TIME	6 Mon – 7/14	7 Tue– 7/15	8 Wed– 7/16	9 Thu– 7/17	10 Fri – 7/18
9:30 - 10:45 AM	Integrated Discovery I	Visiting Lecturer (Oki)	Beyond Hypotheses Generation I	Beyond Hypotheses Generation III	Beyond Hypotheses Generation Applications
10:45 - 11:00 AM	Break	Break	Break	Break	Break
11:00 AM - 12:15 PM	Integrated Discovery II	Hands-on Interactive Lab Experimentation (magnets)	Beyond Hypotheses Generation II	Beyond Hypotheses Generation IV	Student Presentations
12:15 - 1:30 PM	Lunch	Lunch	**Yorktown BBQ**	Lunch	Lunch
1:30 - 2:45 PM	Integrated Discovery III	Yorktown Lab Tour	Beyond Hypotheses Generation Tutorial	Working groups	Student Presentations
2:45 - 3:00 PM	Break	Break	Break	Break	Break
3:00 - 4:30 PM	Tutorial	Working Groups	Working Groups	**MOC Reception**	Summary

Algorithm Generation and FunSearch

SLMath Summer School Lecture

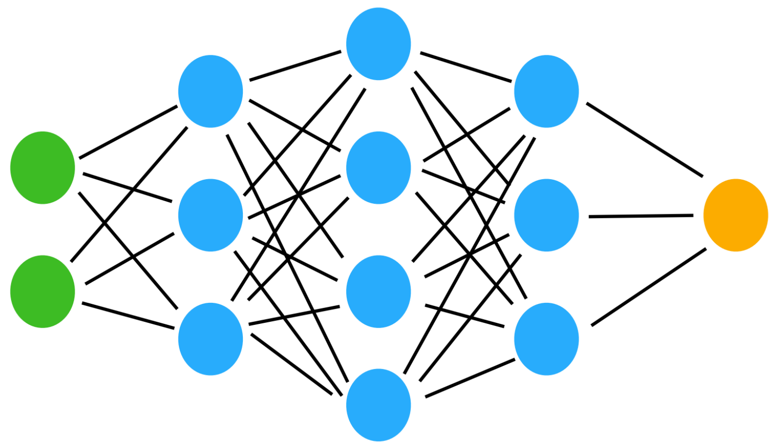
Karan Srivastava | IBM Research Intern | University of Wisconsin Madison

The story thus far

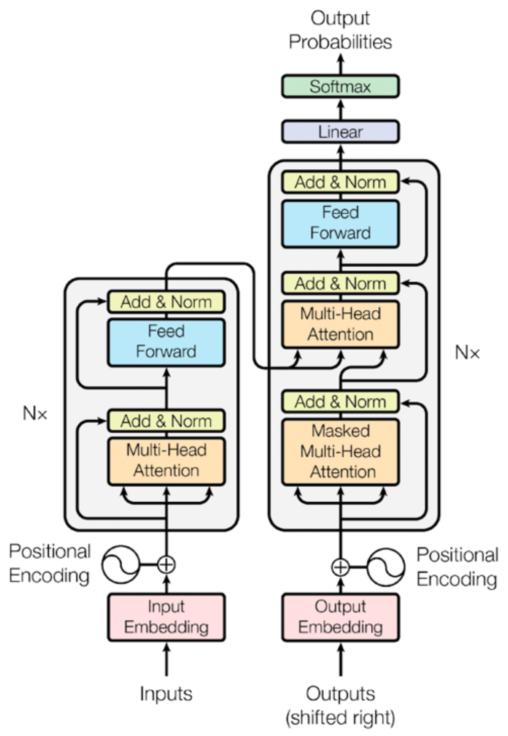
Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Traditional Neural Methods

Neural Networks



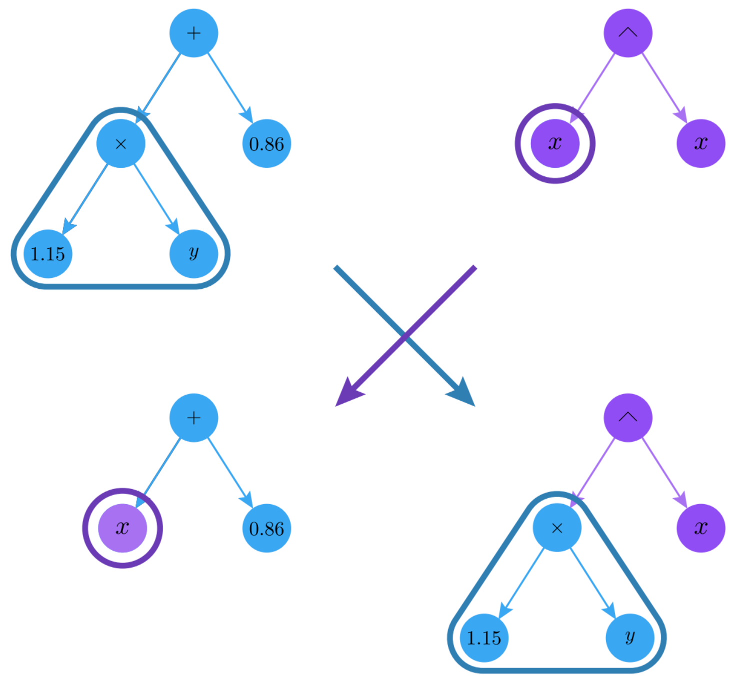
Transformers



Symbolic Regression

Genetic Approaches

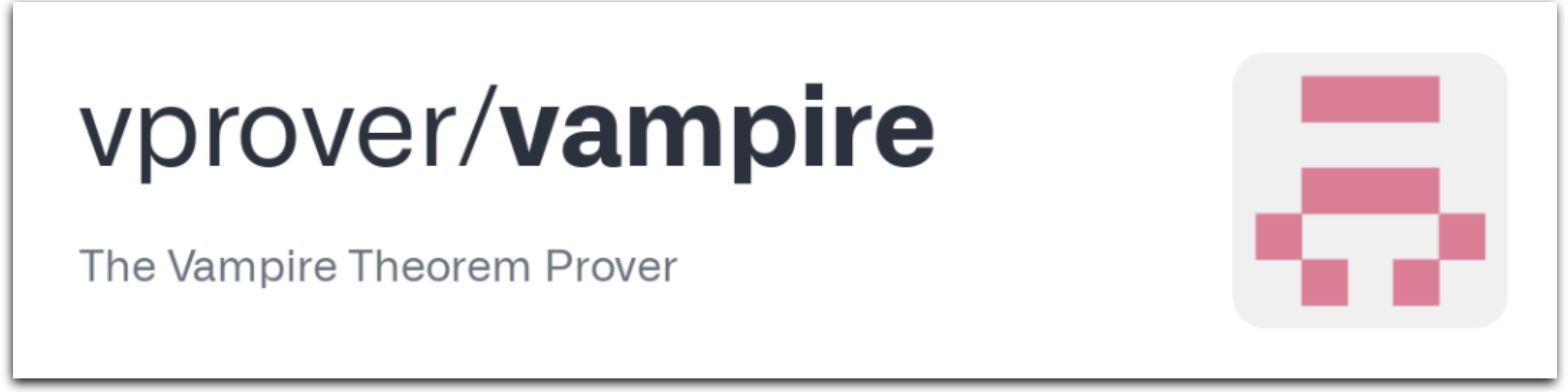
MINLP

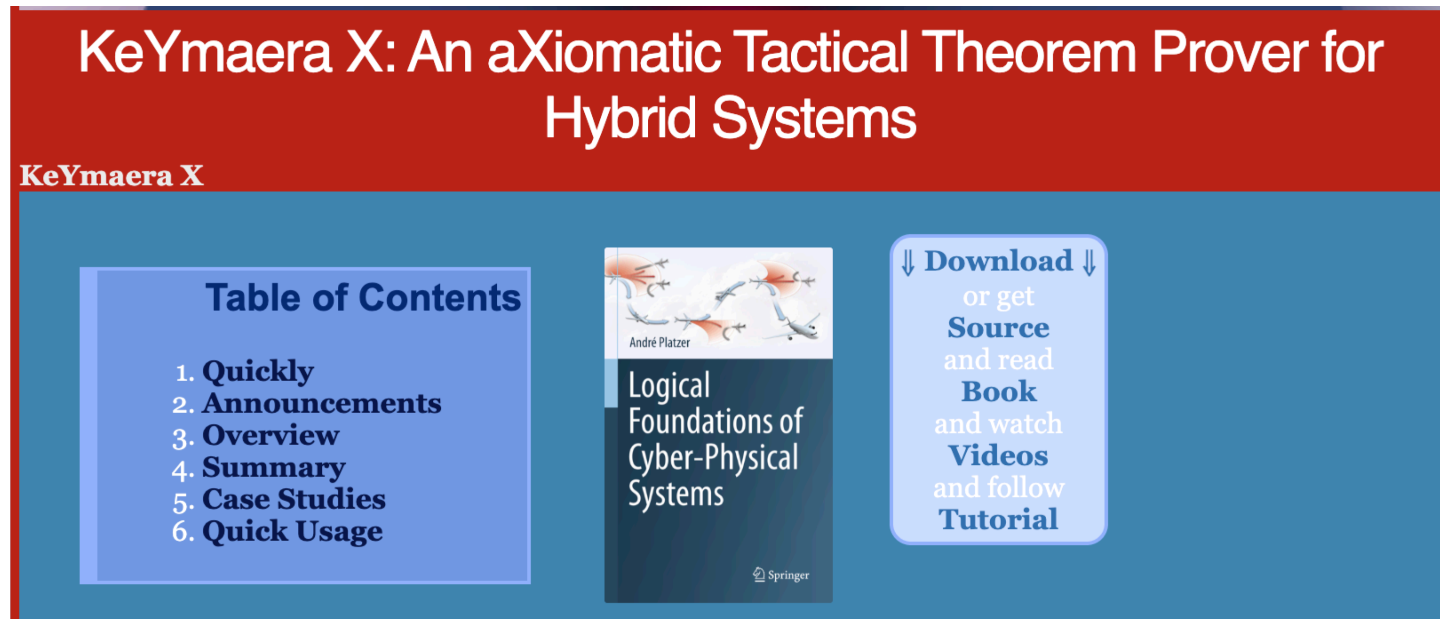


The story thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.



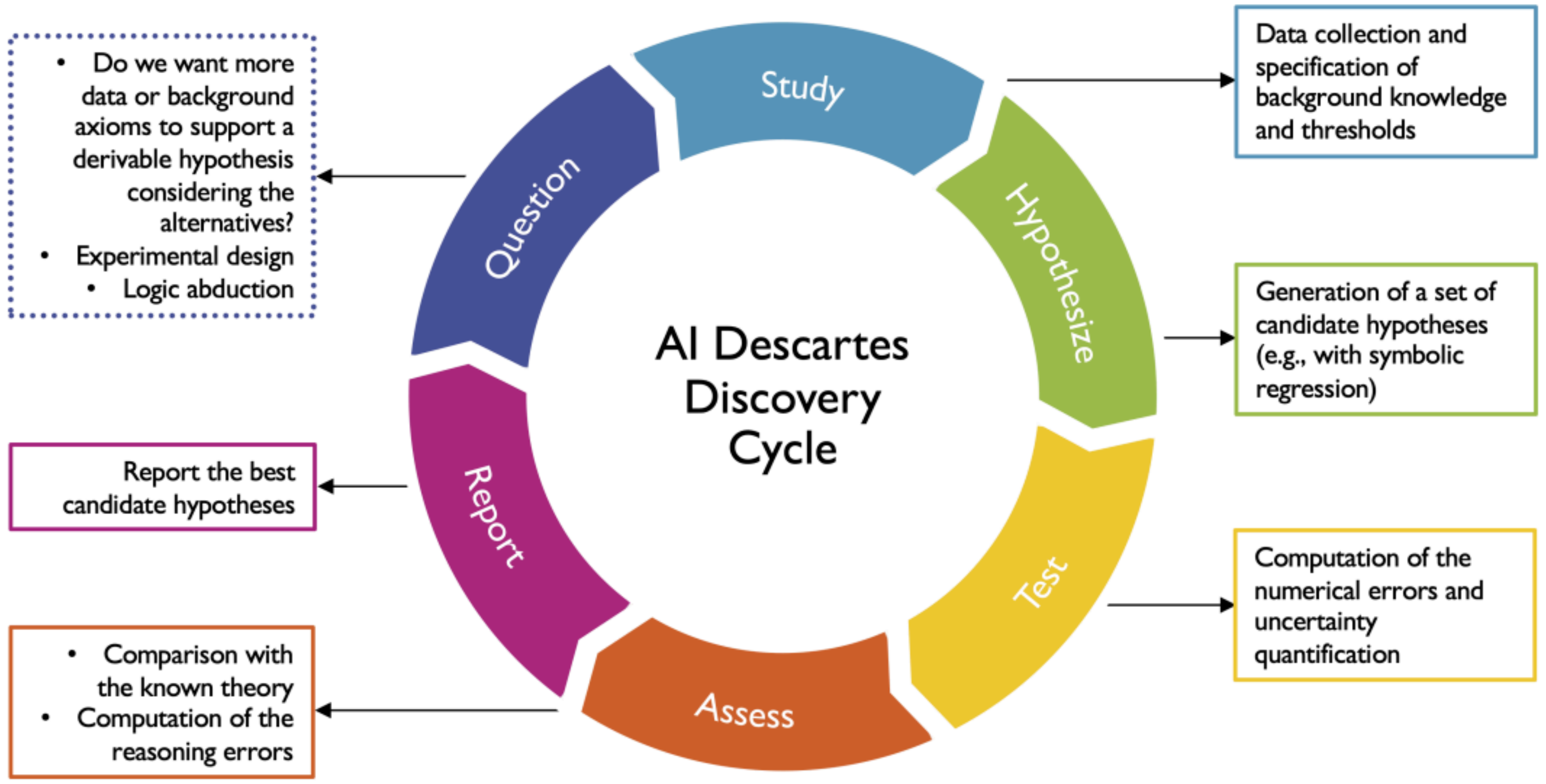


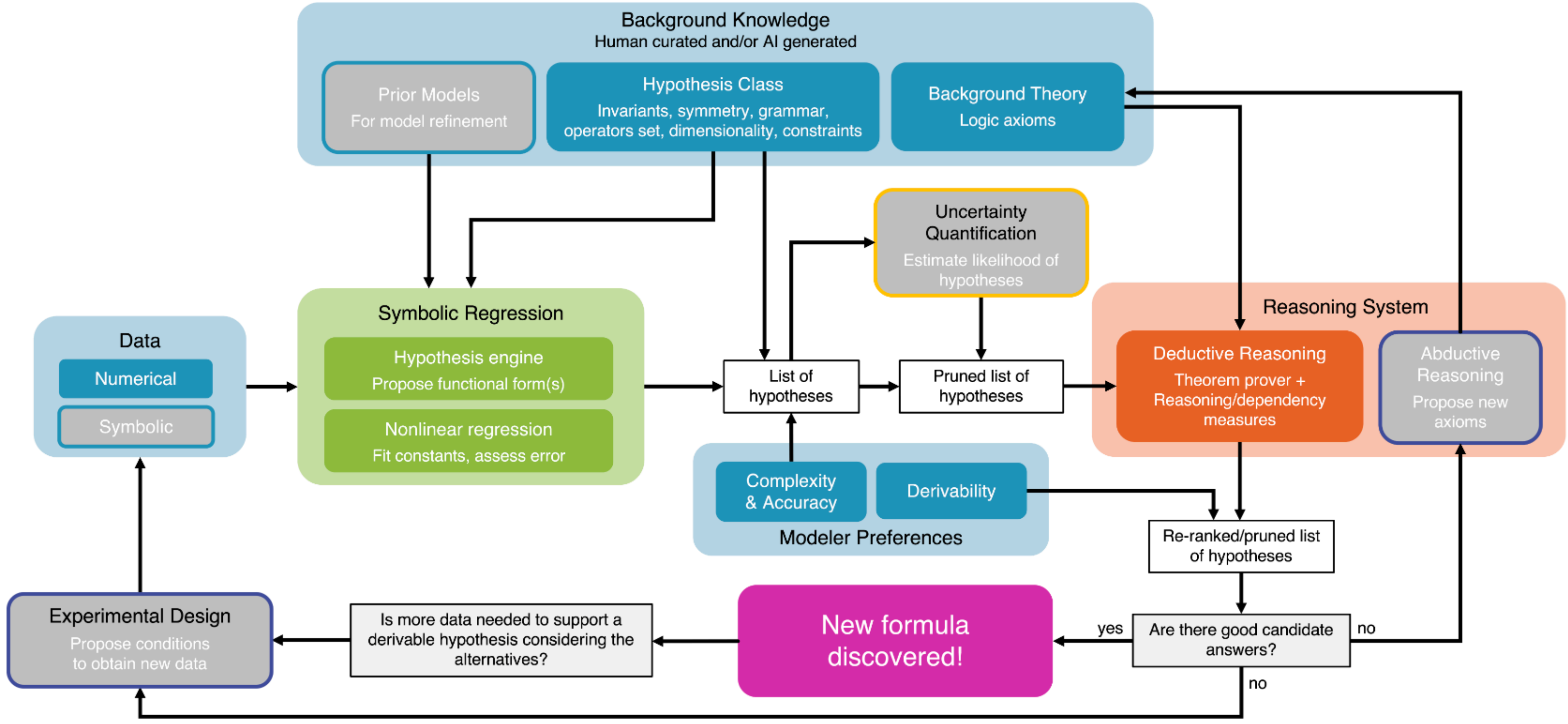
The story thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)





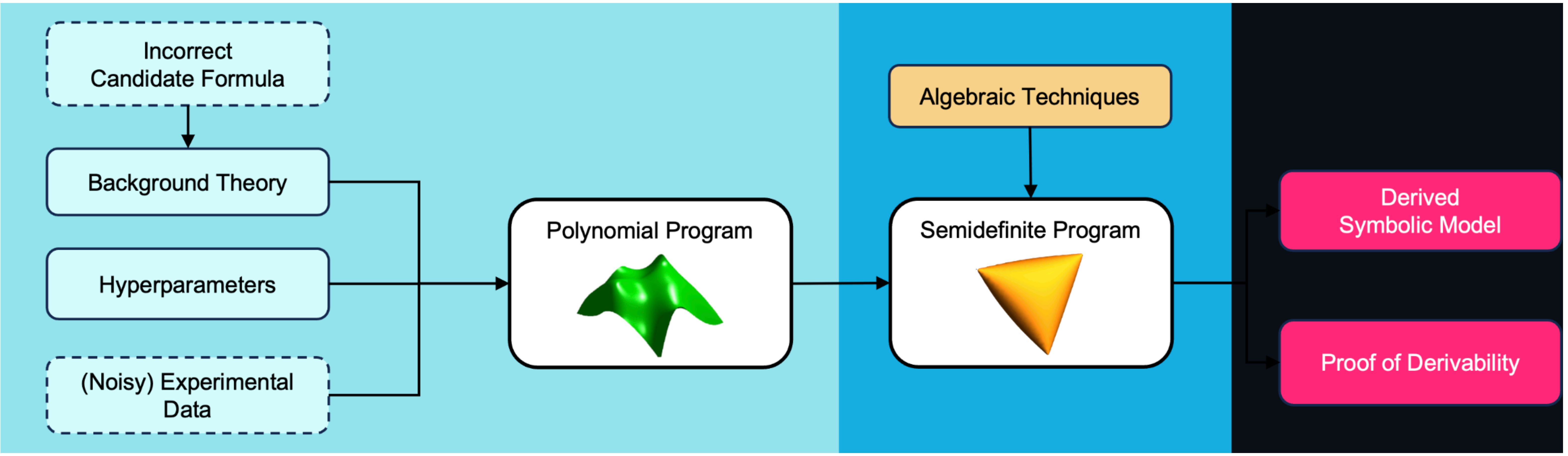
The story thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

