place of generic decision variables.

Preserving part of a solution and re-optimizing the rest is the classical idea of large neighborhood search (LNS) [Shaw, 1998], later extended with adaptive selection among destroy and repair heuristics [Ropke and Pisinger, 2006]. Our method can also be partially viewed as neural LNS, with remasking as the destroy step and conditional completion as the repair step.

2.3 Object-level Transformers and inpainting diffusion

Set-style architectures for unordered object collections [Zaheer et al., 2017, Lee et al., 2019] provide the conceptual basis for the object-level representation we use. In the continuous circle-packing domain, the conditional generator is realised with denoising diffusion probabilistic models (DDPM) [Ho et al., 2020] and flow matching (FM) [Lipman et al., 2023], and our active-core conditioning is closest in spirit to the inpainting perspective of RePaint [Lugmayr et al., 2022], with the difference that the known content is a mathematically certified partial construction rather than an observed image region.

3 Framework

We formulate extremal structure discovery as object-level active-core completion. Three abstract components compose the framework: a slot-based representation, a partial-completion operator q_θ, and an outer search loop we call *Inpainting PatternBoost*. We define them in turn; Section 4 instantiates each on two extremal combinatorial problems.

3.1 Slot configurations

In extremal combinatorics, the target combinatorial object to be optimized is typically a collection of unordered or weakly ordered mathematical objects. We represent it by a fixed collection of K semantic slots. A complete construction is represented as a full slot configuration

$$x = (x_1, \ldots, x_K), \qquad x_i \in \mathcal{Z}_i,$$

where the slot state space $\mathcal{Z}_i$ is problem-dependent and may be discrete, continuous, or mixed. When the underlying construction has variable size, $\mathcal{Z}_i$ includes a null or masked state so that unused capacity is represented within the fixed width.

A configuration is checked by a feasibility predicate $C(x) \in \{0, 1\}$ and scored by an objective $F(x) \in \mathbb{R}$. The search problem is

$$x^\star \in \arg\max_{x \in \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_K} F(x) \quad \text{s.t.} \quad C(x) = 1.$$

3.2 Partial construction completion

Given a slot configuration x, we choose a subset of slots

$$M \subseteq [K]$$

to be masked. The complementary set $\bar{M} = [K] \setminus M$ is the *active core*. The learning problem is to model a conditional completion operator

$$q_\theta(x_M \mid x_{\bar{M}}, M),$$

which proposes new values for the masked slots conditioned on the active core. When the masked slots also carry useful draft information, such as values inherited from a re-masked elite construction or a problem-specific heuristic, we pass this as extra input to the conditional completion operator. A proposed completion produces a candidate

$$\hat{x} = (x_{\bar{M}}, \hat{x}_M),$$

which is then checked, repaired, or rejected by a problem-specific environment. Completion need not fill all of M at once: we iteratively select a high-confidence subset of $\hat{x}_M$, commit those slots into the active core $\bar{M}$, and re-apply q_θ, growing the active core one step at a time until all masked slots are resolved (Section 4). This formulation is particularly well suited to problems in extremal combinatorics for the following reasons.

From absolute to relational structure. Symmetries and equivalent reorderings make near-optimal extremal solutions highly multimodal, and the diversity in the early stages of construction is combinatorially explosive. The absolute placement of any single object is therefore poorly determined; a model fit directly to complete high-quality constructions risks averaging across these modes, producing predictions that need not correspond to any valid configuration. What the model can reliably learn is relational structure, namely whether an object is feasible given those already committed. Fixing an active core breaks much of this symmetry and exposes the relational signal directly, turning learning into a compatibility and conflict-resolution problem rather than unconditional generation.

Data amplification from rare elites. High-quality constructions are rare near the optimum. Masking different subsets of a single elite construction yields many distinct conditional reconstruction tasks, amplifying the effective training signal from each sample [Chen et al., 2026].

3.3 Inpainting PatternBoost

We implement the active-core completion framework as *Inpainting PatternBoost*. The outer loop preserves the elite pool and the train–generate alternation of PatternBoost [Charton et al., 2024], but replaces whole-configuration generation with conditional inpainting: elite constructions are partially destroyed by remasking, and the model learns to reconstruct the masked slots conditioned on the active core. Training uses the same masking interface, framed as a denoising-style objective.

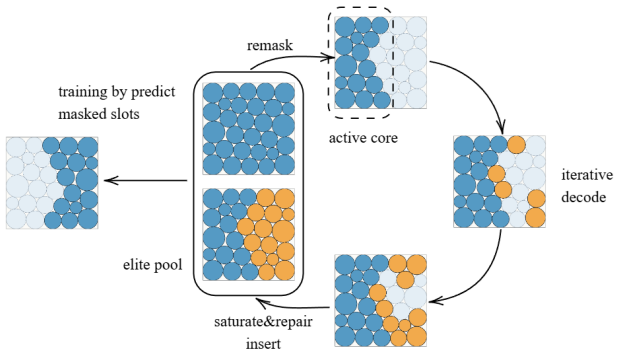


Figure 1: Inpainting PatternBoost loop on circle packing in a square. Blue circles form the active core, pale circles are masked slots, and orange circles are newly completed objects.

Algorithm 1 Inpainting PatternBoost

Require: Environment $\mathcal{E}$, initial pool $\mathcal{P}_0$ of size P, model θ

$\mathcal{P} \leftarrow \mathcal{P}_0$
for generation $g = 1, \dots, G$ **do**
  for step $t = 1, \dots, T$ **do**
    Sample $x \sim \mathcal{P}$ and random mask M
    Update θ to minimise $\mathcal{L}(\theta;\, x, M)$
  end for
  for candidate $c = 1, \dots, C$ **do**
    Sample $x \sim \mathcal{P}$ and mask $M \sim \pi(M)$
    for decoding step $s = 1, \dots, S$ **do**
      $\hat{x}_M \leftarrow q_\theta(x_{\bar{M}},\, M)$
      Select $A \subseteq M$ via decoding schedule $\mathcal{D}$
      $(x_{\bar{M}},\, M) \leftarrow (x_{\bar{M}} \cup \hat{x}_A,\, M \setminus A)$ ▷ commit $\hat{x}_A$ into active core
    end for
    $\tilde{x} \leftarrow \mathcal{E}.\mathrm{Repair\&Saturate}(x_{\bar{M}})$
    $\mathcal{P} \leftarrow \mathrm{InsertSorted}(\mathcal{P},\, \tilde{x})$ ▷ Keep top-P candidates
  end for
end for
return $\arg\max_{x \in \mathcal{P}} F(x)$

At each generation, Algorithm 1 (illustrated in Figure 1) alternates a training phase and a generation phase, both sampling from a top-P elite pool $\mathcal{P}$. The training phase corrupts elite constructions with random masks and updates the model to reconstruct the masked slots. The generation phase samples a mask M from a remasking policy $\pi(M)$, then the model iteratively decodes the masked slots: at each of S steps the model proposes values for the currently masked slots and a decoding schedule $\mathcal{D}$ commits a subset of those proposals to the active core. Each completed candidate is repaired and greedily saturated by the environment, then inserted into $\mathcal{P}$ to keep the top-P entries. The remasking policy π, decoding schedule $\mathcal{D}$, and elite-pool sampling strategy can be either domain-specific or uniformly random. Concrete instantiations of them are in Section 4 and Appendix B.1.

4 Implementation

This section instantiates the framework of Section 3.3 on both problems. The shared backbone is a pre-norm Transformer encoder with L layers, H attention heads and embedding dimension d. Each slot i contributes one token, and full bidirectional self-attention is applied with no causal ordering. Training uses random masks, but with the same mask ratio distribution as the remasking policy π. Per-problem setups follow in Sections 4.1 and 4.2, and the implementation details are in Appendix B.

4.1 No-Three-in-Line

Problem. The No-Three-in-Line problem [Dudeney, 1917] is a discrete problem on the integer grid

$$G_n = \{0, \dots, n-1\}^2.$$

The goal is to select as many grid points as possible while ensuring that no straight line contains three selected points:

$$\max_{S \subseteq G_n} |S| \quad \text{s.t.} \quad |\ell \cap S| \leq 2 \quad \text{for every line } \ell.$$

Slot configuration. Since every vertical line $x = i$ can contain at most two selected points, we allocate two slots per column, setting $K = 2n$. Each slot has categorical state space

$$\mathcal{Z}_{i,j} = \{\texttt{[MASK]}, 0, 1, \dots, n-1\}, \qquad i \in \{0, \dots, n-1\},\ j \in \{1, 2\}.$$

If $x_{i,j} = y$, the slot represents the point (i, y); if $x_{i,j} = \texttt{[MASK]}$, the slot is unused or released for completion. The induced point set is

$$S(x) = \{(i, y) : x_{i,j} = y \neq \texttt{[MASK]}\},$$

and a valid complete configuration is one for which $S(x)$ contains no three collinear points.

Embedding. Each slot (i, j) is embedded as

$$e_{ij} = \mathrm{Emb}_{\mathrm{state}}(x_{ij}) + \mathrm{Emb}_{\mathrm{col}}(i) + \mathrm{Emb}_{\mathrm{pos}}(j) + \mathrm{SinEmb}(t),$$

where $\mathrm{Emb}_{\mathrm{pos}}(j)$ distinguishes the two slots of the same column and $\mathrm{SinEmb}(t)$ is a sinusoidal time embedding of the time feature t, which is set to the fraction of currently filled slots. A linear head maps each slot's final hidden state to a categorical distribution over the state space.

Decoding schedule $\mathcal{D}$. $\mathcal{D}$ runs $S = 5$ steps, and at each step it commits up to 20% of the masked slots. Masked slots are ranked by a weighted combination of a model-side confidence score and an environment-side legal score (see Appendix B.4). Intuitively, slots with fewer legal options and higher model confidence are committed first. For each selected slot, three legal candidates sampled from the logits are proposed to the environment, which greedily resolves conflicts between proposed points before they are committed to the active core.

Remasking policy π. $\pi(M)$ mixes uniform random masking with a domain-specific heuristic at a fixed ratio. The heuristic prioritises slots with more remaining legal states (see Appendix B.2).

Training. Training is essentially the same as that of an absorbing-state discrete diffusion [Austin et al., 2021]. We mask a uniformly random subset of slots, and the model is trained to optimize the cross-entropy loss on the masked slots conditioned on the active core.