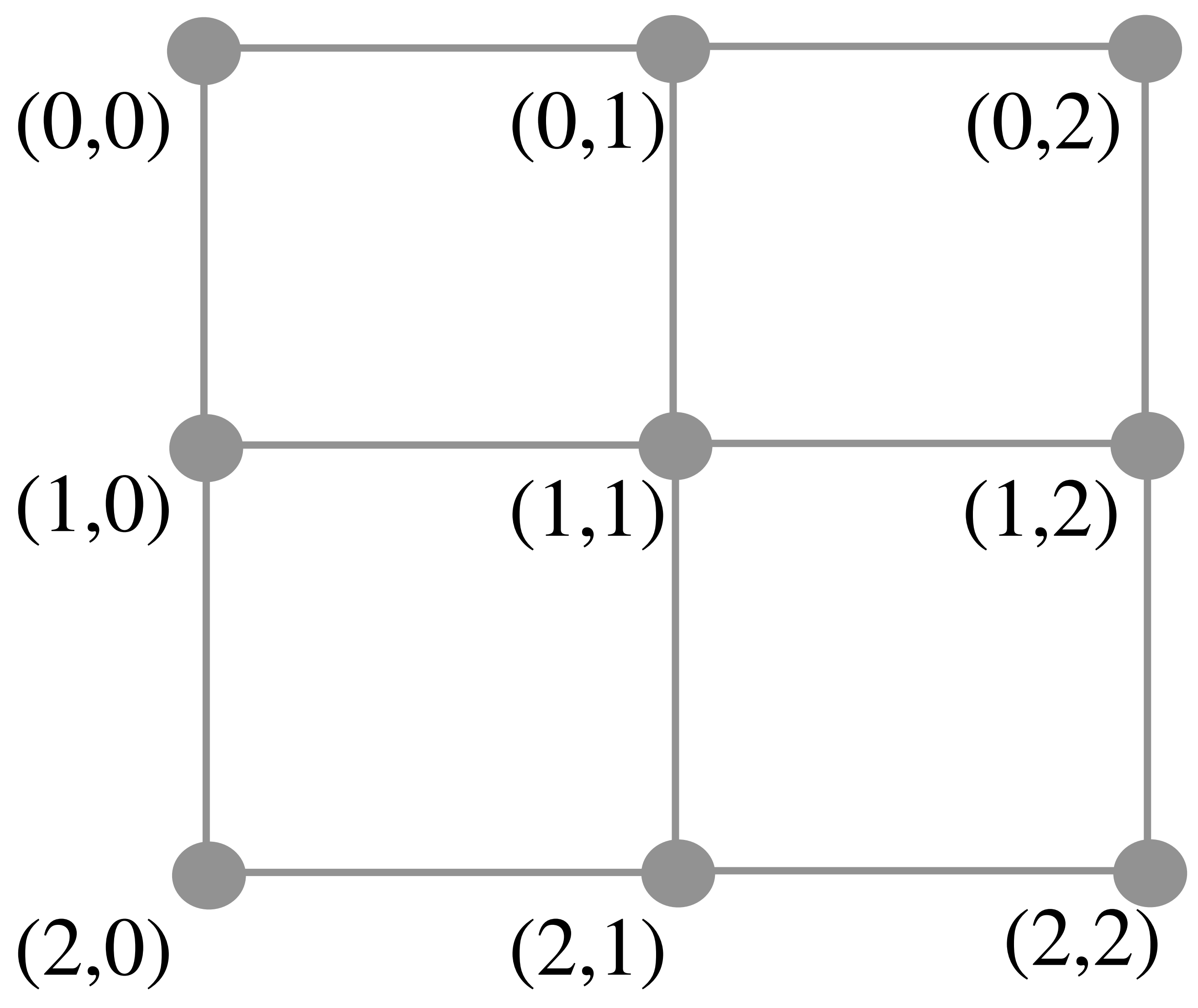
Data + Reasoning Integrated
Integrating both data and theory in the search process can improve search (AI Hilbert)



Some motivating problems for today

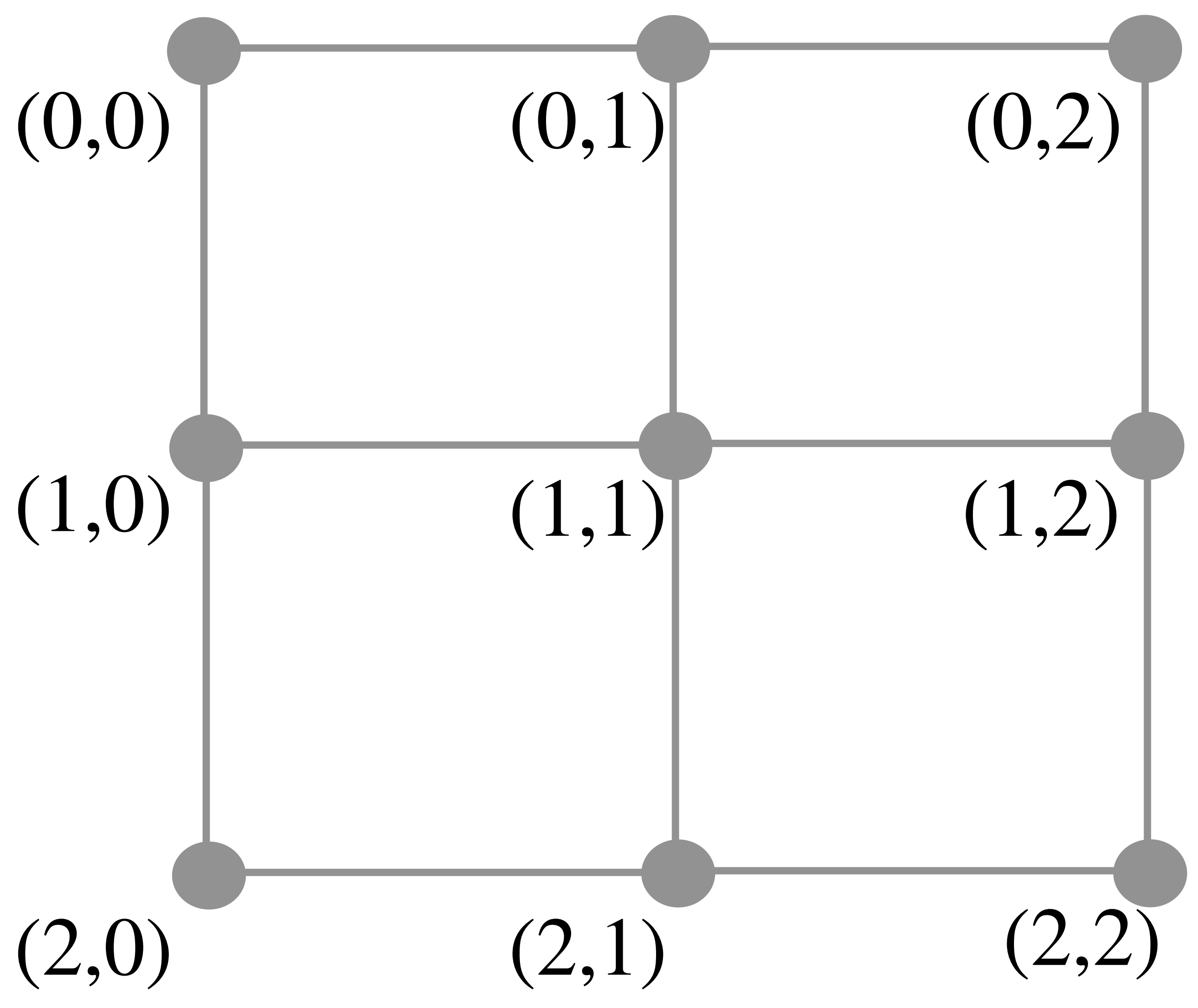
Motivating Example 1

We have a 3x3 integer lattice whose coordinates are in $\{0,1,2\} \times \{0,1,2\}$ or $\mathbb{Z}_3^2$. What's the largest subset of points on the lattice so that no three points sum up (coordinate-wise) to $(0,0)$ with addition modulo 3?



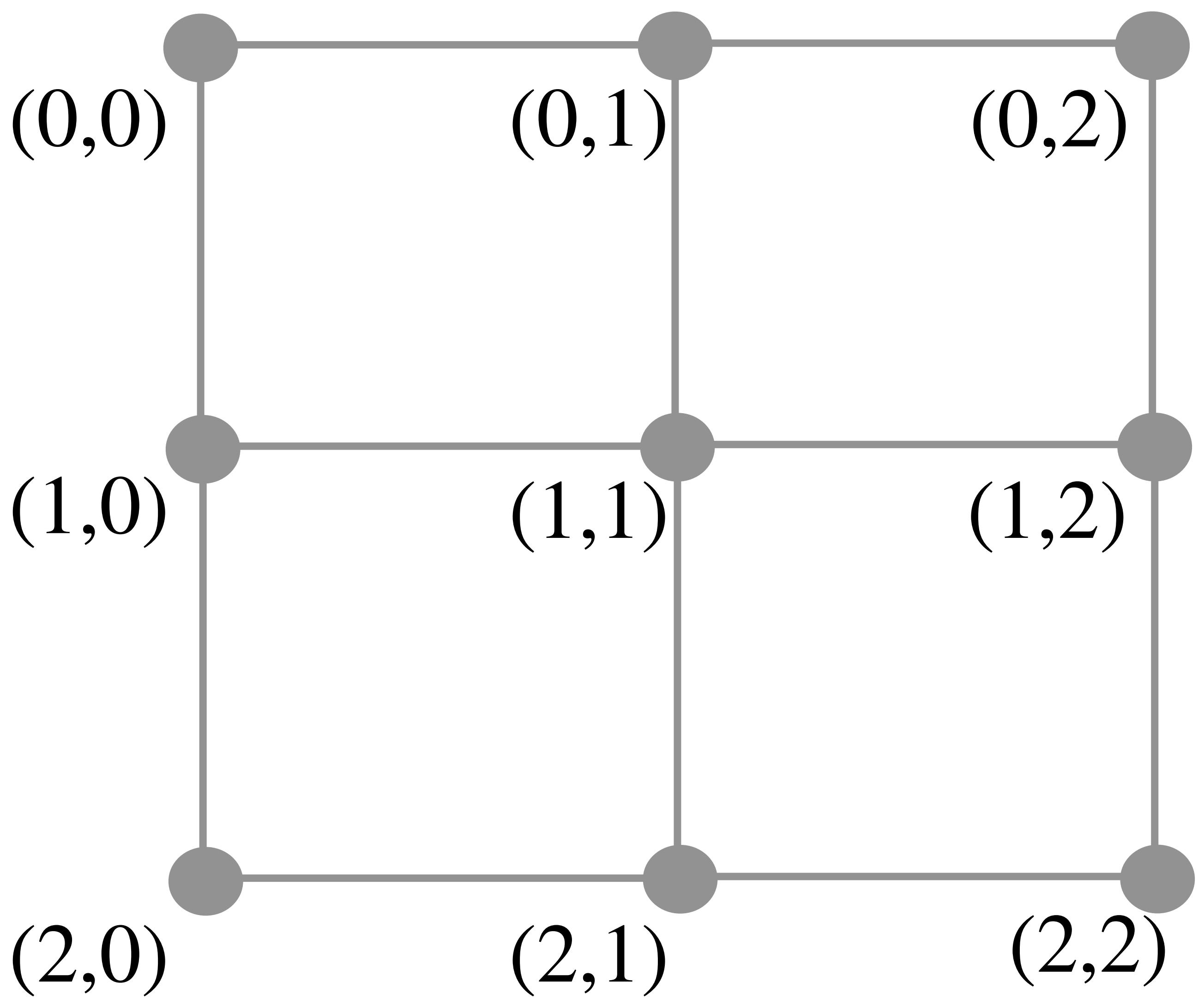
Motivating Example 1

We have a 3x3 integer lattice whose coordinates are in $\{0,1,2\} \times \{0,1,2\}$ or $\mathbb{Z}_3^2$. What's the largest subset of points on the lattice so that no three points sum up (coordinate-wise) to $(0,0)$ with addition modulo 3?



Motivating Example 1

We have a 3x3 integer lattice whose coordinates are in $\{0,1,2\} \times \{0,1,2\}$ or $\mathbb{Z}_3^2$. What's the largest subset of points on the lattice so that no three points sum up (coordinate-wise) to $(0,0)$ with addition modulo 3?

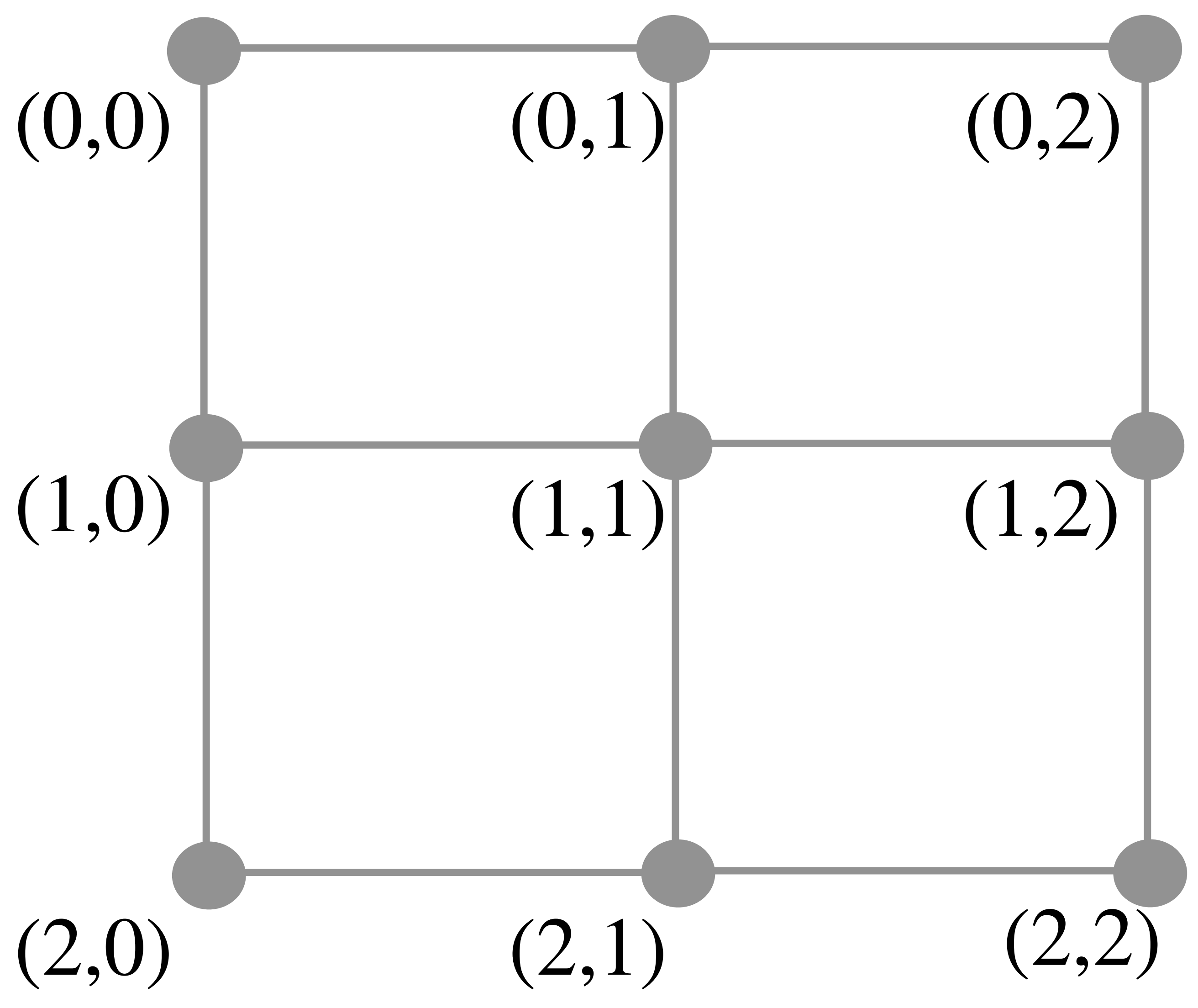


What's the largest subset of $\mathbb{Z}_3^d$ with no three points in a line?

"Perhaps my favorite open question" - Terrance Tao, 2007 blog post[1].

Motivating Example 1

We have a 3x3 integer lattice whose coordinates are in $\{0,1,2\} \times \{0,1,2\}$ or $\mathbb{Z}_3^2$. What's the largest subset of points on the lattice so that no three points sum up (coordinate-wise) to $(0,0)$ with addition modulo 3?

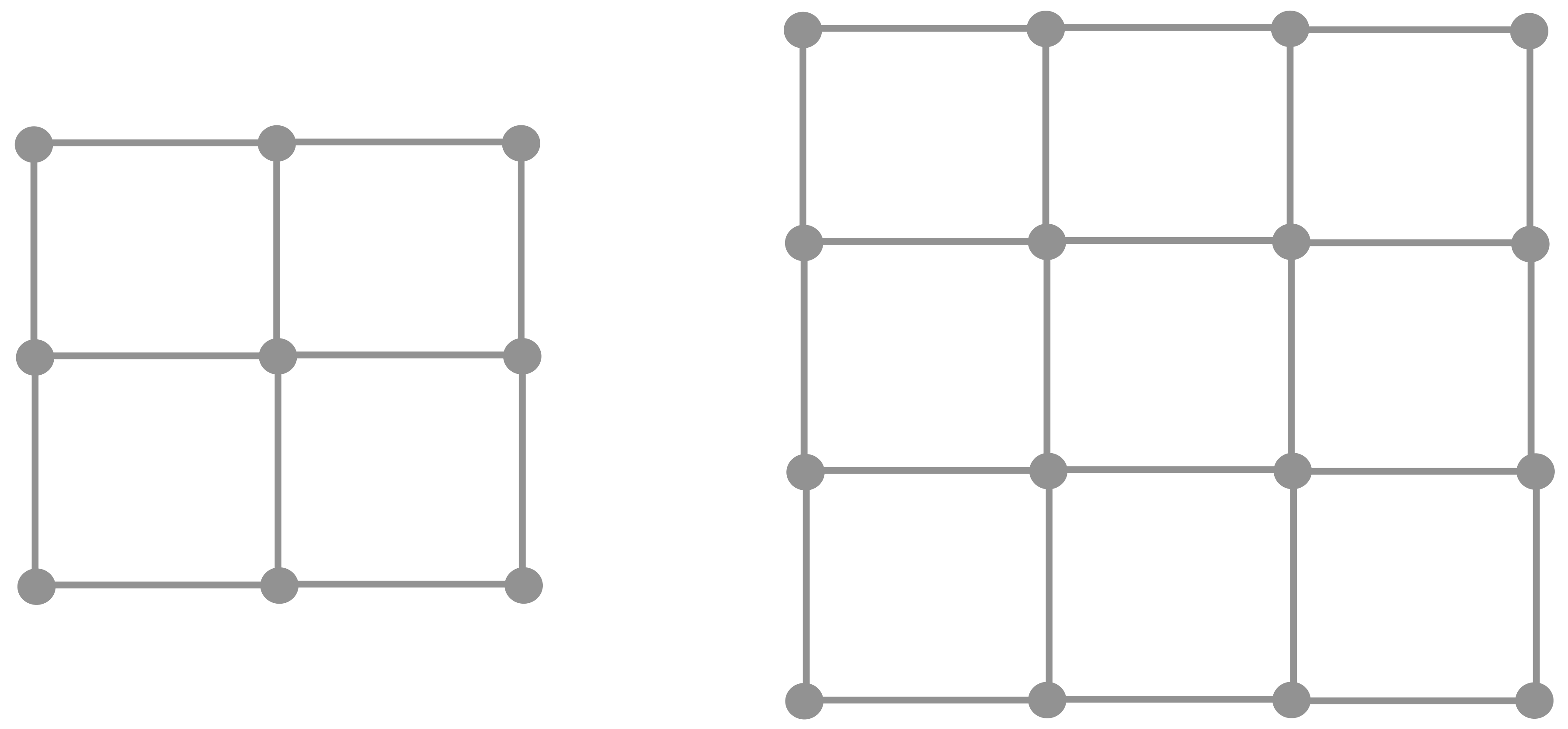


What's the largest subset of $\mathbb{Z}_3^d$ with no three points in a line? Largest sets known only till n = 6. Lower and upper bounds are 2.2202^d and 2.756^d.