4.2 Maximum sum-of-radii circle packing in the unit square

Problem. Maximum sum-of-radii circle packing in the unit square is a classical problem in discrete geometry and extremal combinatorics. We henceforth refer to it as *circle packing in a square*. The goal is to place K non-overlapping circles (with flexible radii) inside the unit square $[0, 1]^2$ so as to maximise the sum of their radii. The problem has recently attracted LLM-driven program search such as AlphaEvolve [Novikov et al., 2025]. Instead of searching for centers and radii jointly, we follow the implementation of Bérczi et al. [2026]. We decouple center placement from radius assignment: given any center configuration, the maximum sum of radii is determined by a linear programming (LP) solver. Because LP solvers are efficient and numerically stable, we treat it as part of the environment and restrict the slot configuration to center coordinates:

$$x = (c_1, \ldots, c_K), \qquad c_i = (u_i,\, v_i) \in [0, 1]^2.$$

Given x, the environment solves

$$r^*(x) \;=\; \arg\max_{r \geq 0} \; \sum_{i=1}^{K} r_i$$

under the linear constraints

$$r_i \leq u_i,\; r_i \leq 1{-}u_i,\; r_i \leq v_i,\; r_i \leq 1{-}v_i,\; r_i + r_j \leq \|c_i{-}c_j\|_2 \;\; \forall\, i{\neq}j,$$

and the objective is

$$F(x) = \sum_{i=1}^{K} r_i^*(x).$$

Slot configuration. Each slot stores a circle's center coordinates (u_i, v_i). Masked slots carry a draft of the same form, obtained by noising their pre-remask coordinates (see Appendix B.3). This anchors the inference trajectory near the parent configuration, localizing exploration around an elite configuration.

Embedding. Each slot is represented by the logit transform of its center coordinates $(u_i, v_i) \in [0, 1]^2$. A binary flag marks whether the slot is masked or part of the active core. The draft stored in a masked slot serves as the starting point of the denoising trajectory or is fed to the network as an extra input feature. We instantiate the conditional completion operator as a DDPM or FM model, whose output is a noise estimate or velocity field per slot in the same logit space. Both the DDPM denoising process and the FM ODE integration are carried out in logit space, and the resulting coordinates are mapped back to $[0, 1]^2$ at the end. During inference, active-core slots are clamped to their given values: any update to them is discarded after every DDPM reverse step or FM integration step, so only masked-slot coordinates evolve.

Table 1: Main comparison on No-Three-in-Line at $n \in \{15, 20, 25\}$. Best score (single seed); bold marks the best method per row.

n	$2n$	Random+Sat	Ramanathan	Ours
15	30	28	29	**30**
20	40	36	38	**39**
25	50	44	46	**49**

Decoding schedule $\mathcal{D}$. At each round, $\mathcal{D}$ runs the DDPM or FM reverse process over the currently masked slots and then commits a random subset of them to the active core. The number of rounds and the per-round commit count scale with the problem size K.

Remasking policy π. $\pi(M)$ mixes uniform random masking with a local-block heuristic that masks the k circles nearest a randomly selected point. In both modes, the remask ratio is drawn from a fixed schedule. The mixing ratio and the schedule are given in Appendix B.2.

Training. We follow the standard DDPM and FM training and inference procedures, with MSE losses on their respective objectives. The only modification is that active-core slots have their target noise estimate or velocity zeroed, restricting the loss to masked slots and matching the inference-time clamp. Noising schedule, denoising step counts and integration schedules for FM are reported in the next section.

5 Experiments

5.1 No-Three-in-Line

Baselines. **Random + Saturation** greedily adds feasible points selected uniformly at random to an empty grid until none remain. **Ramanathan et al.** [Ramanathan et al., 2025] apply the original PatternBoost algorithm with an autoregressive Transformer over a vocabulary of n^2 grid points. Table 1 reports our cross-method comparison at $n \in \{15, 20, 25\}$.

Scaling. We evaluate on grid sizes $10 \leq n \leq 20$ and $n \in \{25, 30, 35, 45\}$ (Table 2). With the model size of Ramanathan et al. [Ramanathan et al., 2025] (d64/L4/H4 backbone) and a smaller search budget ($P = 4096$, $G = 30$ generations, $C = 8192$ candidates per generation, totalling 245K candidates; $C = 16384$ at $n = 16$), our method recovers the optimal $2n$-point configuration for $n \leq 16$ but only reaches $2n - 1$ at $n = 17$. Scaling up the backbone and search budget for $n \geq 17$ closes the gap at $n = 17, 18$ and continues to produce strong constructions all the way up to $n = 45$. This indicates that our framework retains its utility for larger problem sizes. Full per-n configurations are in Appendix C.2.

Table 2: Scaling on No-Three-in-Line. "First gen" is the generation at which "Best" was first reached; "†" marks an incomplete run; "‡" marks an experiment conducted in two stages.

n	$2n$	Best	First gen	Total cand.
10	20	20	1	245K
11	22	22	2	245K
12	24	24	4	245K
13	26	26	5	245K
14	28	28	9	245K
15	30	30	14	245K
16	32	32	15	491K
17	34	34	19	600K
18	36	36	26	245K
19	38	37	16	245K
20	40	39	27	197K
25	50	$49^{\dagger\ddagger}$	$74^{\dagger\ddagger}$	786K
30	60	58	74	786K
35	70	66	23	800K
45	90	$82^{\dagger}$	$13^{\dagger}$	1000K

Ablation. We isolate three design choices: slot vs. sequence representation, active-core conditioning (AC), and iterative vs. one-shot decoding. AC affects both training and inference: training conditions on a feasible preserved core rather than a fully masked input, and inference clamps active-core slots throughout decoding. Iterative vs. one-shot decoding is purely an inference-time choice over the same trained model: one-shot predicts all masked slots in a single forward pass, whereas iterative commits a subset per round and re-runs the model on the shrinking masked set. Table 3 fixes $n = 25$, backbone ($d = 256$, $L = 6$, $H = 8$) and PatternBoost budget ($P = 4096$, $G = 64$, $C = 4096$). The five rows pair two baselines with a

Table 3: Ablation on No-Three-in-Line at $n = 25$. Best and Mean pool are mean $\pm$ std over 3 random seeds. Among the 3 seeds, only those attaining the row's highest Best are averaged for "#@best" (final-pool entries at that score, out of 4096) and "First gen" (generation first reaching it). First gen of Ramanathan et al. not directly comparable due to different initiation strategy.

Method	Best	Mean pool	#@best	First gen
Random + Saturation	44.00 ± 0.00	41.02 ± 0.00	1.0	30.0
Ramanathan et al. [Ramanathan et al., 2025] (our re-run)	46.33 ± 0.58	43.86 ± 0.01	1.0	—
Slot + one-shot (no active core)	44.67 ± 0.58	42.34 ± 0.01	3.5	21.0
Slot + active core + one-shot decode	47.67 ± 0.58	46.19 ± 0.15	38.5	29.0
Slot + active core + iterative (Ours)	$\mathbf{48.00 \pm 0.00}$	$\mathbf{46.93 \pm 0.15}$	**54.0**	**28.7**

three-row Slot progression that successively adds AC and then iterative decoding: Slot alone, Slot + AC + one-shot, and Slot + AC + iterative.

Findings. AC is the main driver of performance: adding it (Row 3 vs. Row 4) lifts Best by 3.00 and Mean pool by 3.85. Slot representation alone (Row 2 vs. Row 3) does not outperform the sequence baseline, indicating the gain comes from AC rather than representation. Iterative decoding (Row 4 vs. Row 5) adds a modest further gain.

5.2 Circle Packing in a Square

Setup. We evaluate on $K \in \{15, 26, 30, 32, 40\}$ with backbone $d = 128$, $G = 40$ generations, $C = 2048$ candidates per generation, running both DDPM and FM variants. DDPM is trained with $T = 64$ diffusion steps and inferred with 32 uniformly sub-sampled steps. We set $\beta_{\text{start}} = 5 \times 10^{-5}$ and $\beta_{\text{end}} = 0.04$ with a linear noising schedule. FM integrates the velocity ODE in 64 steps. Table 4 reports the main results across K.

An optional local search. FlowBoost [Bérczi et al., 2026] pairs its generative model with a local-search step called Stochastic Relaxation with Perturbations (SRP), which perturbs circle centers and applies gradient descent. Both their paper and our experiment show that SRP alone pushes solutions to near-optimal within the first few generations, thus including it would mask the contribution of the learned proposals. We therefore report main results and ablations without SRP, and defer the head-to-head comparison with FlowBoost (with SRP) to Appendix C.3.

Table 4: Main results on circle packing in a square across K without SRP. Mean $\pm$ std over 3 random seeds. "Pool p90/mean" over the final elite pool. "Last mean" is the mean candidate score in the final generation, capturing raw model proposal quality.

K	Method	Best	Pool p90	Pool mean	Last mean
15	Random + LP	$1.478 \pm .015$	$1.311 \pm .002$	$1.257 \pm .001$	$0.957 \pm .003$
15	DDPM	$\mathbf{1.846 \pm .029}$	$1.833 \pm .029$	$1.826 \pm .029$	$1.768 \pm .026$
15	FM	$1.812 \pm .013$	$1.790 \pm .013$	$1.773 \pm .013$	$1.631 \pm .010$
26	Random + LP	$1.840 \pm .005$	$1.675 \pm .001$	$1.619 \pm .001$	$1.315 \pm .003$
26	DDPM	$\mathbf{2.370 \pm .043}$	$2.341 \pm .044$	$2.325 \pm .044$	$2.209 \pm .071$
26	FM	$2.323 \pm .014$	$2.271 \pm .011$	$2.242 \pm .012$	$2.031 \pm .020$
30	Random + LP	$1.945 \pm .034$	$1.788 \pm .003$	$1.732 \pm .002$	$1.425 \pm .001$
30	DDPM	$\mathbf{2.473 \pm .045}$	$2.440 \pm .042$	$2.420 \pm .042$	$2.267 \pm .043$
30	FM	$2.426 \pm .012$	$2.371 \pm .016$	$2.341 \pm .020$	$2.116 \pm .015$
32	Random + LP	$2.012 \pm .013$	$1.843 \pm .001$	$1.785 \pm .001$	$1.477 \pm .003$
32	DDPM	$\mathbf{2.604 \pm .026}$	$2.566 \pm .025$	$2.546 \pm .026$	$2.393 \pm .036$
32	FM	$2.546 \pm .034$	$2.492 \pm .037$	$2.462 \pm .038$	$2.224 \pm .006$
40	Random + LP	$2.267 \pm .049$	$2.041 \pm .007$	$1.983 \pm .002$	$1.670 \pm .004$
40	DDPM	$\mathbf{2.833 \pm .005}$	$2.788 \pm .018$	$2.766 \pm .017$	$2.560 \pm .013$
40	FM	$2.697 \pm .028$	$2.630 \pm .019$	$2.596 \pm .016$	$2.327 \pm .022$

Table 5: Ablation on circle packing in a square at $K = 26$. Backbone $d = 128$. "no AC" denotes full-mask one-shot inference (no active core preserved). Mean $\pm$ std over 3 random seeds.