"Perhaps my favorite open question" - Terrance Tao, 2007 blog post[1].

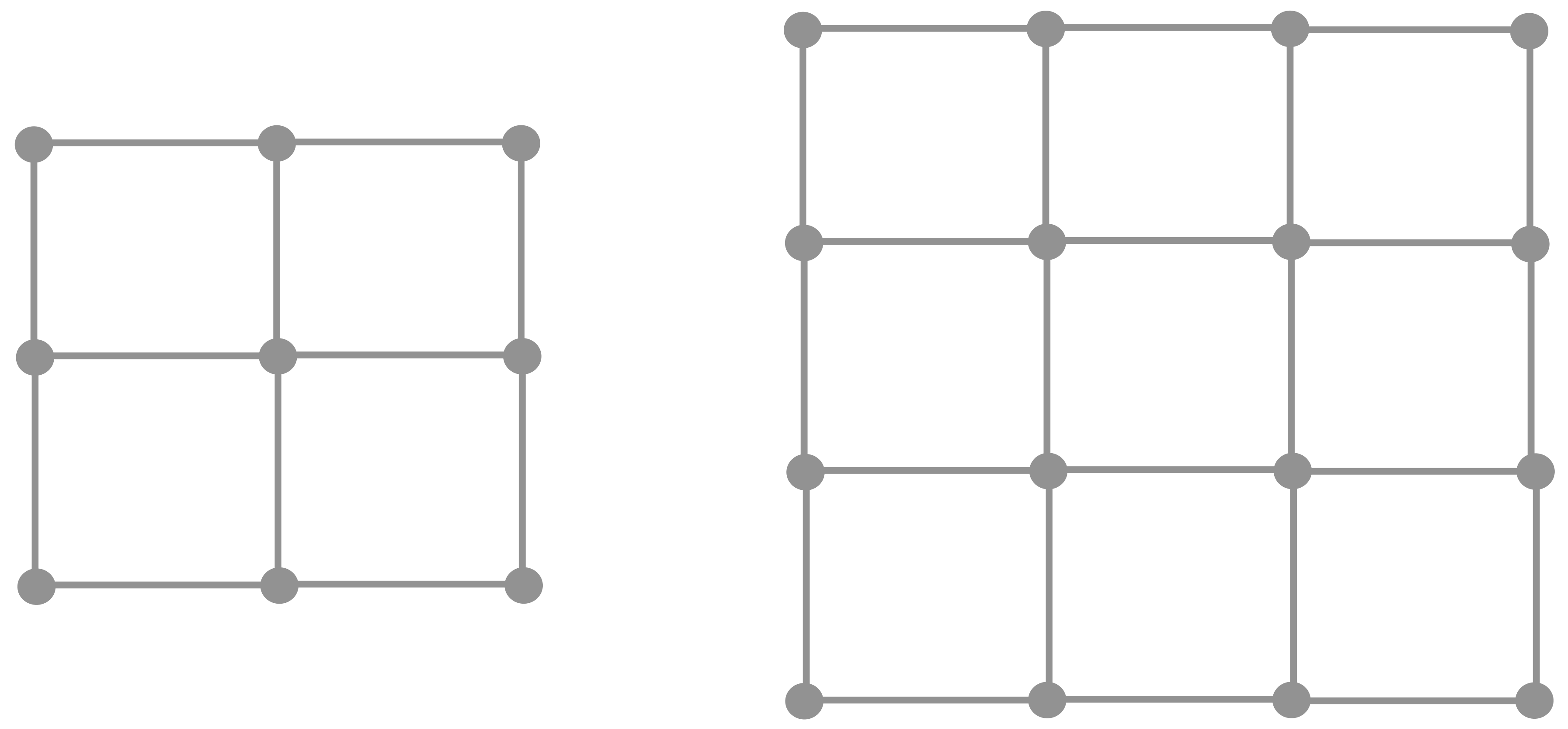
Motivating Example 2

What is the largest subset of 3x3 lattice such that no three points form an isosceles triangle?



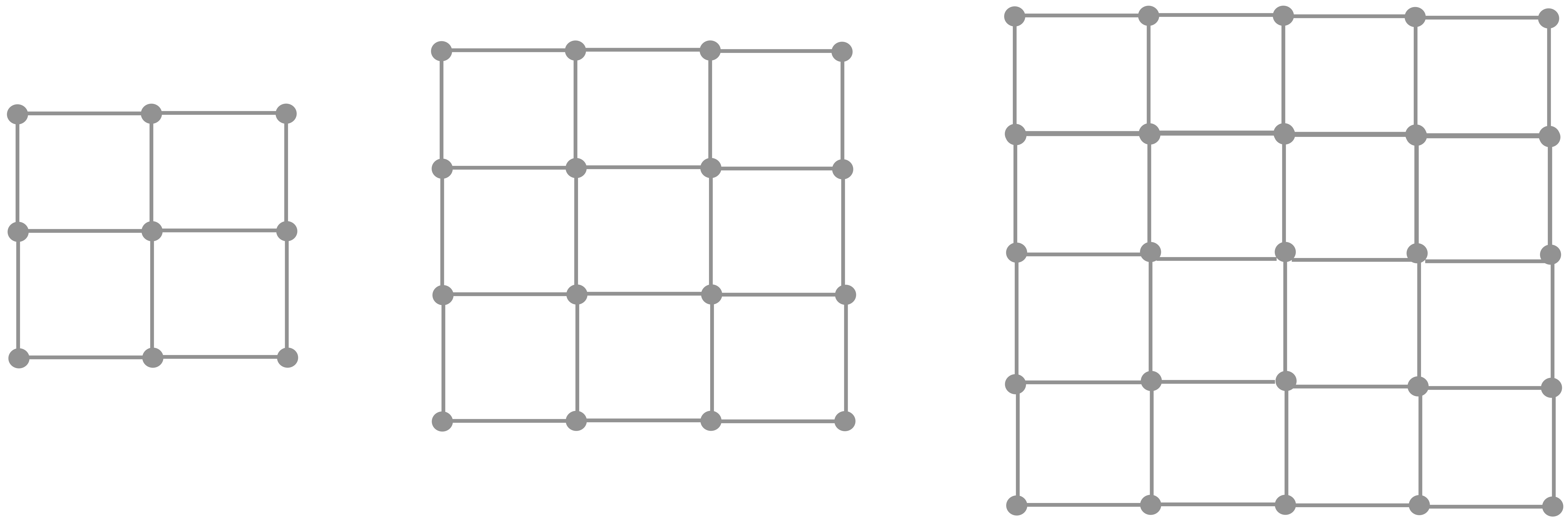
Motivating Example 2

What is the largest subset of 3x3 lattice such that no three points form an isosceles triangle?



Motivating Example 2

What is the largest subset of 3x3 lattice such that no three points form an isosceles triangle?

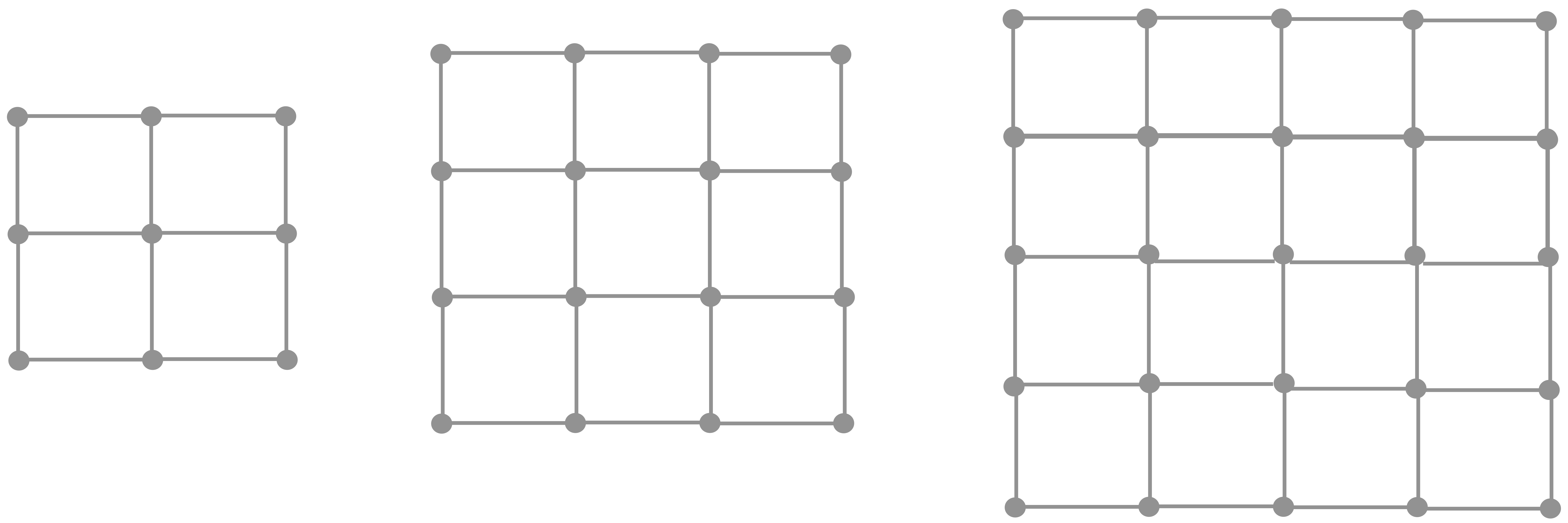


What's the size of the largest subset of an integer lattice with no 3 points forming an isosceles triangle?

"Karan, this would be a cool problem to think about" - Jordan Ellenberg, mathematician and PhD advisor.
- Problem originated from a study of convergence rates on ordinal embeddings[2].

Motivating Example 2

What is the largest subset of 3x3 lattice such that no three points form an isosceles triangle?

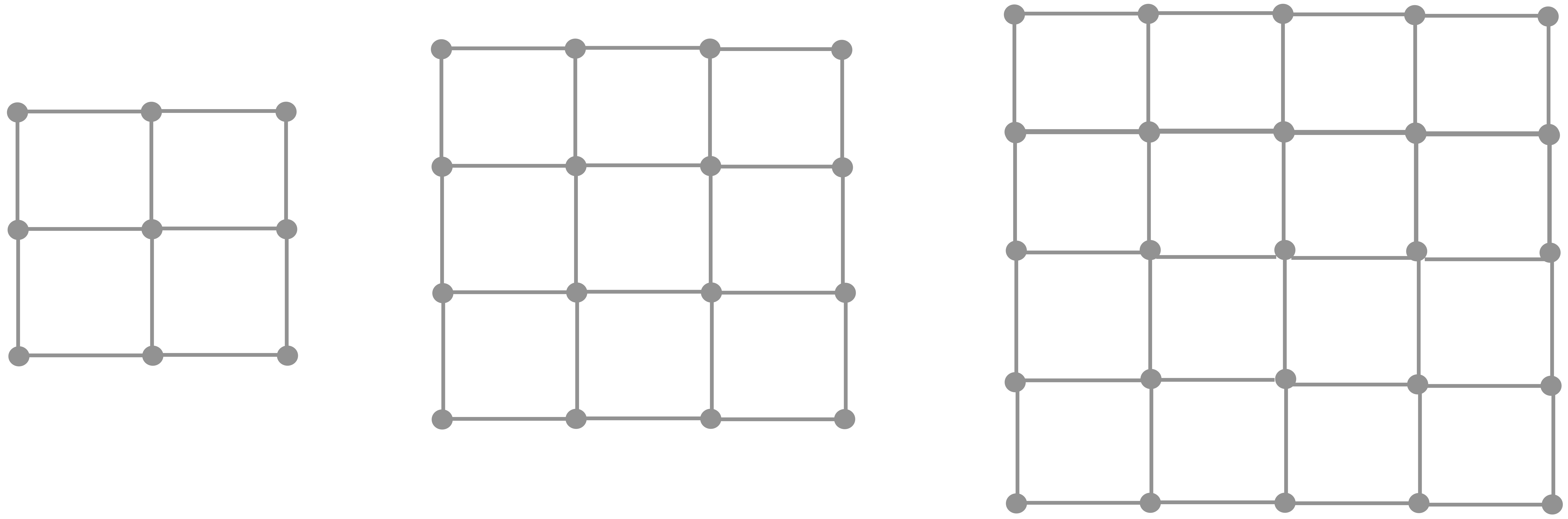


What's the size of the largest subset of an integer lattice with no 3 points forming an isosceles triangle? Largest sets known only till n=10.

"Karan, this would be a cool problem to think about" - Jordan Ellenberg, mathematician and PhD advisor.
- Problem originated from a study of convergence rates on ordinal embeddings[2].

Motivating Example 2

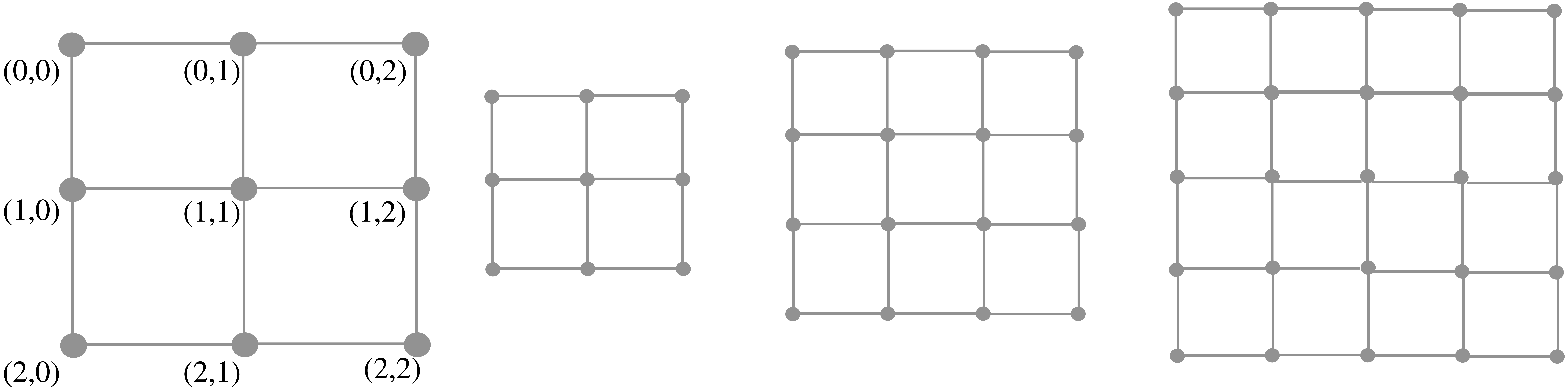
What is the largest subset of 3x3 lattice such that no three points form an isosceles triangle?



What’s the size of the largest subset of an integer lattice with no 3 points forming an isosceles triangle? Largest sets known only till n=10. Lower and upper bounds are still far apart: $\epsilon' \frac{N}{\sqrt{\log N}} \leq S \leq \exp(-c(\log N)^{\frac{1}{9}})N^2$

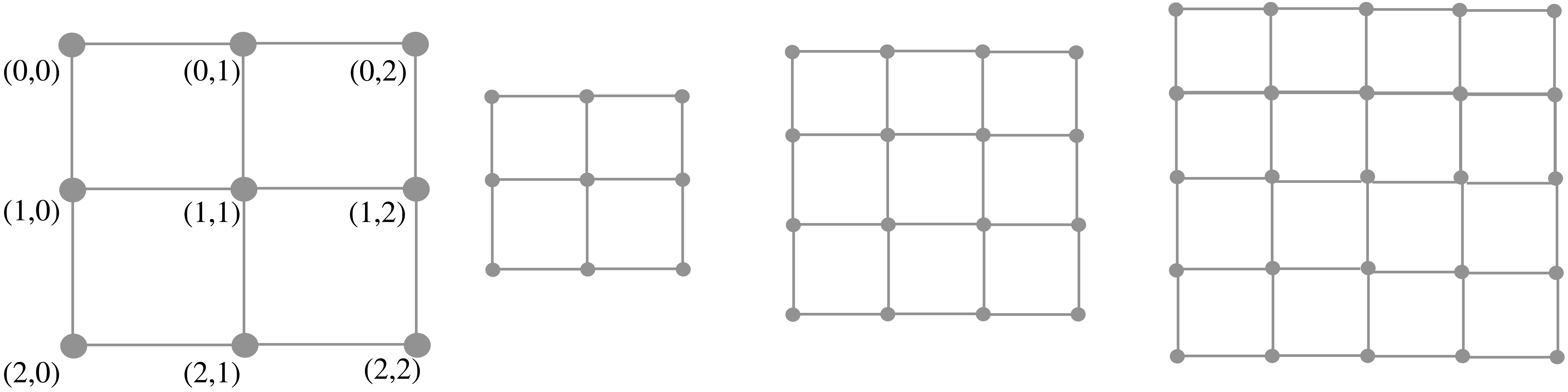
“Karan, this would be a cool problem to think about” - Jordan Ellenberg, mathematician and PhD advisor.
- Problem originated from a study of convergence rates on ordinal embeddings[2].

Motivating Example 2



Question: If you had no prior information, how would you use some of the techniques we have seen so far to generate examples?