Method	Best	Pool p90	Pool mean	Last mean
Random + LP	$1.840 \pm .005$	$1.675 \pm .001$	$1.619 \pm .001$	$1.315 \pm .003$
DDPM, no AC (full-mask one-shot)	$2.090 \pm .030$	$1.995 \pm .018$	$1.960 \pm .017$	$1.795 \pm .024$
DDPM, AC + one-shot decode	$2.369 \pm .047$	$2.343 \pm .049$	$2.329 \pm .049$	$2.241 \pm .054$
DDPM, AC + iterative (Ours)	$2.370 \pm .043$	$2.341 \pm .044$	$2.325 \pm .044$	$2.209 \pm .071$
FM, no AC (full-mask one-shot)	$1.757 \pm .007$	$1.610 \pm .013$	$1.555 \pm .012$	$1.280 \pm .009$
FM, AC + one-shot decode	$2.318 \pm .023$	$2.268 \pm .023$	$2.242 \pm .022$	$2.052 \pm .025$
FM, AC + iterative (Ours)	$2.323 \pm .014$	$2.271 \pm .011$	$2.242 \pm .012$	$2.031 \pm .020$

Baselines. **Random + LP** samples uniform random centers and applies LP solver for radii at a matched candidate budget.

Ablation. Table 5 fixes $K = 26$ and isolates active-core conditioning, decoding schedule, and the generative model family (DDPM vs. FM).

Findings. DDPM is the stronger learned method at every K; a plausible reason is that DDPM injects noise at every denoising step while FM is deterministic given its input, giving DDPM more inference-time exploration. Random + LP trails the learned methods substantially at every K. In the ablation, active-core conditioning is the dominant contributor for both DDPM and FM, but FM is roughly $2\times$ more sensitive: without AC, FM falls below even the Random + LP baseline, indicating that the velocity head is specialised to conditional completion and breaks under unconditional generation. Iterative versus one-shot decoding has only a marginal effect, within the seed-to-seed variation.

A potential caveat on mean-pool metrics. The metrics Pool mean and Last mean partly reflect inheritance from parent elites: under active-core conditioning at small mask ratios, a new candidate retains most of its parent's structure and is likely to score similarly. The effect is more pronounced in circle packing in a square because the search space is continuous: masked centers can be moved by negligible displacements and still yield a score nearly identical to the parent's. In practice, however, the average displacement from denoising start to output is around $0.03 \sim 0.07$ in our packing runs, which is not negligible at the K values tested. See Table 8. Moreover, our model has not completely converged by the last generation, suggesting that it does not merely imitate configurations in the elite pool. Finally, our headline conclusions rest on Best, which is unaffected. On the other hand, this inheritance dynamic perhaps mirrors actual mathematical exploration, where negligible or even trivial modifications to high-quality constructions accumulate into new records.

6 Limitations

Distance to classical methods. Our No-Three-in-Line results remain far behind specialized solvers: a constraint-satisfaction approach exhibits $2n$-point configurations for all $n \leq 60$ [Prellberg, 2026].

Problem domains. We only examined two problems. Evidence for generalization beyond the two tested problem domains is limited.

Transferable learning. Evidence that the model learns transferable features is limited on No-Three-in-Line (see Appendix D.2).

7 Conclusion

We frame extremal structure discovery as *object-level active-core completion* and instantiate it as *Inpainting PatternBoost*, which replaces PatternBoost's complete-configuration generation with conditional reconstruction of remasked elite constructions. On No-Three-in-Line, this recovers the optimal $2n$-point configuration for n up to 18 and remains close to $2n$ at larger sizes tested up to $n = 45$. On maximum sum-of-radii circle packing, it matches FlowBoost when paired with the same

SRP polish and reaches an average of 88.5% of FlowBoost's reported scores without polish. These results support active-core completion as a more effective interface than unconditional generation for AI-assisted structure discovery in mathematics.

Future work. PatternBoost-style methods and program-evolution methods such as FunSearch [Romera-Paredes et al., 2024] and AlphaEvolve [Novikov et al., 2025] share the same template, differing only in their search space (*constructions* vs. *programs*). A mechanism for transferring knowledge learned in one space to the other could substantially advance AI-assisted mathematical discovery.

Acknowledgments and Disclosure of Funding

This work received no external funding; the computing resources for the experiments were paid for personally by the authors. The authors declare no competing interests.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Gergely Bérczi, Baran Hashemi, and Jonas Klüver. Flow-based extremal mathematical structure discovery. *arXiv preprint arXiv:2601.18005*, 2026.

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.

Lvda Chen, Yang Li, and Junchi Yan. MaskCO: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *International Conference on Learning Representations*, 2026.

Henry E. Dudeney. *Amusements in Mathematics*. Thomas Nelson and Sons, 1917.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022.

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Yichen Huang and Lin F. Yang. Winning gold at IMO 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.

Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753, 2019.

Yaron Lipman, Ricky T.Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11461–11471, 2022.

Abbas Mehrabian, Ankit Anand, Hyunjik Kim, Nicolas Sonnerat, Matej Balog, Gheorghe Comanici, Tudor Berariu, Andrew Lee, Anian Ruoss, Anna Bulanova, Daniel Toyama, Sam Blackwell, Bernardino Romera-Paredes, Petar Veličković, Laurent Orseau, Joonkyung Lee, Anurag Murty Naredla, Doina Precup, and Adam Zsolt Wagner. Finding increasingly large extremal graphs with AlphaZero and tabu search. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*, pages 6985–6993, 2024.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Thomas Prellberg. Constraint satisfaction programming for the no-three-in-line problem. *arXiv preprint arXiv:2602.07751*, 2026.

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, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431, 1998.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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

Adam Zsolt Wagner. Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

A Example constructions

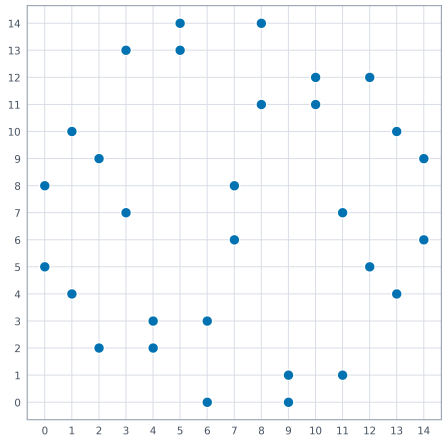


Figure 2: An optimal 30-point configuration on the 15×15 grid (no three collinear) found by our framework.

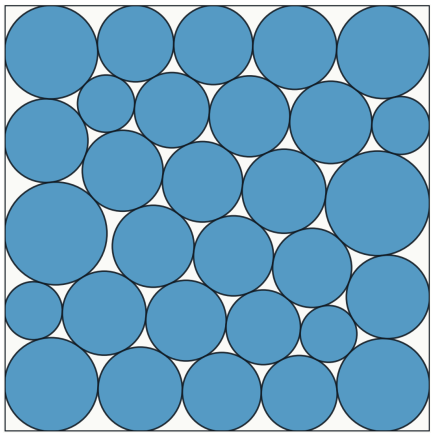


Figure 3: A near optimal packing of 30 circles in the unit square produced by our framework, with sum of radii 2.8424.

B Algorithm Details

B.1 Elite-pool sampling

In both the training and generation phases of Algorithm 1, samples are drawn from the elite pool $\mathcal{P}$ via a fixed mixture: with probability 0.9 we sample from the top-50 entries of $\mathcal{P}$ with probability proportional to score, and with probability 0.1 we sample uniformly at random from $\mathcal{P}$. The same schedule is used for both phases.

B.2 Remasking policy $\pi(M)$

At each generation step the mask M is drawn from a mixture of two policies sampled at a fixed ratio α: with probability α a random mask is sampled, and with probability $1-\alpha$ the domain-specific heuristic below is used. The mask-ratio schedule is essentially the same for both strategies.

No-Three-in-Line degree-of-freedom heuristic. The degree of freedom d_{ij} of slot (i, j) is the number of row values $y \in \{0, \ldots, n-1\}$ that are legally assignable given the current active core without this slot. The heuristic deterministically selects the slots with the largest d_{ij}, favouring columns that remain flexible. The mask ratio ρ and the mixing ratio α follow a two-stage schedule. In generations 1–$(G-1)$, ρ is sampled from $\{0, 0.2, 0.4, 0.6, 0.8\}$ with weights 1:2:2:1:1, and the mixing ratio is $\alpha = 0.3$. From generation G onward, ρ is sampled from $\{0.2, 0.4, 0.6\}$ with weights 4:2:1, and the mixing ratio becomes $\alpha = 0.1$. The switch generation G grows with n.

Circle packing local-block heuristic. A query point q is uniformly sampled from $[0, 1]^2$, and the mask M consists of the $k = \lceil \rho K \rceil$ circle slots whose current centers are nearest to q in Euclidean distance. In both the uniform and local-block modes, ρ is sampled from $\{0.1, 0.2, 0.4, 0.6, 1.0\}$ with weights 6:6:4:3:1, and the mixing ratio is $\alpha = 0.4$.

B.3 Circle packing draft construction

For each masked circle slot, we first construct a *draft* as the pre-mask center coordinates plus isotropic Gaussian noise; we use $\sigma = 0.05$ for FM and $\sigma = 0.02$ for DDPM. The draft acts as the per-slot input feature for both methods. For FM, the draft itself is the starting point of the reverse-time integration. For DDPM, the reverse process instead starts from

$$x_{\text{init}} = \sqrt{\alpha_{\text{eff}}}\, x_{\text{draft}} + \sqrt{1 - \alpha_{\text{eff}}}\, \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$, x_{draft} is the (logit-space) draft, and $\alpha_{\text{eff}} = s^2 \cdot \bar{\alpha}_t$ with $\bar{\alpha}_t$ the standard DDPM cumulative-noise schedule. We fix the draft strength $s = 0.7$.