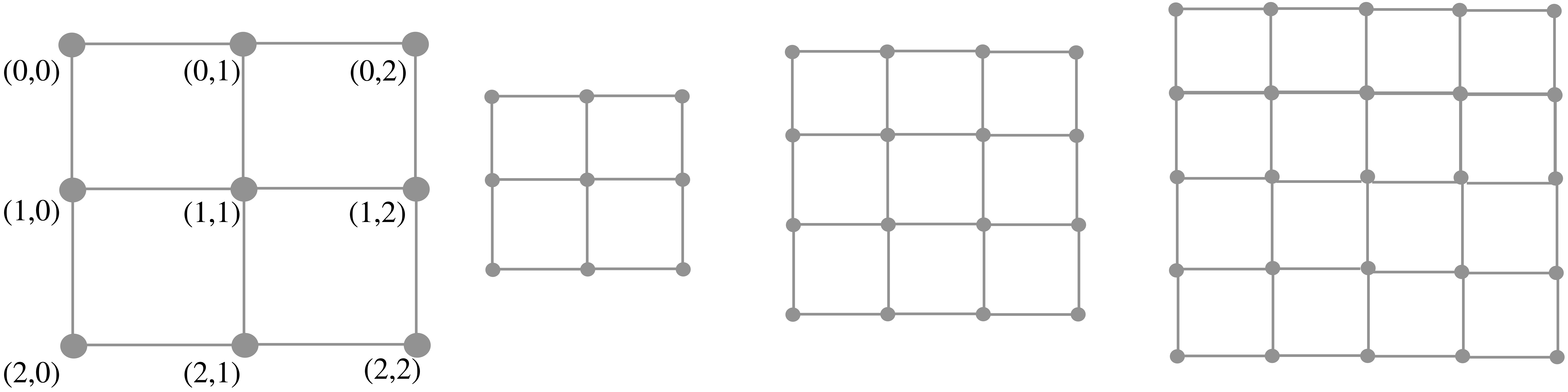
Motivating Example 2



Question: If you had no prior information, how would you use some of the techniques we have seen so far to generate examples?

Key features:

Motivating Example 2

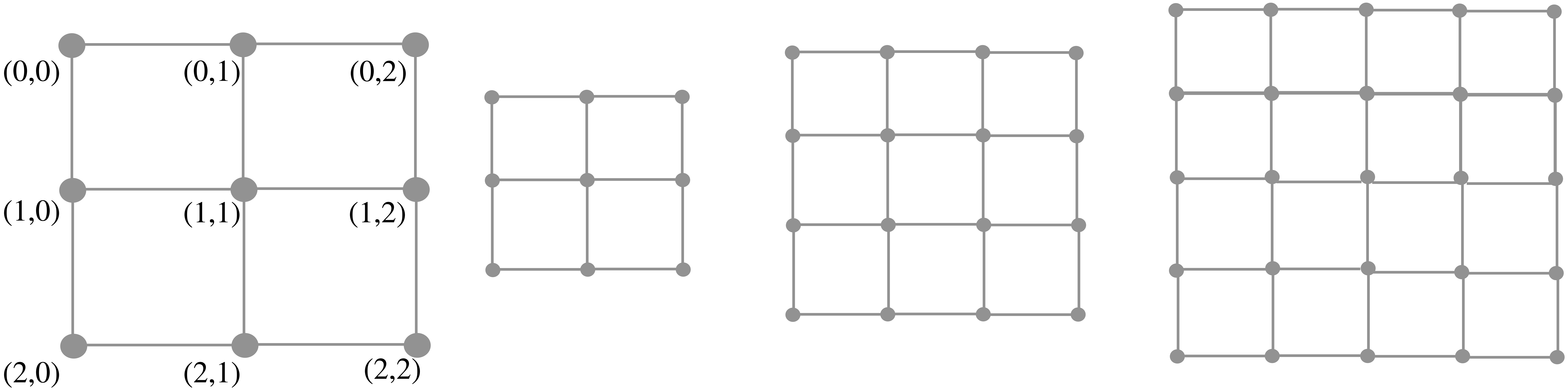


Question: If you had no prior information, how would you use some of the techniques we have seen so far to generate examples?

Key features:

- Hard to solve (exponential in n^2 via brute force)

Motivating Example 2

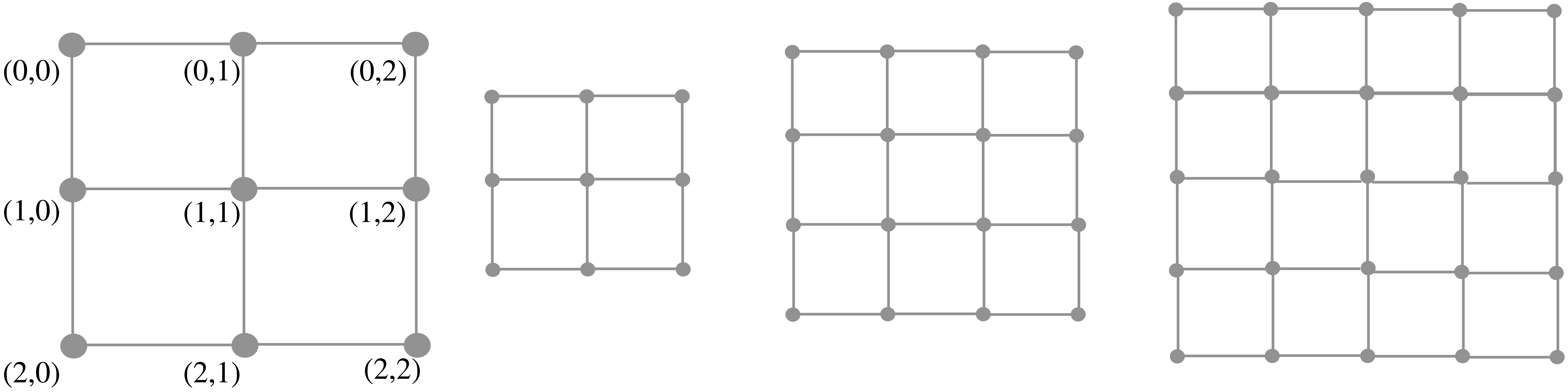


Question: If you had no prior information, how would you use some of the techniques we have seen so far to generate examples?

Key features:

- Hard to solve (exponential in n^2 via brute force)
- Easy to verify

Motivating Example 2

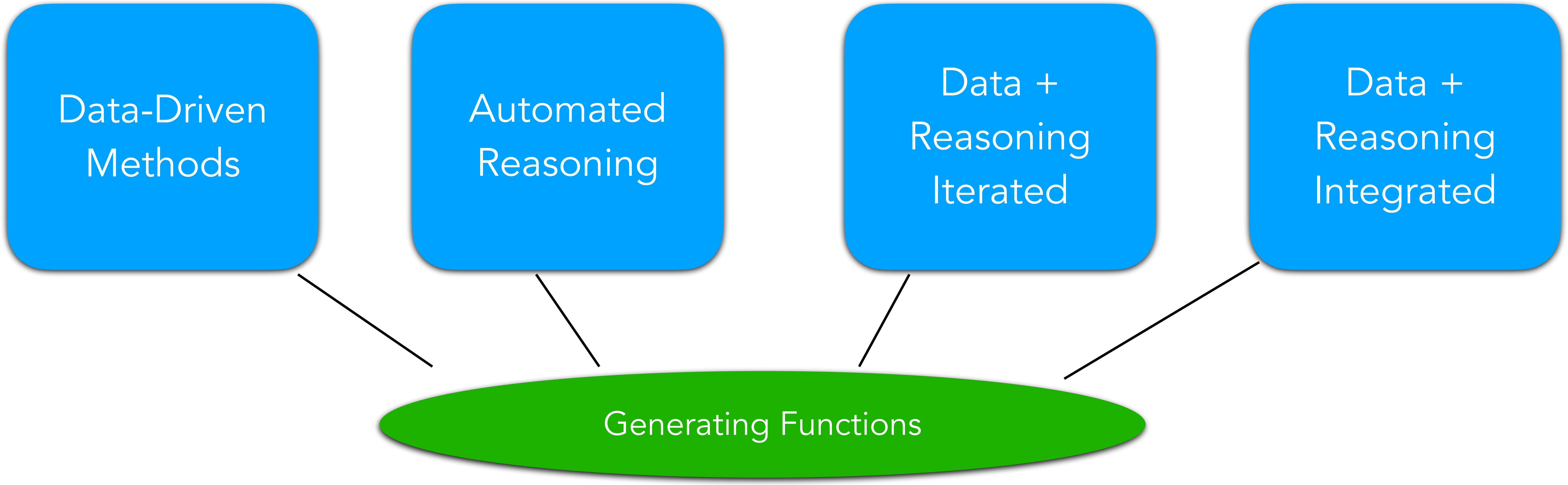


Question: If you had no prior information, how would you use some of the techniques we have seen so far to generate examples?

Key features:

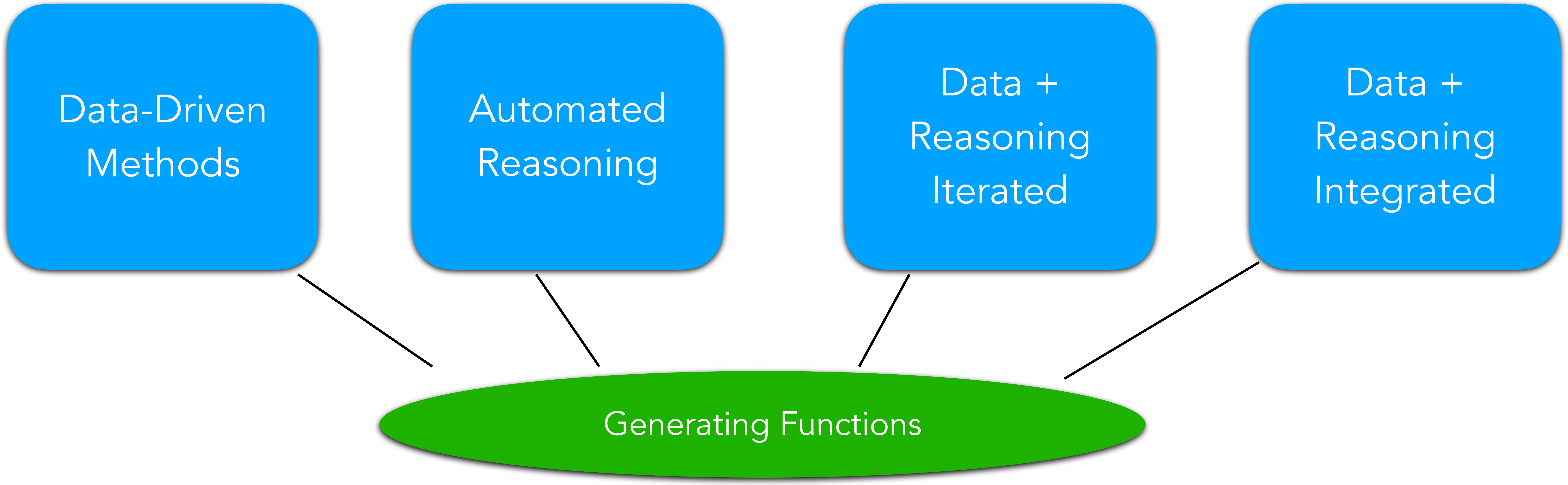
- Hard to solve (exponential in n^2 via brute force)
- Easy to verify
- No data available

How might we approach this?



In all the approaches we have seen so far, we are learning functions.

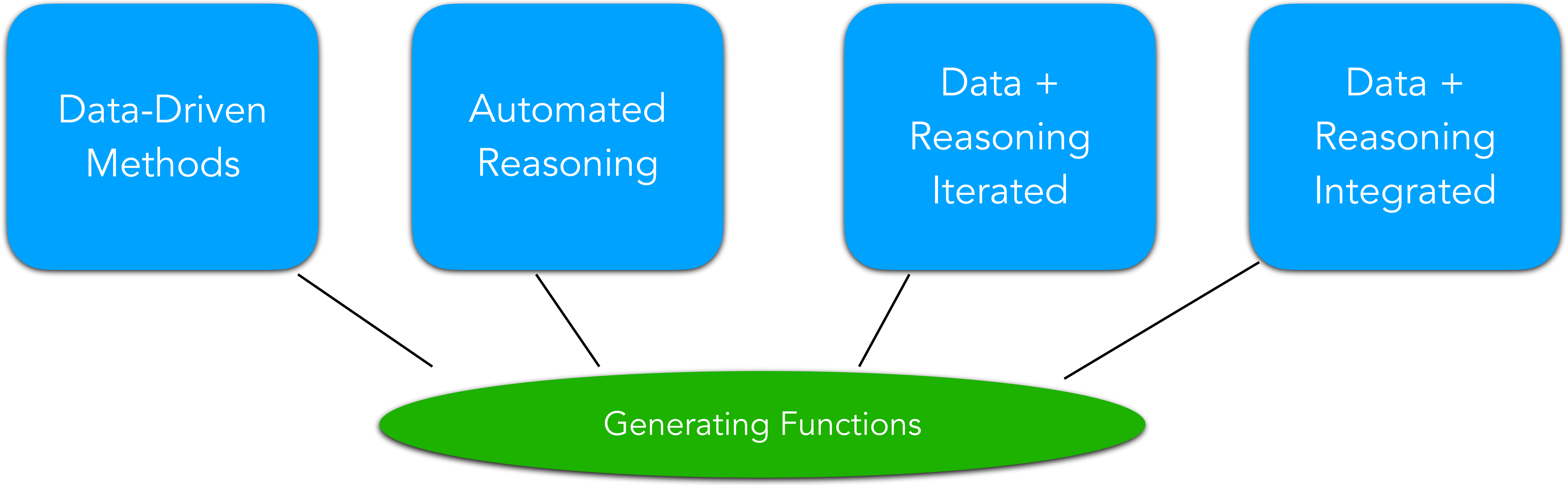
How might we approach this?



In all the approaches we have seen so far, we are learning functions.

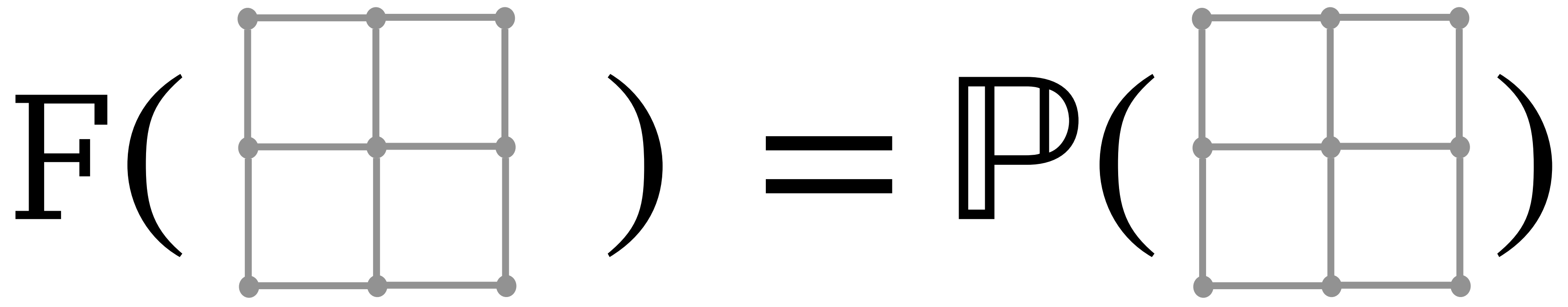
Design question: What function can we try to learn? This will inform what approach we try.

How might we approach this?

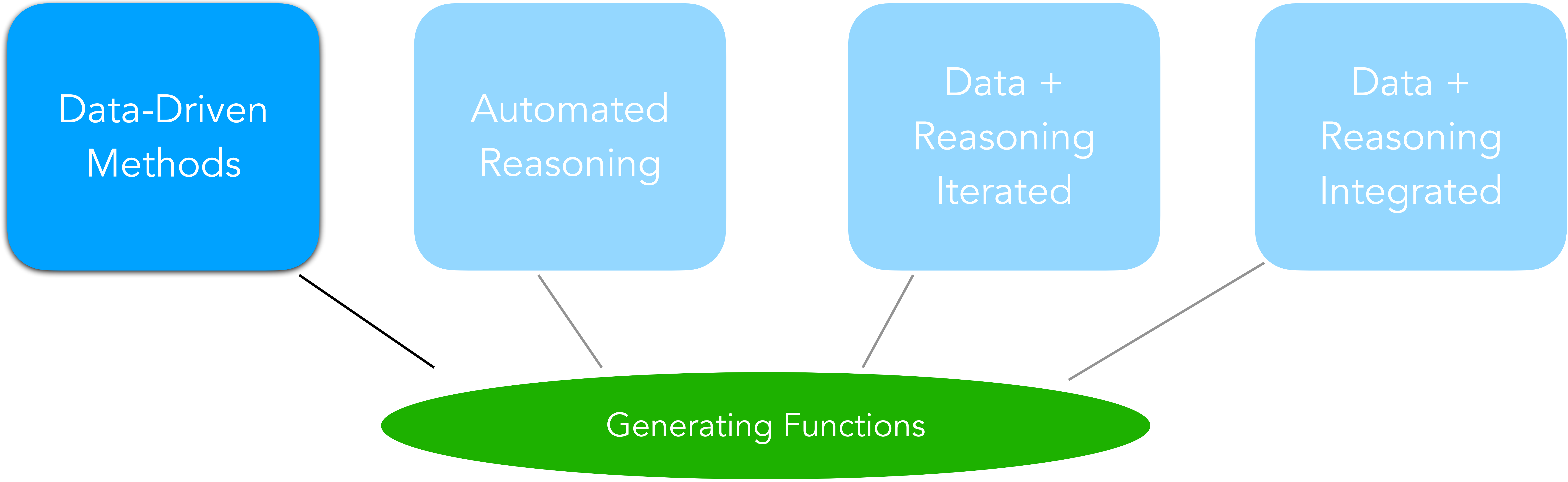


In all the approaches we have seen so far, we are learning functions.

Design question: What function can we try to learn? This will inform what approach we try. One idea: learn a probability distribution on points on the grid.



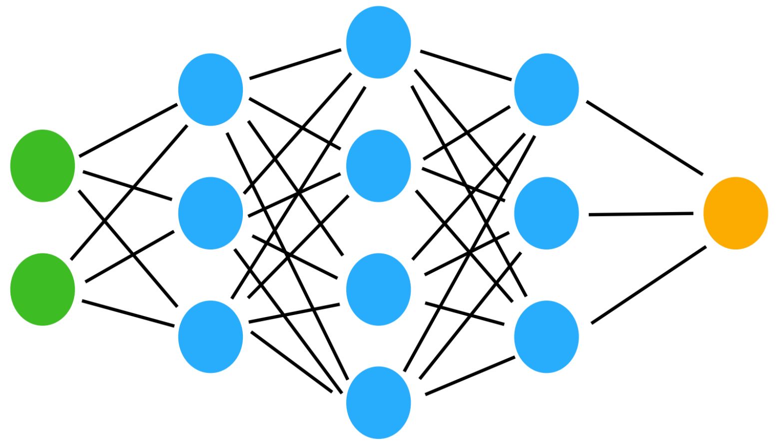
How might we approach this?