B.4 Iterative decoding for No-Three-in-Line

The Decoding schedule $\mathcal{D}$ in Section 4.1 ranks masked slots by a weighted combination of two scores. Options for the model-side *confidence* score include the top-1/top-2 probability margin $p^{(1)}_{i,j} - p^{(2)}_{i,j}$, the negative `[MASK]` probability $1 - p_{i,j}(\texttt{[MASK]})$, and the negative Shannon entropy of $p_{i,j} = \mathrm{softmax}(\ell_{i,j})$. In our experiments we use the top-1/top-2 margin. The environment-side *legal* score is $1/d_{i,j}$, with $d_{i,j}$ the number of labels currently legal at slot (i, j). The ranking score is fixed throughout all experiments to margin $+ 0.15 \cdot$ legal.

C Experiment Details

C.1 Training hyperparameters

All runs use the AdamW optimizer with learning rate 5×10^{-4} and batch size 32. Other training-budget settings (training samples per generation, generations G, candidates per generation C, epochs per generation) are stated elsewhere in the paper. The experiment is run on a single NVIDIA RTX 5090 GPU and Intel 16-core CPU. A No-Three-in-Line run takes approximately 2–5 hours for $n < 20$ and 12–24 hours for $n \geq 20$; a circle-packing run takes approximately 1–2 hours.

C.2 Full No-Three-in-Line scaling data

Table 6 reports our method's results for $10 \leq n \leq 20$ and $n \in \{25, 30, 35, 45\}$. All entries are single-seed.

Table 6: Complete per-n results for our method on No-Three-in-Line. "Model" is given as $d/L/H$ (embedding dimension, number of Transformer layers, attention heads). "PB" abbreviates the PatternBoost configuration as $P/G/C/T$ (elite pool size, generations, candidates per generation, training samples per generation), with K denoting thousands. "First gen" is the generation index at which Best was first reached. "Final loss" is the training loss recorded at the last generation. All runs use 1 epoch per generation except $n = 45$ which uses 2. "†" marks an incomplete run. "‡" marks an experiment conducted in two stages.

n	$2n$	Best	Model	PB ($P/G/C/T$)	First gen	Final loss
10	20	20	64/4/4	4K/30/8K/8K	1	0.755
11	22	22	64/4/4	4K/30/8K/8K	2	0.788
12	24	24	64/4/4	4K/30/8K/8K	4	0.748
13	26	26	64/4/4	4K/30/8K/8K	5	0.780
14	28	28	64/4/4	4K/30/8K/8K	9	1.032
15	30	30	64/4/4	4K/30/8K/8K	14	0.880
16	32	32	64/4/4	4K/30/16K/16K	15	0.639
17	34	34	256/6/8	10K/30/20K/20K	19	0.367
18	36	36	256/8/8	4K/30/8K/8K	26	0.300
19	38	37	256/8/8	4K/30/8K/8K	16	0.258
20	40	39	256/6/8	4K/48/4K/8K	27	0.252
25	50	49†‡	256/6/8	8K/96/8K/8K	74†‡	0.072
30	60	58	320/6/8	8K/96/8K/16K	74	0.147
35	70	66	256/12/8	10K/40/20K/20K	23	0.129
45	90	82†	512/12/16	10K/50/20K/20K	13†	0.256

C.3 Circle packing with SRP local search

Table 7 reports +SRP scores at $K \in \{26, 30, 32\}$.[2] FlowBoost, DDPM+SRP, and FM+SRP converge to identical Best within numerical precision (matching the known optimal sum-of-radii values for

[2]The SRP local-search step and the LP solver used here are reused from the FlowBoost codebase [Bérczi et al., 2026].

these instances). Their Polish mean—the mean over SRP-polished candidates—also coincides within ~ 0.02, indicating that SRP brings all methods to near-Best regardless of the underlying generator.

Table 7: Effect of SRP local search on circle packing in a square (per-K). "Polish mean" is the mean score over candidates that pass through the SRP polishing step.

K	Method	Best	Polish mean
26	FlowBoost [Bérczi et al., 2026]	2.636	2.618
26	DDPM + SRP	2.636	2.633
26	FM + SRP	2.636	2.622
30	FlowBoost [Bérczi et al., 2026]	2.842	2.820
30	DDPM + SRP	2.842	2.839
30	FM + SRP	2.842	2.833
32	FlowBoost [Bérczi et al., 2026]	2.939	2.918
32	DDPM + SRP	2.939	2.934
32	FM + SRP	2.939	2.925

C.4 Displacement of masked centers and final training loss

For each generation, the metric `inference_center_distance_mean` records the mean Euclidean displacement of masked circle centers between the start and end of the denoising / flow-matching reverse process, averaged over all candidates of that generation. Table 8 reports the final-generation value of this metric alongside the final-generation training loss for each (method, K). K=26 DDPM uses the average over the two runtime-seed copies since the canonical run does not record the displacement metric; all other cells are single-run.

Table 8: Final-generation displacement and training loss per K on circle packing in a square under the no-SRP setting. "Displacement" is `inference_center_distance_mean` at the last generation (dimensionless distance in $[0, 1]^2$). "Loss" is `mean_train_loss` at the last generation.