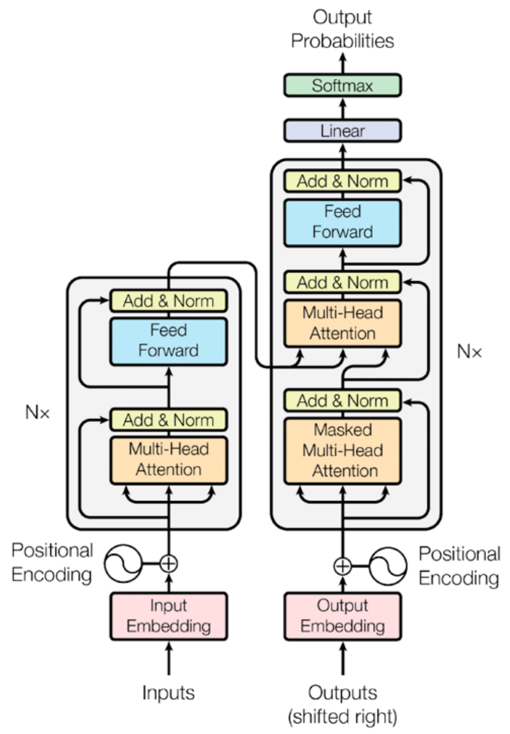


In all the approaches we have seen so far, we are learning functions.

Design question: What function can we try to learn? This will inform what approach we try. One idea: learn a probability distribution on points on the grid.

Neural Networks





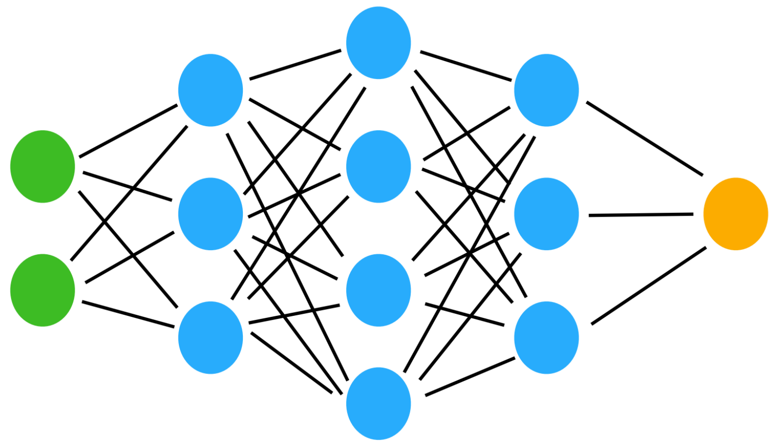
Transformers

How might we approach this?

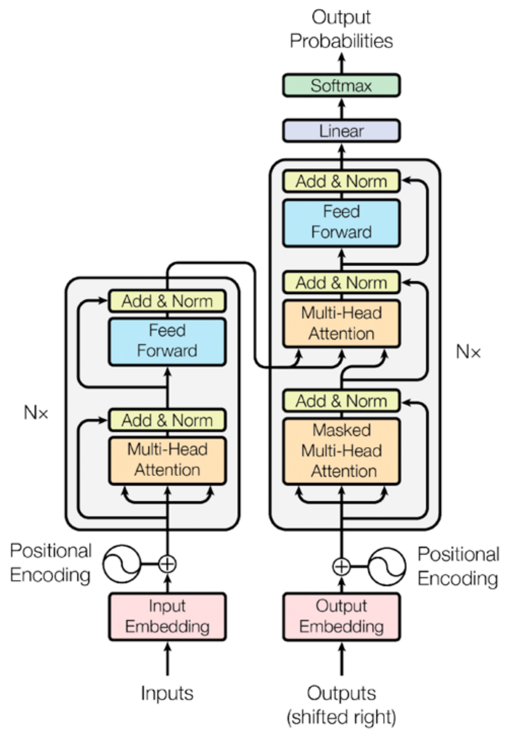
Data-Driven Methods

Advantage of neural-network approaches:

Neural Networks



Transformers



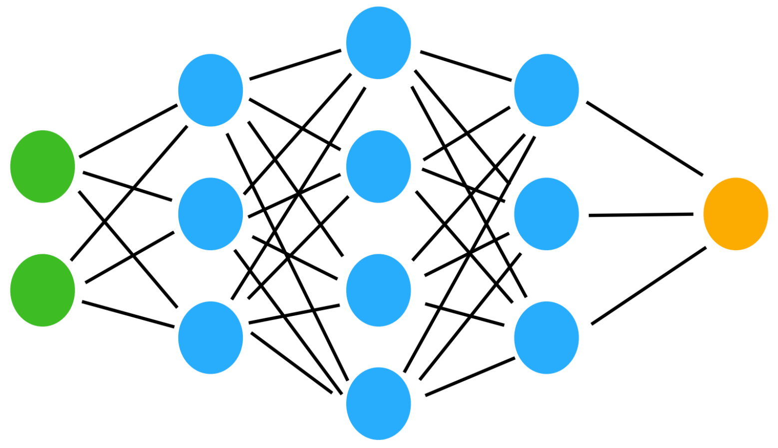
How might we approach this?

Data-Driven Methods

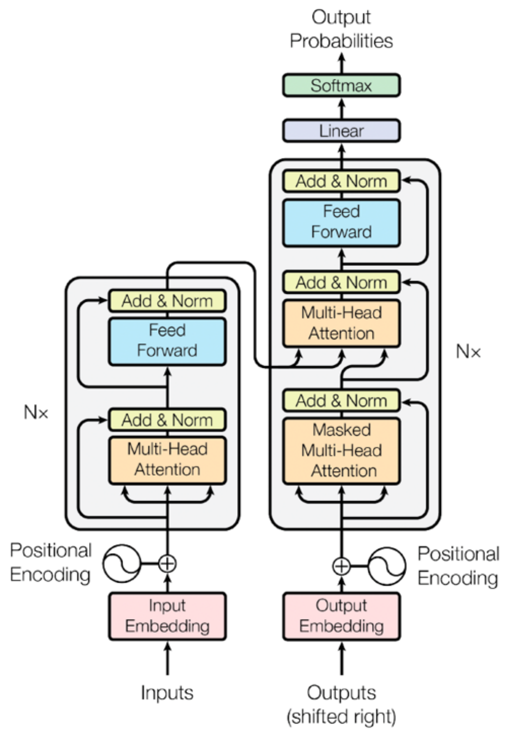
Advantage of neural-network approaches:

- Universal function approximators - so they can learn various kinds of functions.

Neural Networks



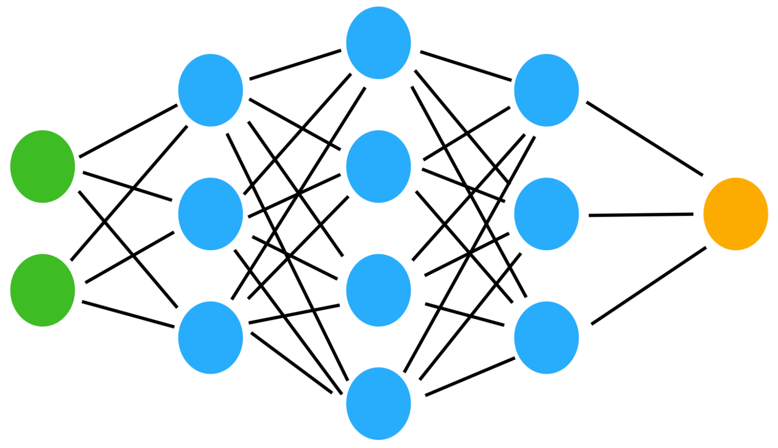
Transformers



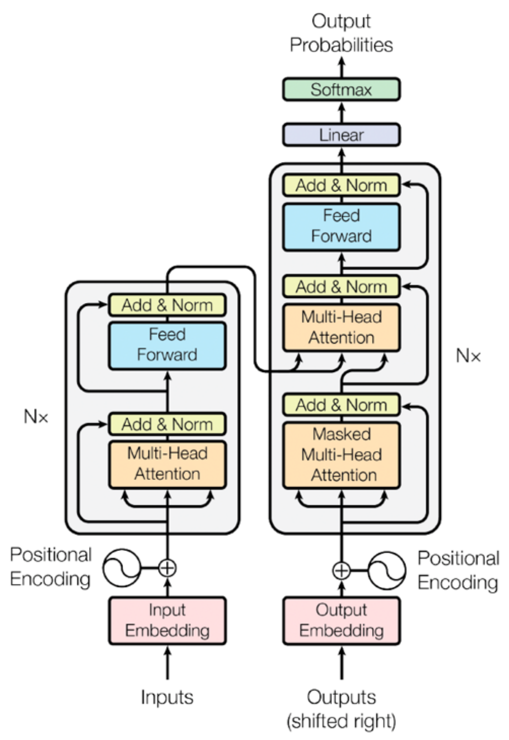
How might we approach this?

Data-Driven Methods

Neural Networks



Transformers



Advantage of neural-network approaches:

- Universal function approximators - so they can learn various kinds of functions.
- Can discover lots of internal structure in data.

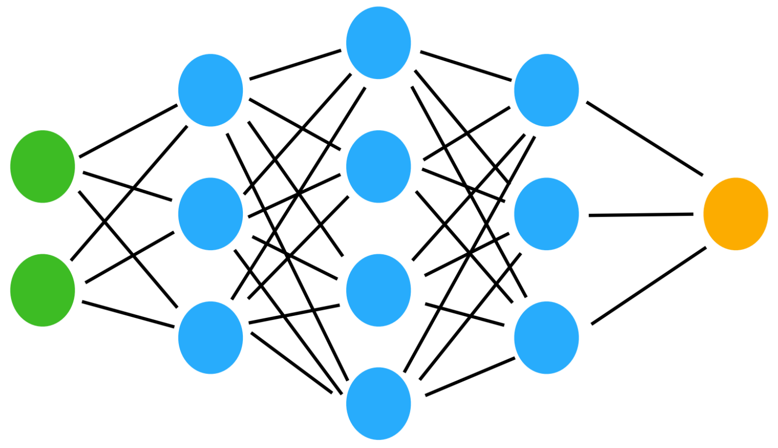
How might we approach this?

Data-Driven Methods

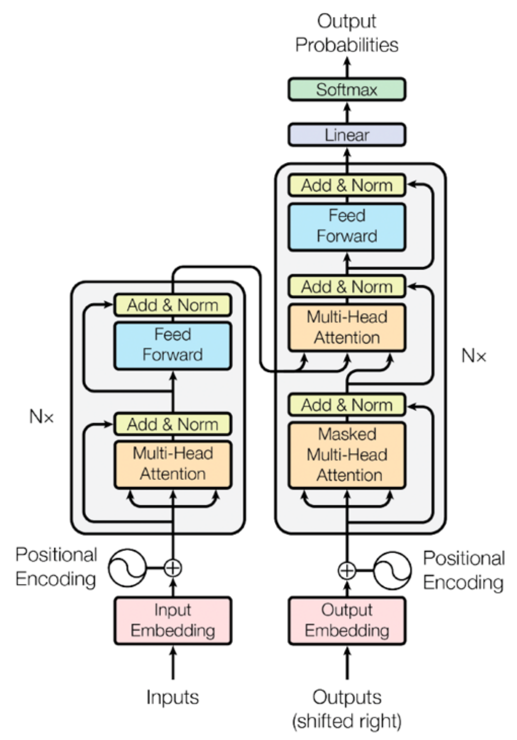
Advantage of neural-network approaches:

- Universal function approximators - so they can learn various kinds of functions.
- Can discover lots of internal structure in data.
- Architecture scales for the problems in higher dimensions and degrees well.

Neural Networks



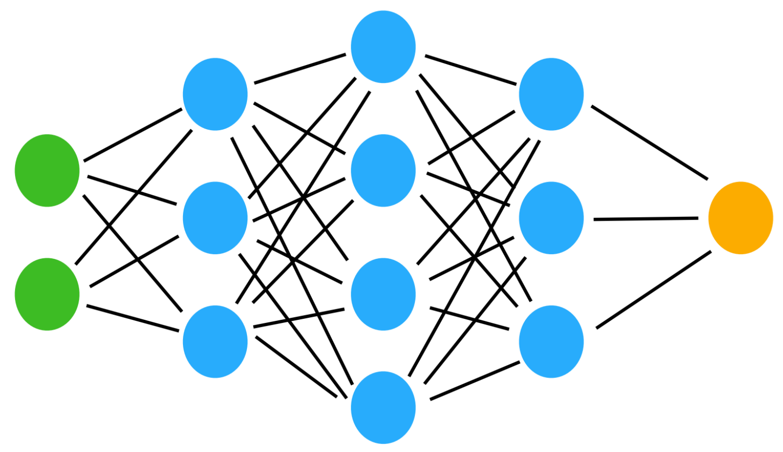
Transformers



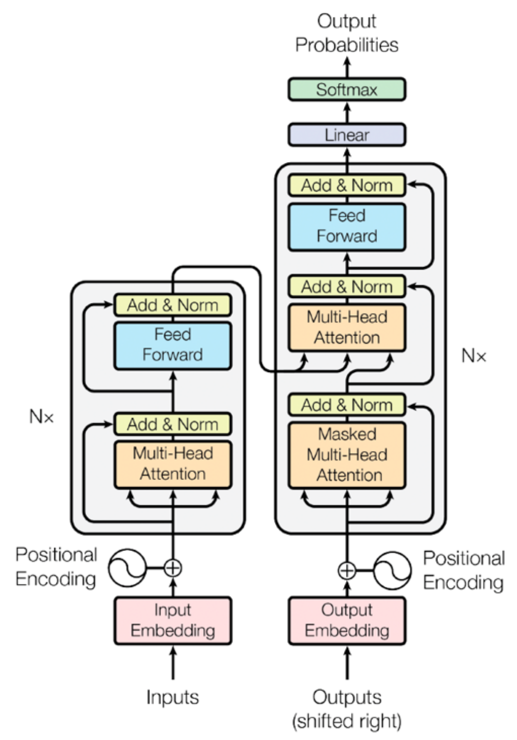
How might we approach this?

Data-Driven Methods

Neural Networks



Transformers



Advantage of neural-network approaches:

- Universal function approximators - so they can learn various kinds of functions.
- Can discover lots of internal structure in data.
- Architecture scales for the problems in higher dimensions and degrees well.

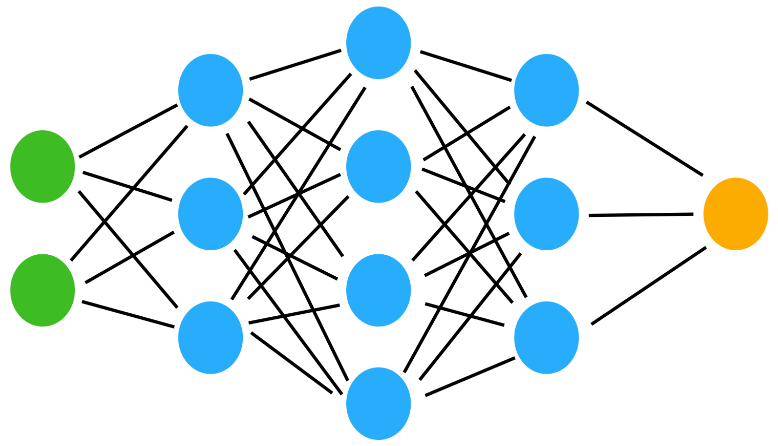
Problem

These examples are hard to come up with. So we have very little training data!

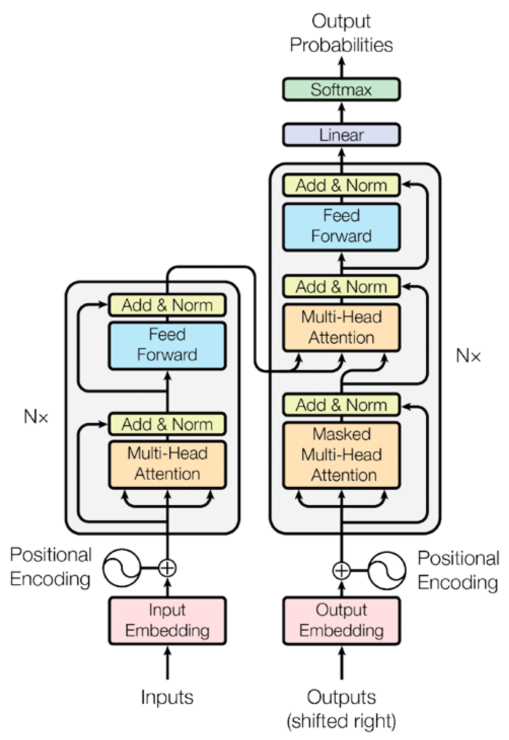
How might we approach this?

Data-Driven Methods

Neural Networks



Transformers



Advantage of neural-network approaches:

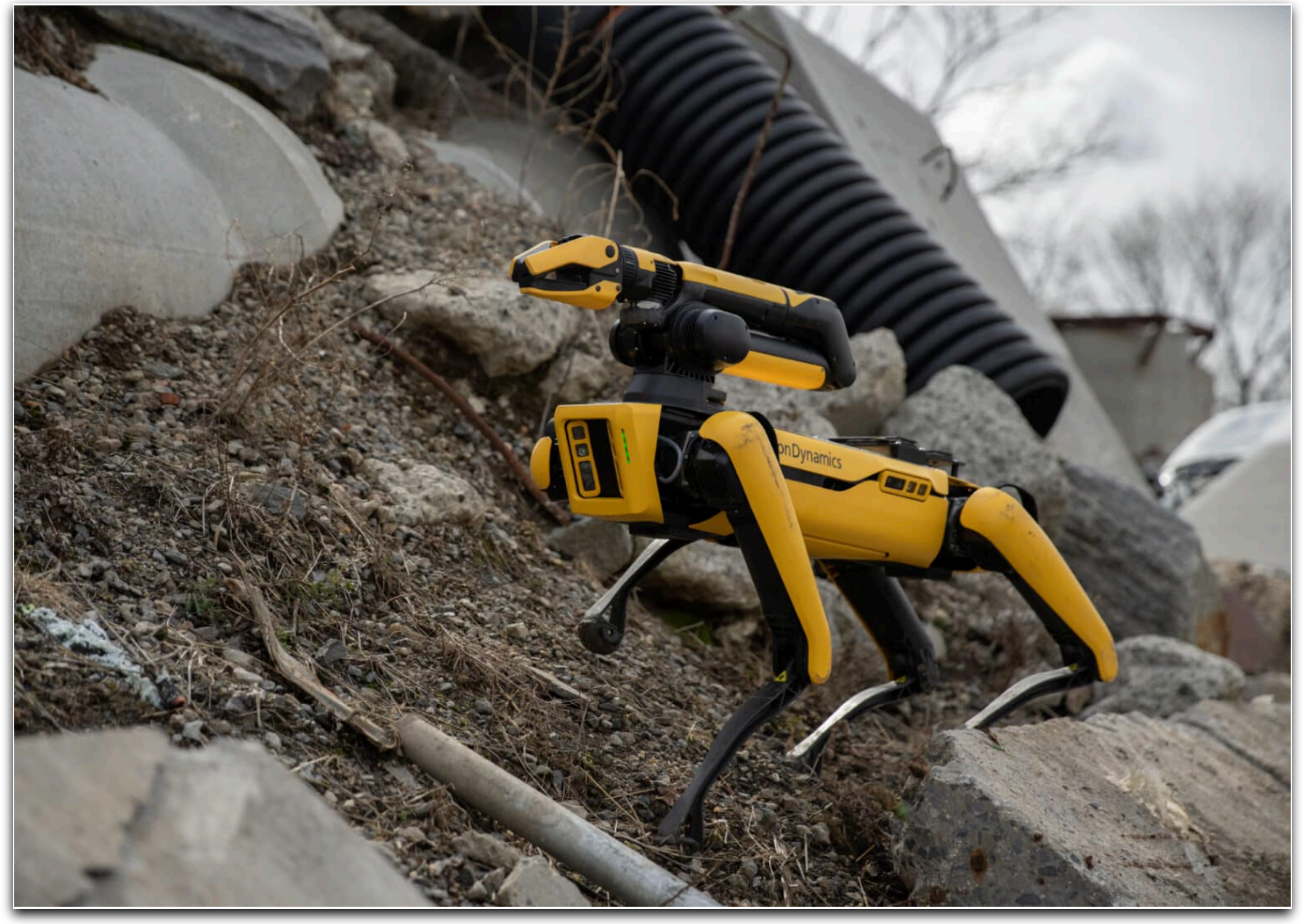
- Universal function approximators - so they can learn various kinds of functions.
- Can discover lots of internal structure in data.
- Architecture scales for the problems in higher dimensions and degrees well.

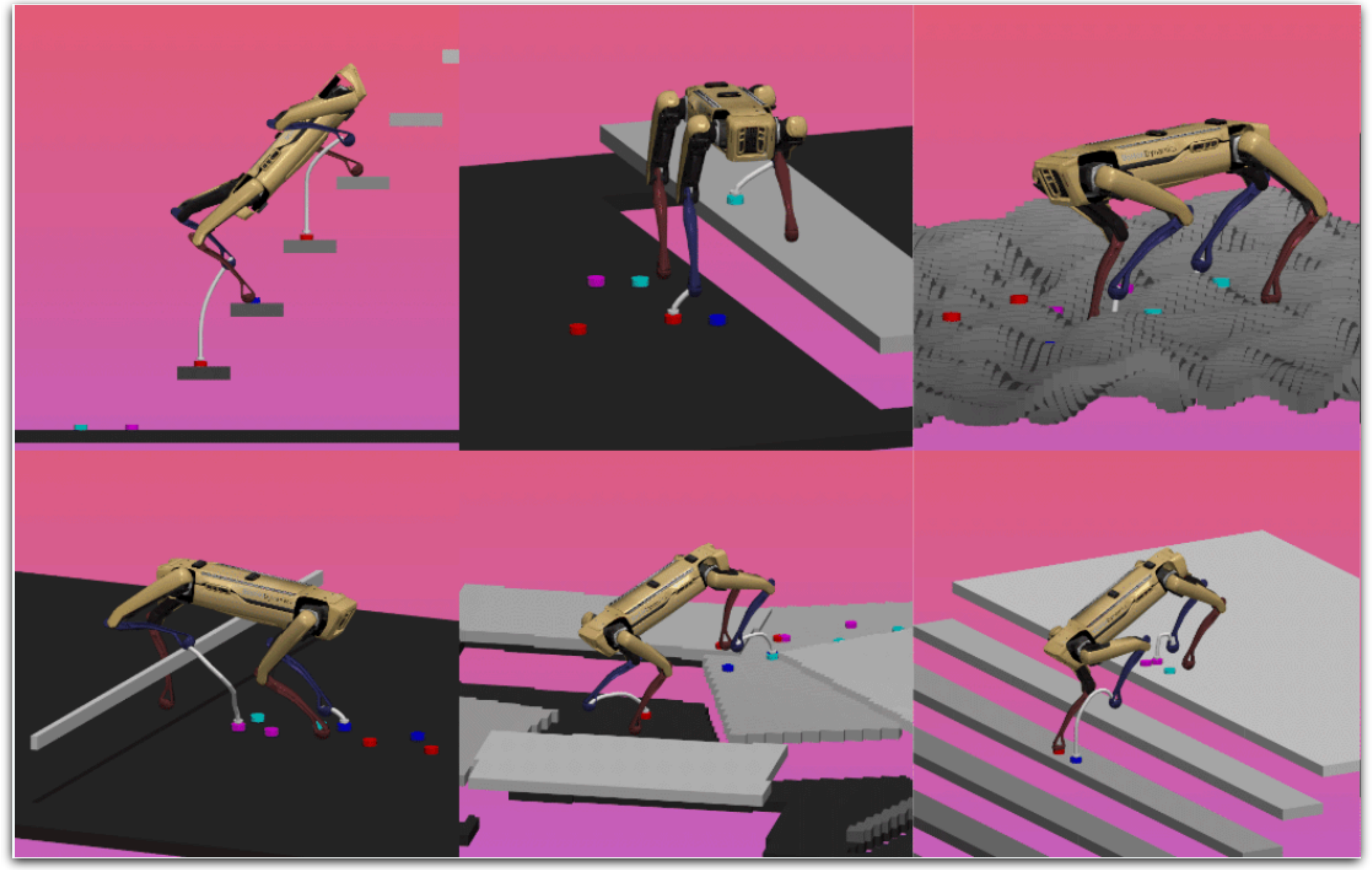
Problem

These examples are hard to come up with. So we have very little training data!

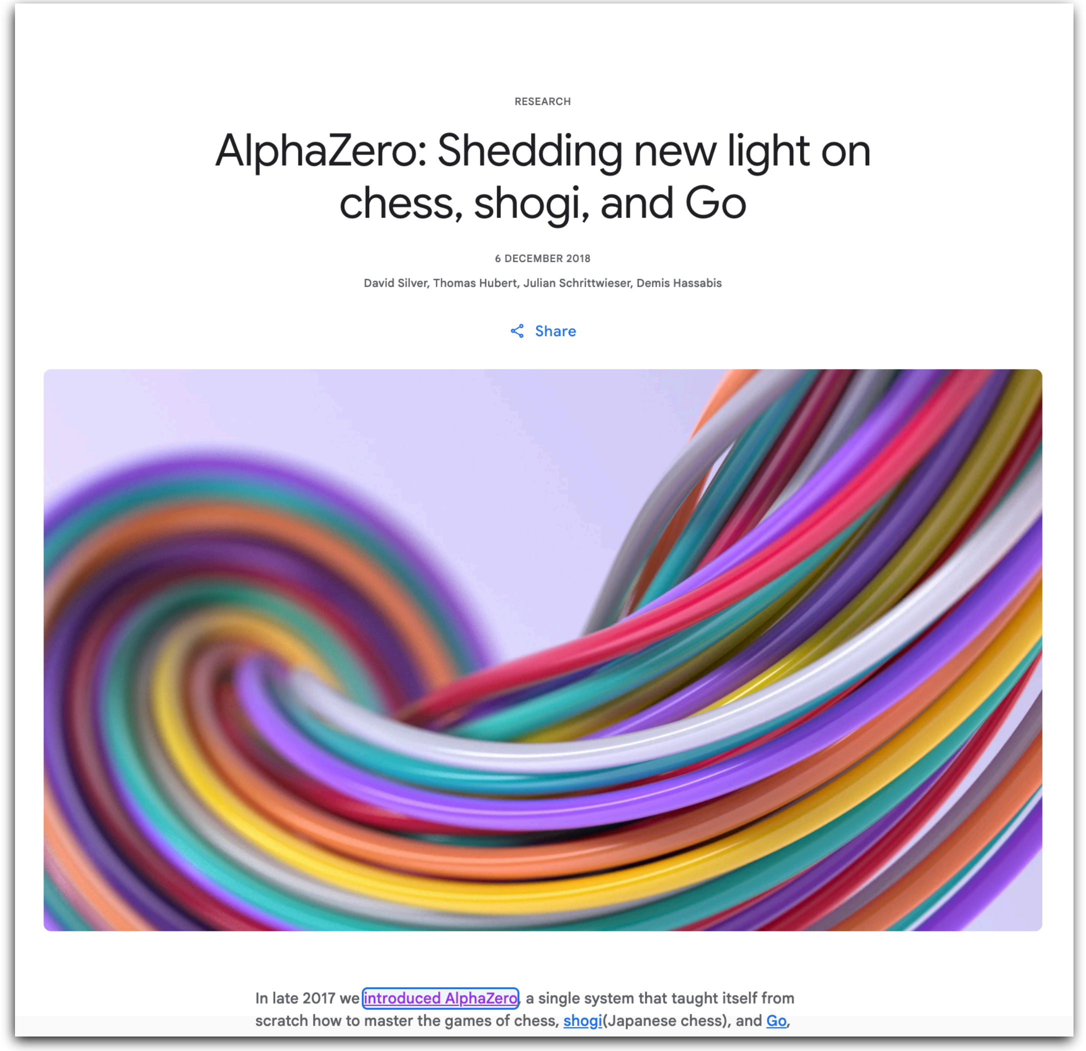
Solution
Self Improvement

Approach 1: Reinforcement Learning





Spot, Boston Dynamics[3].



AlphaZero, DeepMind[4].

Reinforcement Learning

A self improvement approach to machine learning where we need to learn tasks for which we have little to no data.

One step lower

Aim: To train an agent with no prior knowledge to learn a policy for taking actions in the environment in order to maximize a reward and achieve a goal.

Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).

Approach 1: Reinforcement Learning

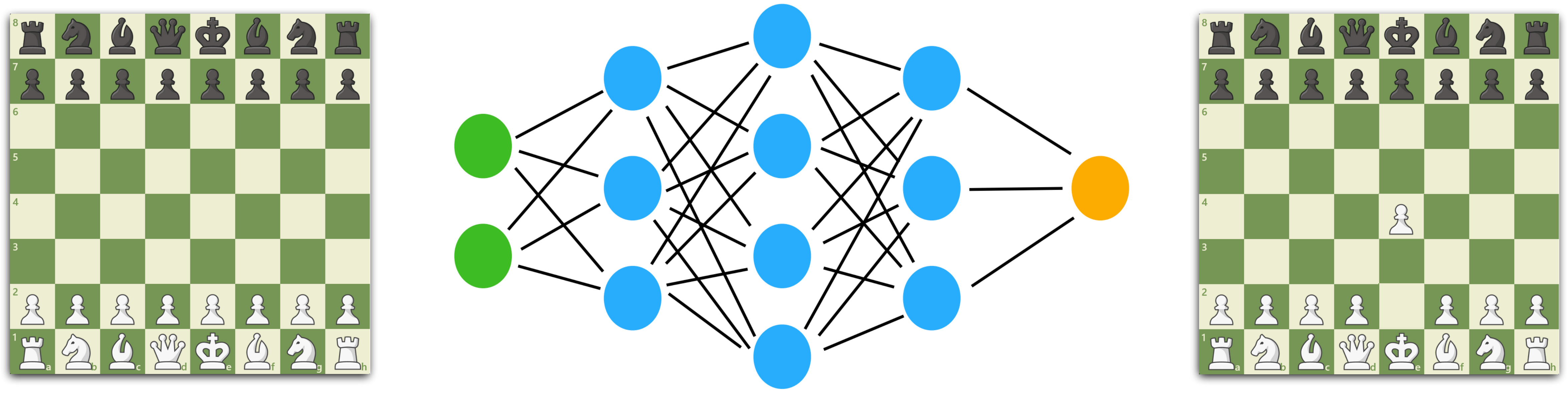
What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).

Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

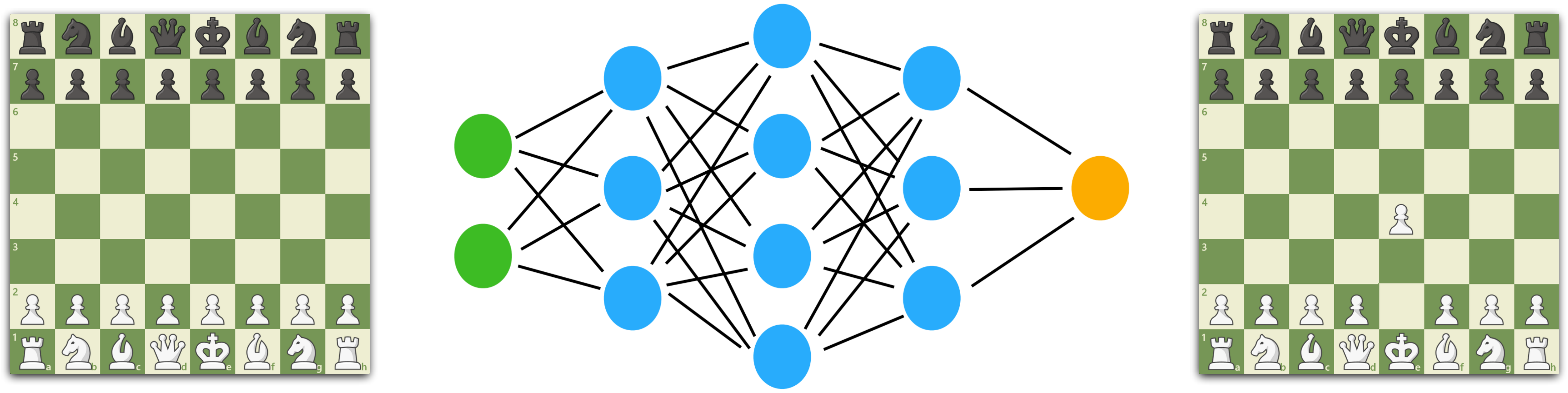
1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.



Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.



Policy encoded in network weights
(Warning: The policy is not always the same as the agent. Eg. Spot, QLearning, etc)

Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

:D

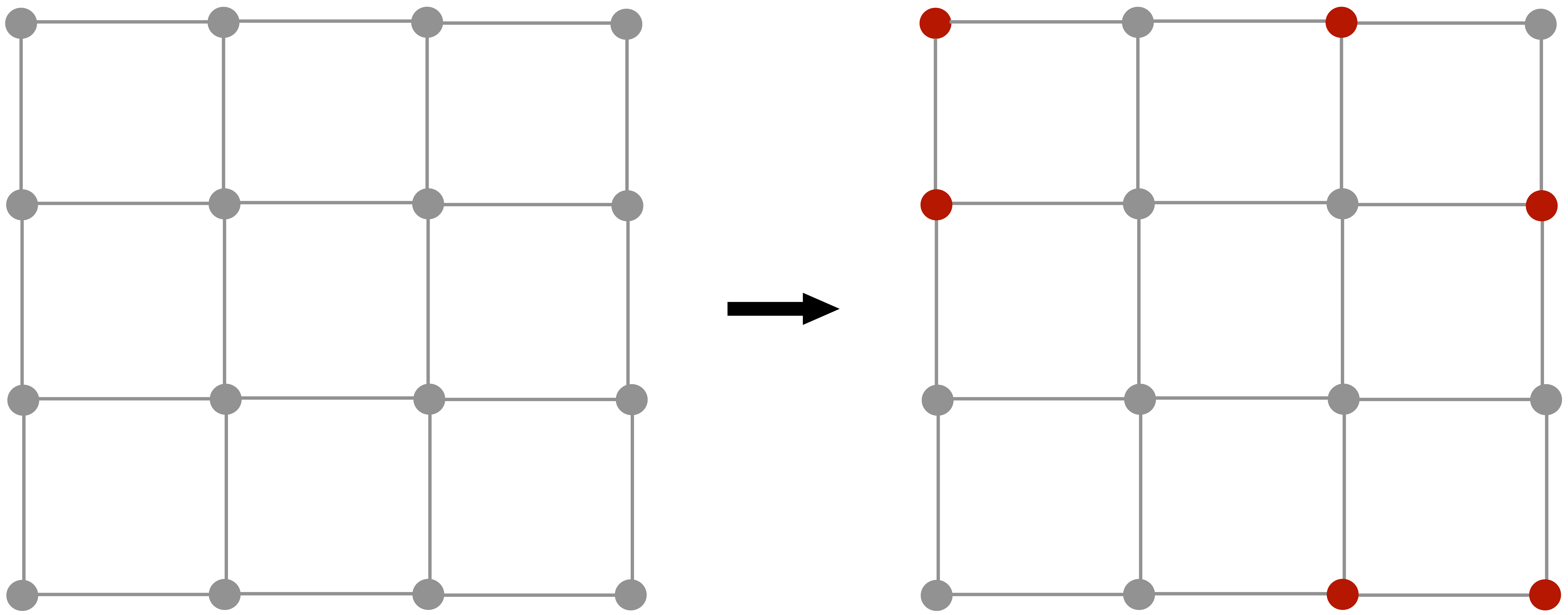
;-;

Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

So how does this work for finding large subsets of lattices?



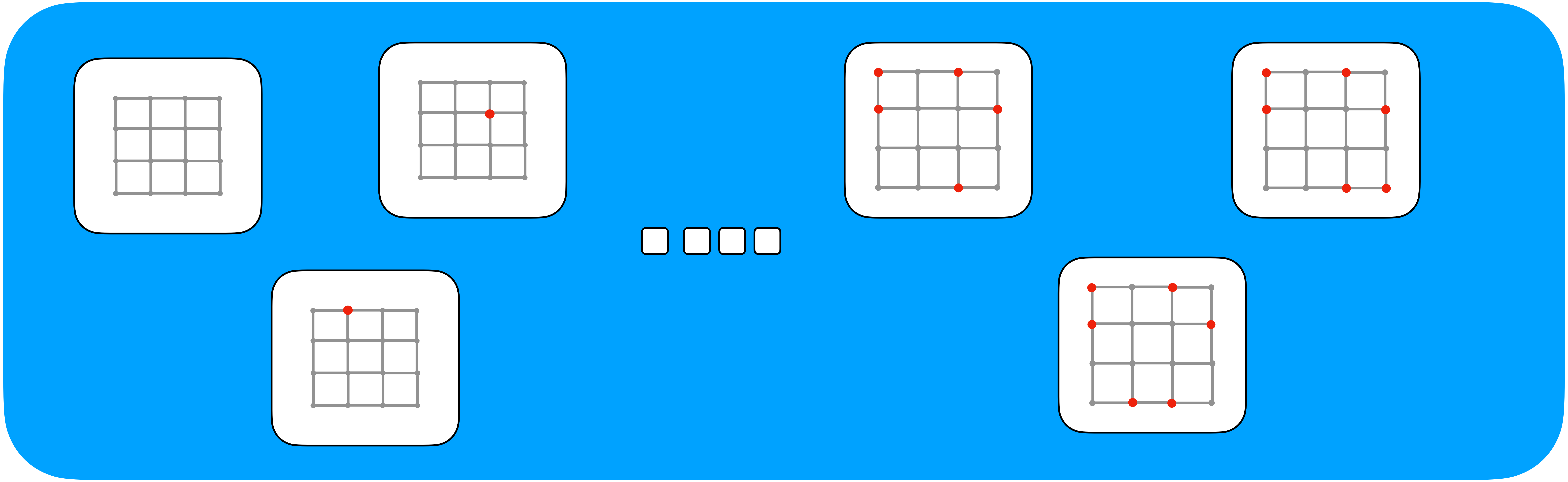
Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

So how does this work for finding large subsets of lattices?

<u>States</u>
The set of all possible subsets of the lattice.