K	DDPM displacement	DDPM loss	FM displacement	FM loss
15	0.031	0.088	0.044	0.144
26	0.038	0.103	0.044	0.203
30	0.043	0.115	0.043	0.199
32	0.043	0.126	0.069	0.177
40	0.048	0.122	0.049	0.380

D Additional experiments

D.1 Fixed-budget comparison across grid sizes for No-Three-in-Line

To separate the effect of active-core conditioning from model size and budget, we rerun the ablation of Table 3 at $n \in \{20, 25, 30\}$ with the backbone and search budget fixed across n (Table 9). For $n = 25$, values are read at generation 48 from runs configured for 64 generations.

Table 9: Best score on No-Three-in-Line at fixed backbone and budget across n, mean $\pm$ std over 3 seeds. Backbone $d = 256$, $L = 6$, $H = 8$; pool 4096, $C = 4096$, 48 generations.

Method	$n = 20$	$n = 25$	$n = 30$
Random + Saturation	36.00 ± 0.00	44.00 ± 0.00	52.00 ± 0.00
Ramanathan et al. [Ramanathan et al., 2025] (our re-run)	38.00 ± 0.00	46.33 ± 0.58	55.67 ± 0.58
Slot + one-shot (no active core)	36.67 ± 0.58	44.67 ± 0.58	53.00 ± 0.00
Slot + active core + one-shot decode	38.67 ± 0.58	47.67 ± 0.58	56.33 ± 0.58
Slot + active core + iterative (Ours)	$\mathbf{39.00 \pm 0.00}$	$\mathbf{48.00 \pm 0.00}$	$\mathbf{57.00 \pm 0.00}$

D.2 Transferable learning and random-operator controls

The experiments in this section test whether the model learns transferable features, rather than relying on local perturbation of preserved cores. On circle packing the evidence for transferable learning is clear; by comparison, it is limited on No-Three-in-Line.

Cross-model completion. For each domain, we first run the full search and record its elite pool, from which we sample elite parents. The *home model* is trained during this source run. The *independent model* is trained by an otherwise identical full run with a different random seed, and therefore on a different elite pool. For every sampled parent, we apply the same remask and complete the resulting partial construction with the home model, the independent model, or a random operator (Table 10). The random operator fills masked slots with random legal values on No-Three-in-Line, and adds Gaussian noise with $\sigma = 0.05$ to masked centers on circle packing, matches the average center displacement of the learned operators over all generations.

Table 10: Cross-model completion at $K = 26$ for circle packing and $n = 25$ for No-Three-in-Line. Entries are paired differences between child-minus-parent scores, with percentages relative to home − random in parentheses.

Domain	home − random	independent model − random	home − independent model
Packing	$+0.176$ (100%)	$+0.120$ (68.2%)	$+0.056$ (31.8%)
No-Three-in-Line	$+1.267$ (100%)	$+0.076$ (6.0%)	$+1.191$ (94.0%)

Full-loop random-operator control. We repeat the complete main experiment with the same settings and budgets. The only change is to replace the learned completion operator with the random operator (Tables 11 and 12).

Table 11: Packing $K = 26$ random-operator control, no SRP.

Completion operator	Best	Mean pool
Learned DDPM	2.370 ± 0.043	2.325 ± 0.044
Random perturb	2.290	2.196

Table 12: No-Three-in-Line random-operator controls. Random completion with single seed. For $n = 25$, values are read at generation 48 from runs configured for 64 generations.

n	Completion operator	Best	#@best	Mean pool	First gen
25	Learned iterative	48	22.7 ± 7.6	46.491 ± 0.153	28.3 ± 0.6
25	Random completion	48	4	46.048	39
30	Learned iterative	57	20.0 ± 22.5	55.340 ± 0.085	37.0 ± 3.6
30	Random completion	56	72	54.584	24

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our main contribution is to formulate extremal structure discovery as object-level active-core completion and instantiate it as Inpainting PatternBoost, validated on No-Three-in-Line and circle packing in a square. This matches the descriptions in the abstract and Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Section 6 and a potential caveat on the Pool mean and Last mean metrics in Section 5.2.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: the paper does not include theoretical results

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Experimental setup is described in Section 5 and Appendix C.1; per-task model and PatternBoost configurations are in Appendix C.2.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code and data are provided via an public GitHub repository.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We include the training and test details in Appendix and Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Tables 3, 4 and 5 report mean $\pm$ standard deviation over 3 random seeds; the scaling and SRP results are single-seed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources (GPU and CPU type) and per-run wall-clock times are reported in Appendix C.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirm that our research conforms to it in every respect.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper concerns AI assisted mathematical discovery and has little to no societal impact.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work poses no safety risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We reuse the SRP local-search and LP-solver components from the FlowBoost codebase [Bérczi et al., 2026] for the +SRP experiments in Appendix C.3; the original work is credited via citation and a footnote at the point of use. Other dependencies are standard open-source libraries (PyTorch, NumPy, SciPy).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Not applicable to our research.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Not applicable to our research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Not applicable to our research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The author used ChatGPT and Claude to assist with code implementation and debugging. The research ideas and algorithm/experiment design were developed by the author.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.