Approach 1: Reinforcement Learning

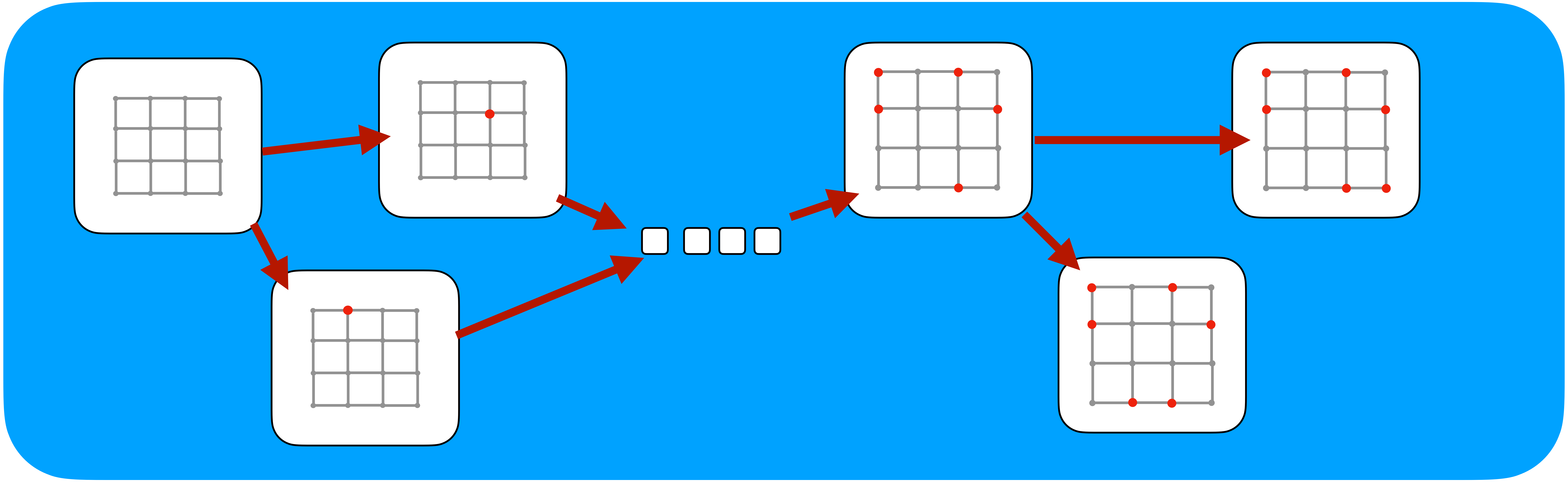
What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

So how does this work for finding large subsets of lattices?

States
The set of all possible subsets of the lattice.

Actions
Adding a point to an existing given state.



Approach 1: Reinforcement Learning

What do we then need to train a reinforcement learning agent:

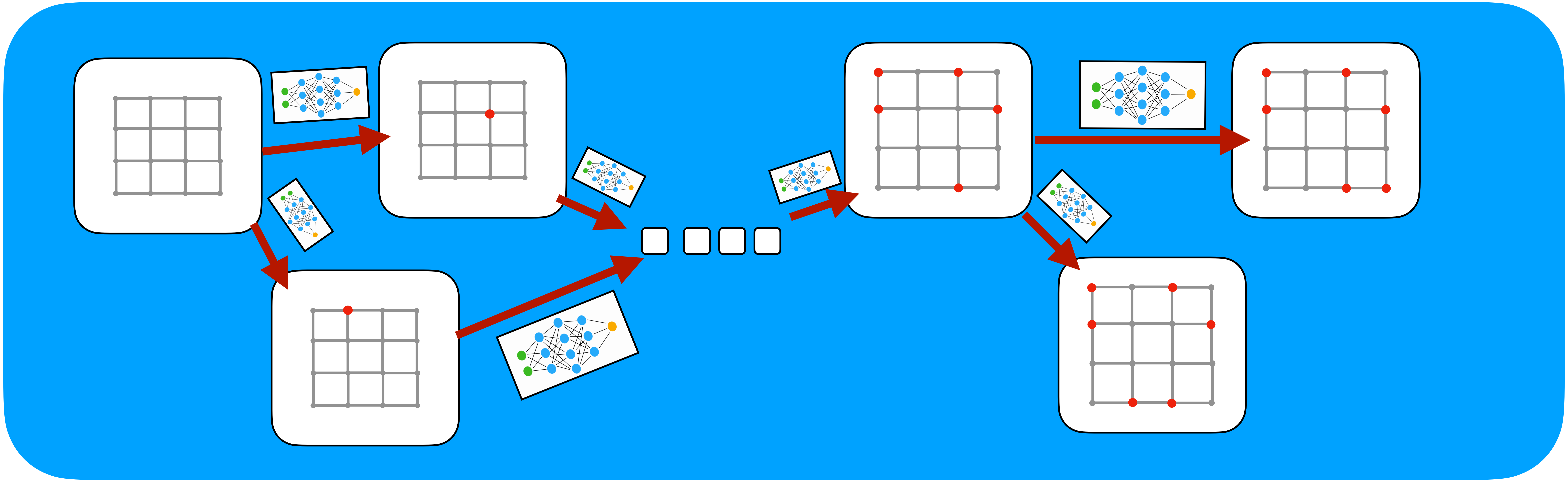
1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

So how does this work for finding large subsets of lattices?

States
The set of all possible subsets of the lattice.

Actions
Adding a point to an existing given state.

Agent
Neural Network that will decide point placement.



Approach 1: Reinforcement Learning

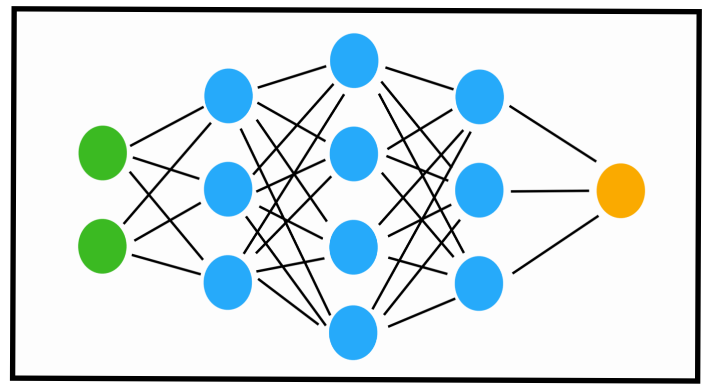
What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state.

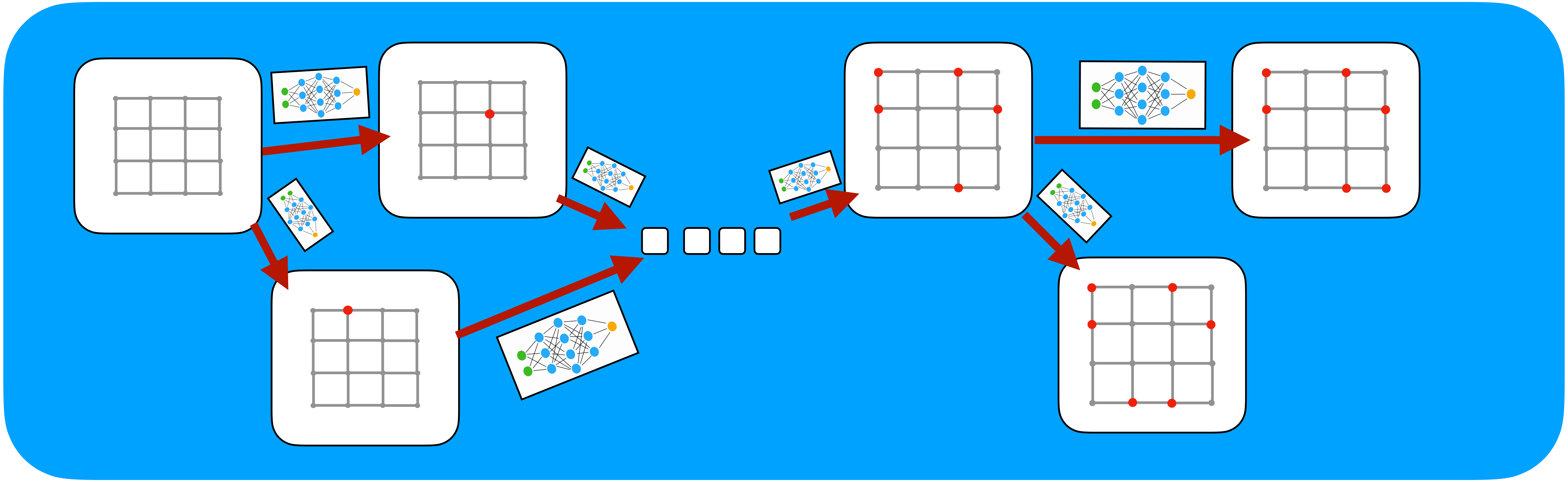
So how does this work for finding large subsets of lattices?

States
The set of all possible subsets of the lattice.

Actions
Adding a point to an existing given state.



Agent
Neural Network that will decide point placement.



Approach 1: Reinforcement Learning

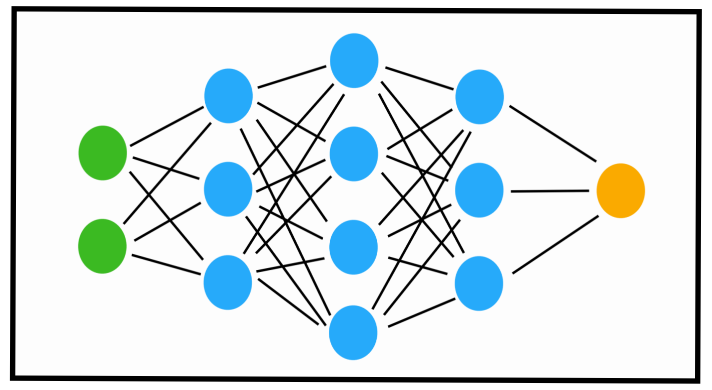
What do we then need to train a reinforcement learning agent:

1. An environment (the state space of the game).
2. An action space (the set of actions that change the state).
3. An agent that will choose actions to move between states.
4. A policy function that the agent will use to choose actions.
5. A value function to assign a score each state

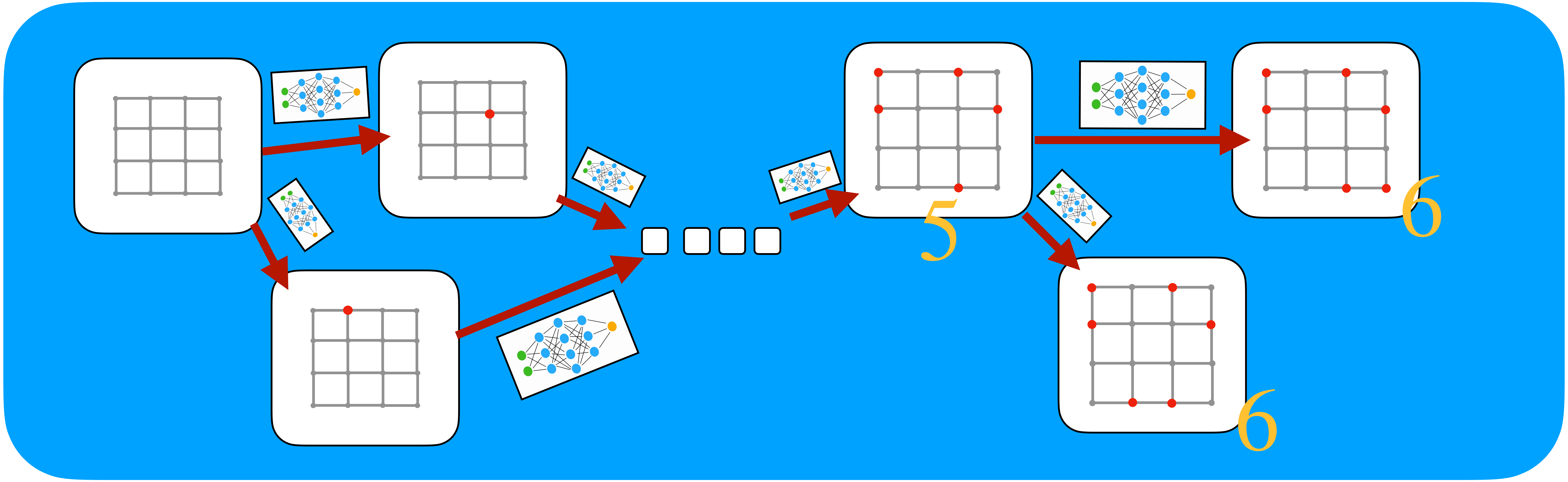
So how does this work for finding large subsets of lattices?

<u>States</u>
The set of all possible subsets of the lattice.

<u>Actions</u>
Adding a point to an existing given state.



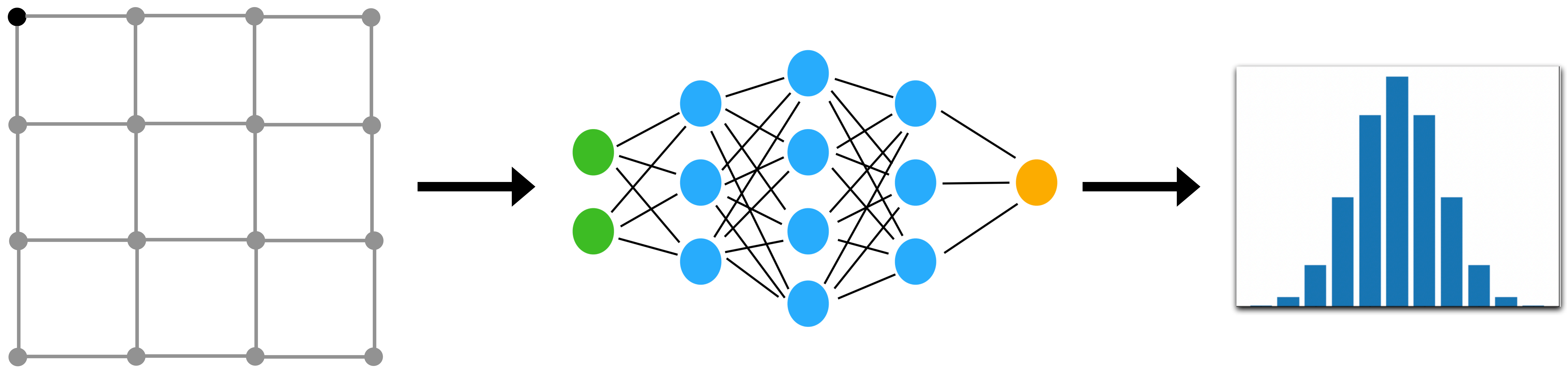
<u>Agent</u>
Neural Network that will decide point placement.



Approach 1: Reinforcement Learning

The game

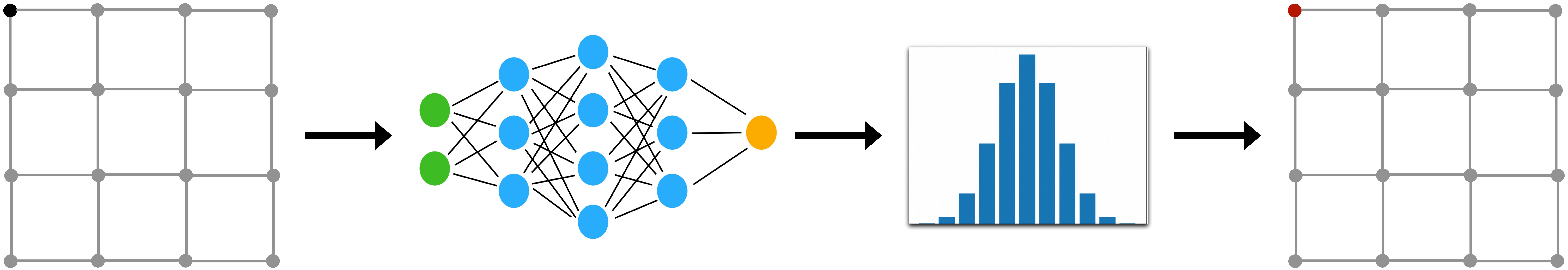
Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$.



Approach 1: Reinforcement Learning

The game

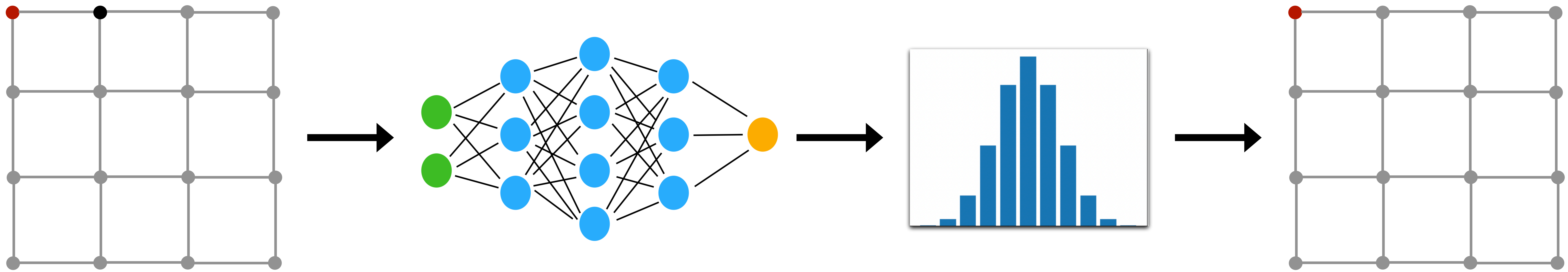
Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p.



Approach 1: Reinforcement Learning

The game

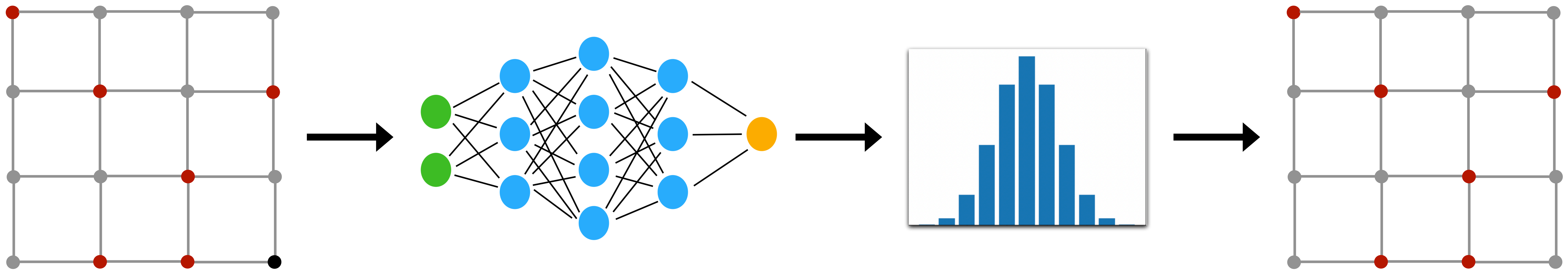
Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index



Approach 1: Reinforcement Learning

<u>The game</u>

Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.



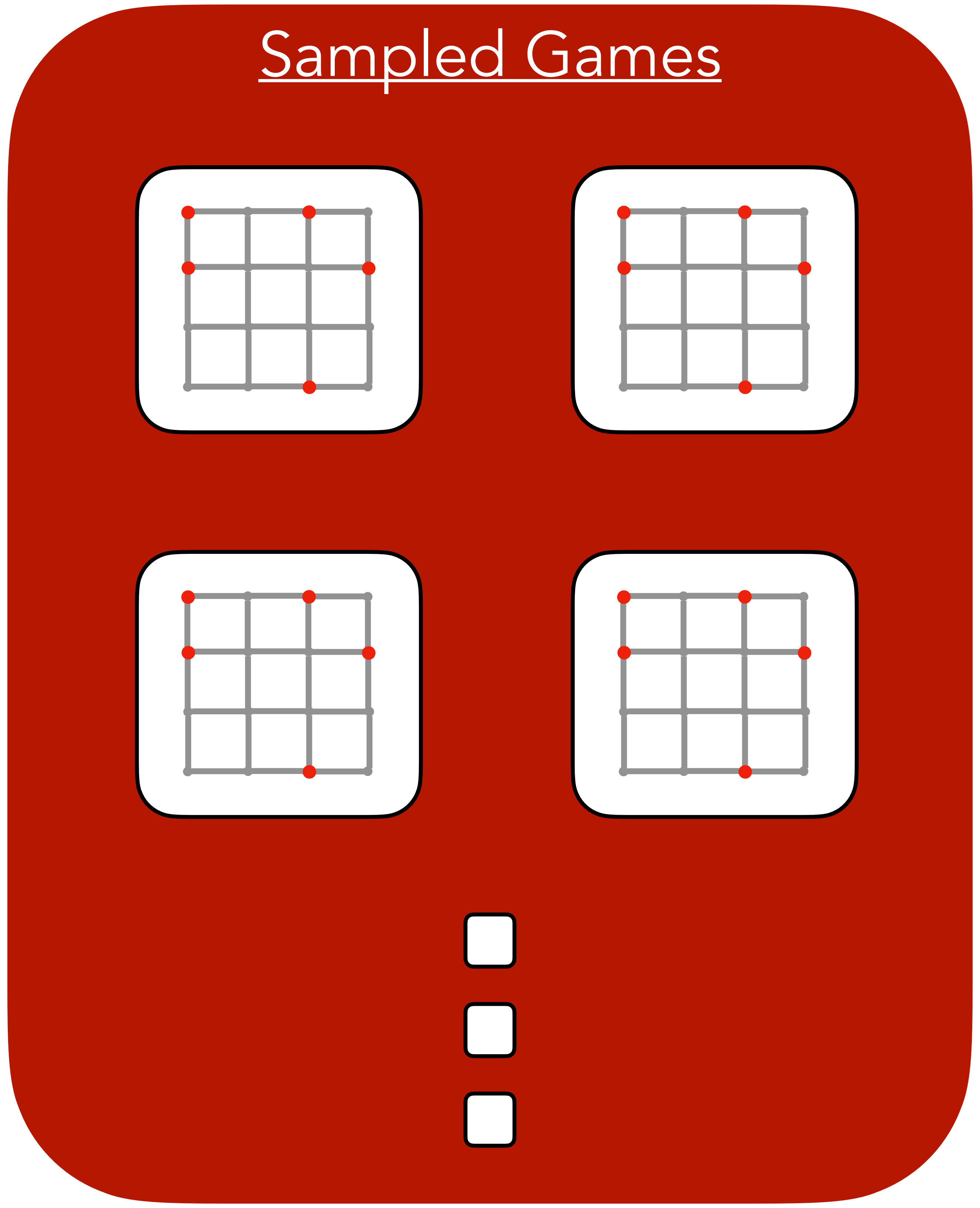
Approach 1: Reinforcement Learning

The game

Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.

The evaluation

Sample lots of games (~2000)



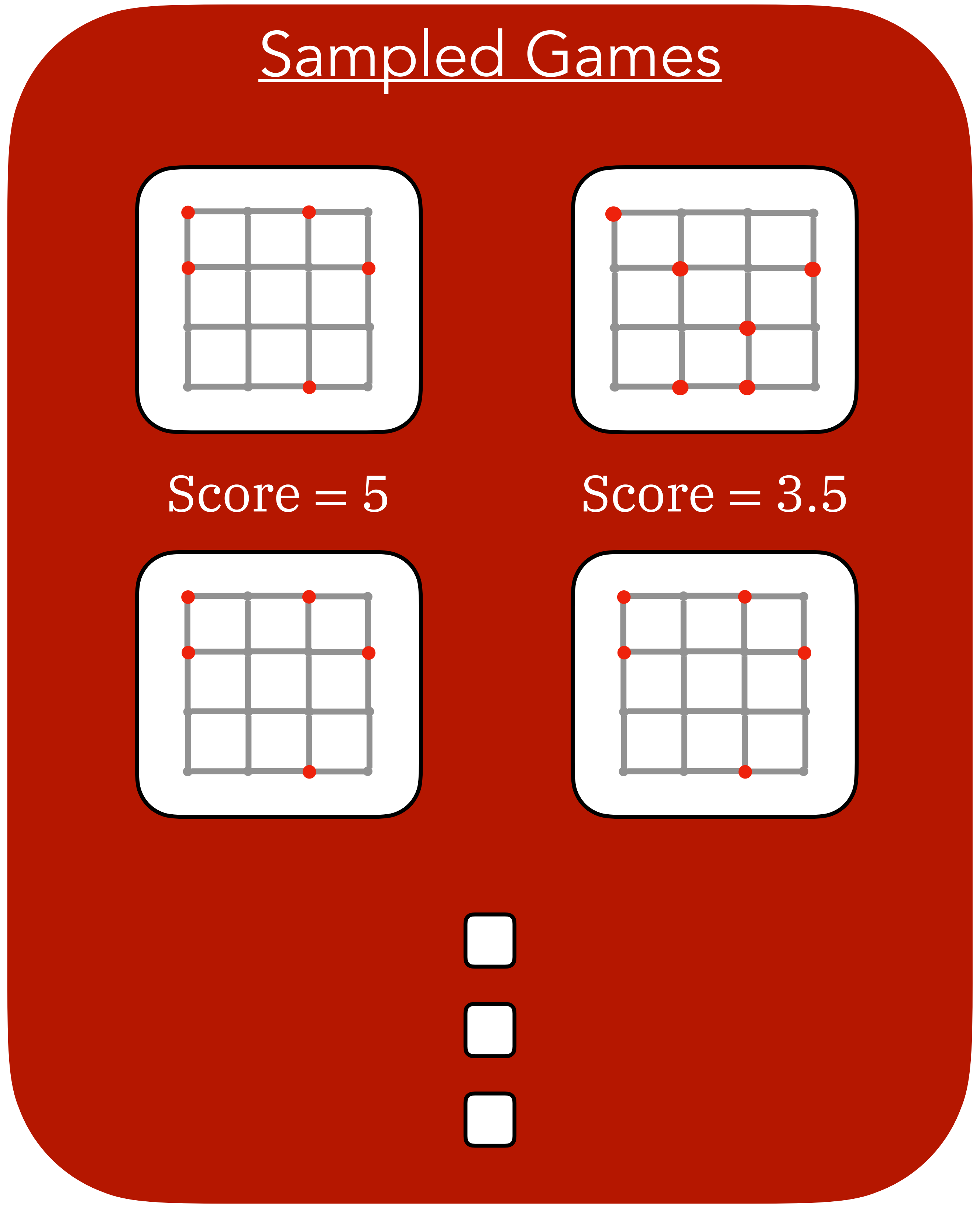
Approach 1: Reinforcement Learning

The game

Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.

The evaluation

Sample lots of games (~2000) and assign a score to each one. Filter out the top $k\,\%$.



Sample Scoring Function

Score = # of points - λ(# of isosceles Δs)

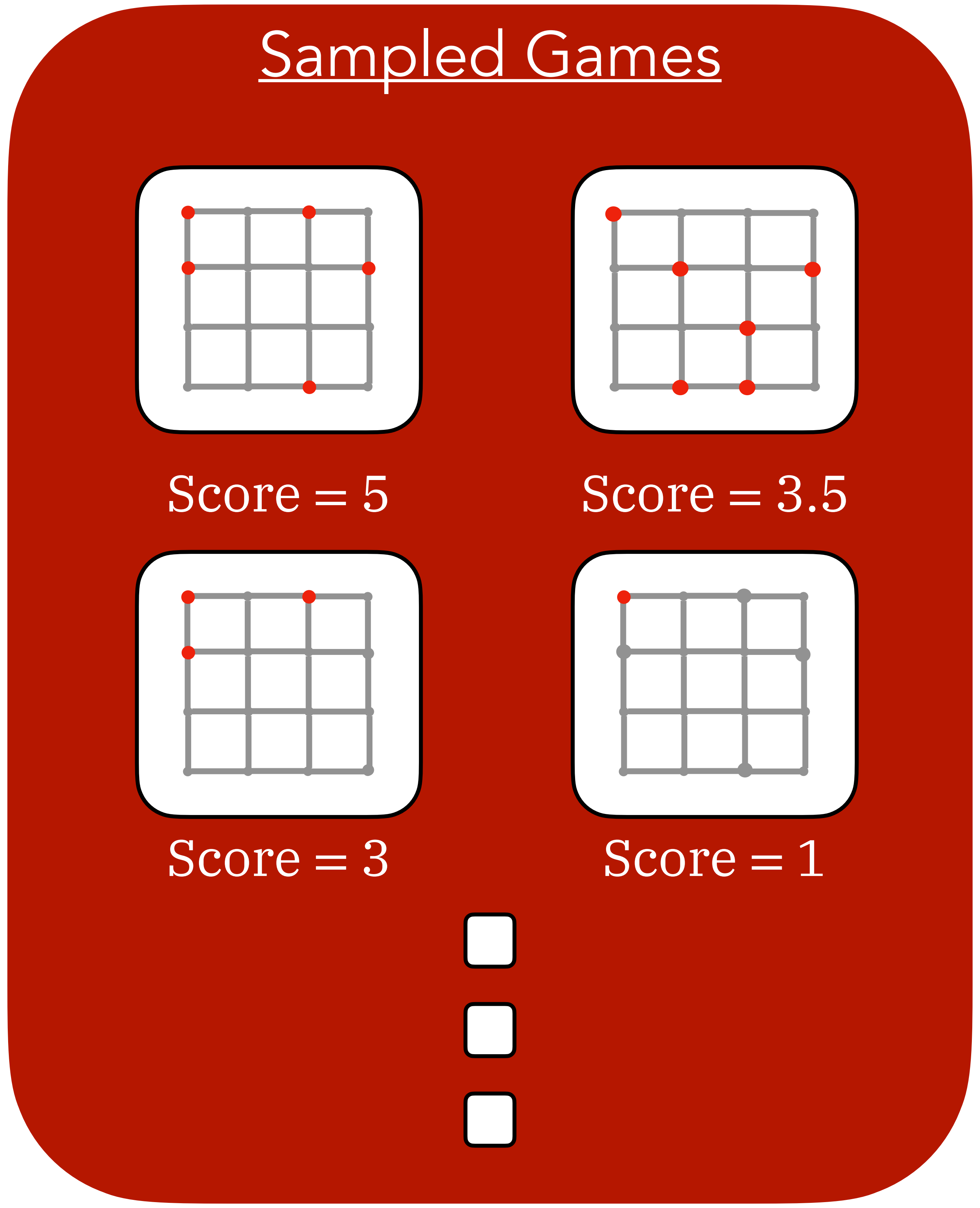
Approach 1: Reinforcement Learning

The game

Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.

The evaluation

Sample lots of games (~2000) and assign a score to each one. Filter out the top $k\,\%$.

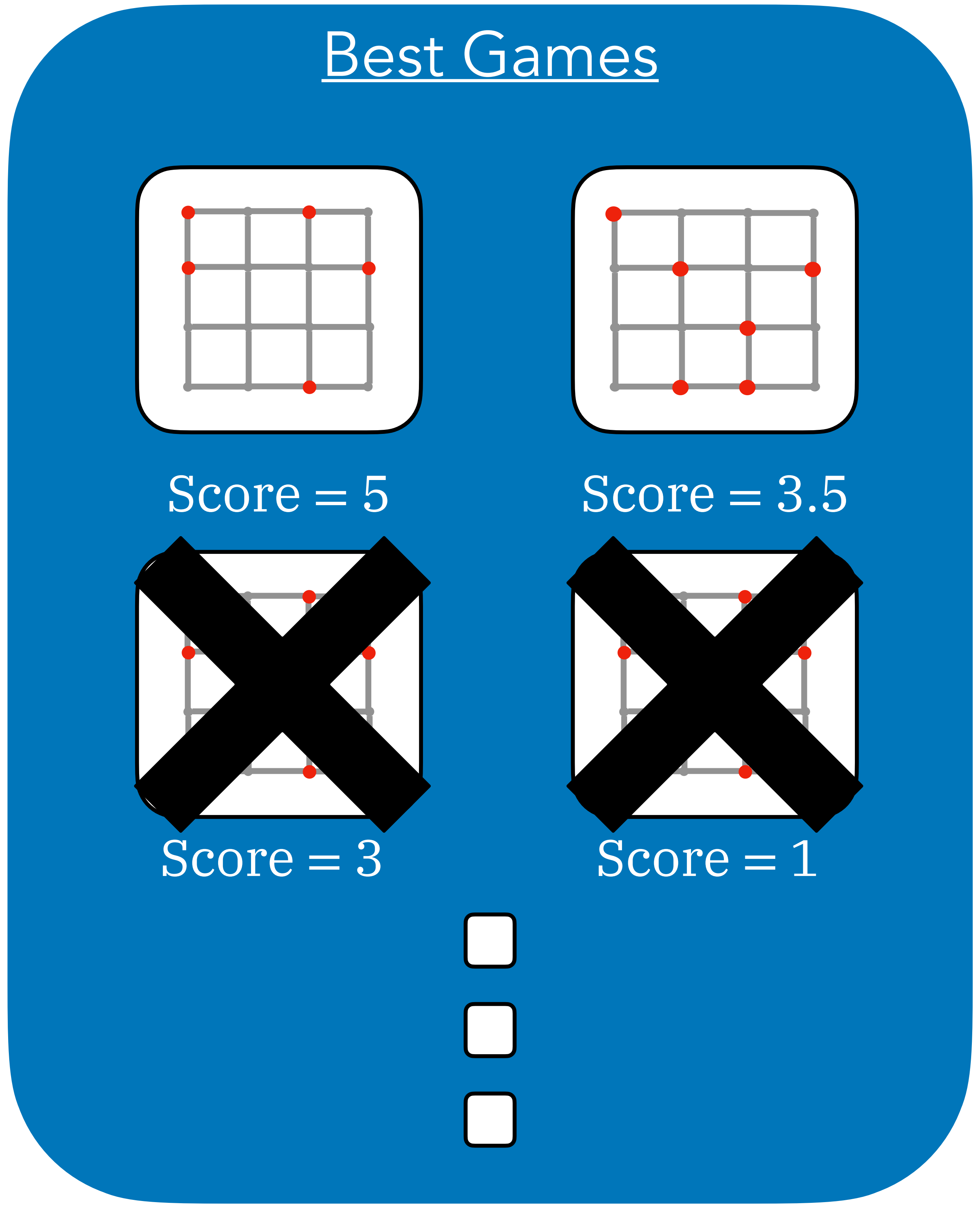


Sample Scoring Function

Score = # of points - λ(# of isosceles Δs)

Note:

The scoring here is smooth. Binary / categorical scoring would be worse.



Approach 1: Reinforcement Learning

The game

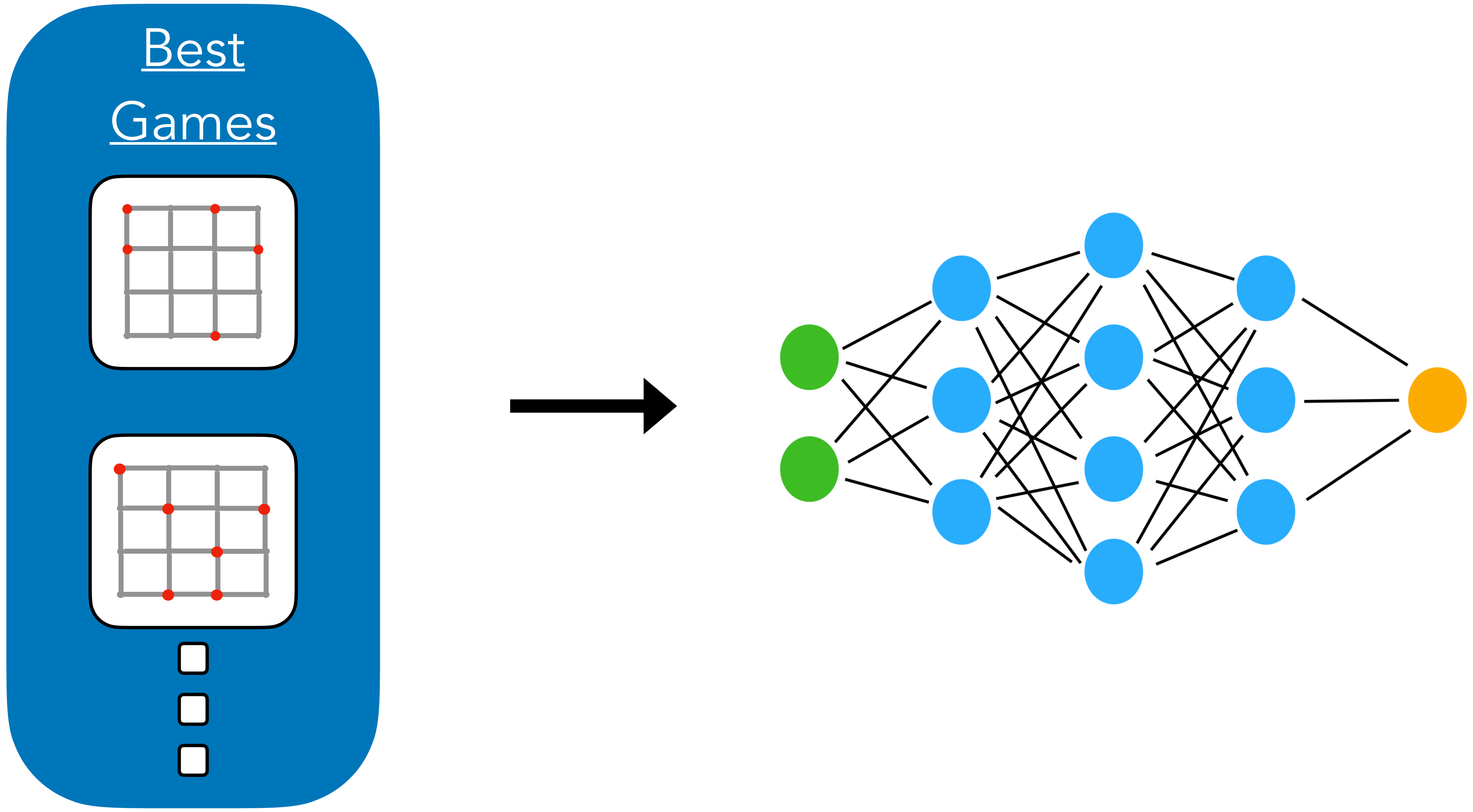
Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.

The evaluation

Sample lots of games (~2000) and assign a score to each one. Filter out the top $k\,\%$.

The Reinforcement Learning

Treat the best games as 'ground truth' and train the network on them.



Approach 1: Reinforcement Learning

The game

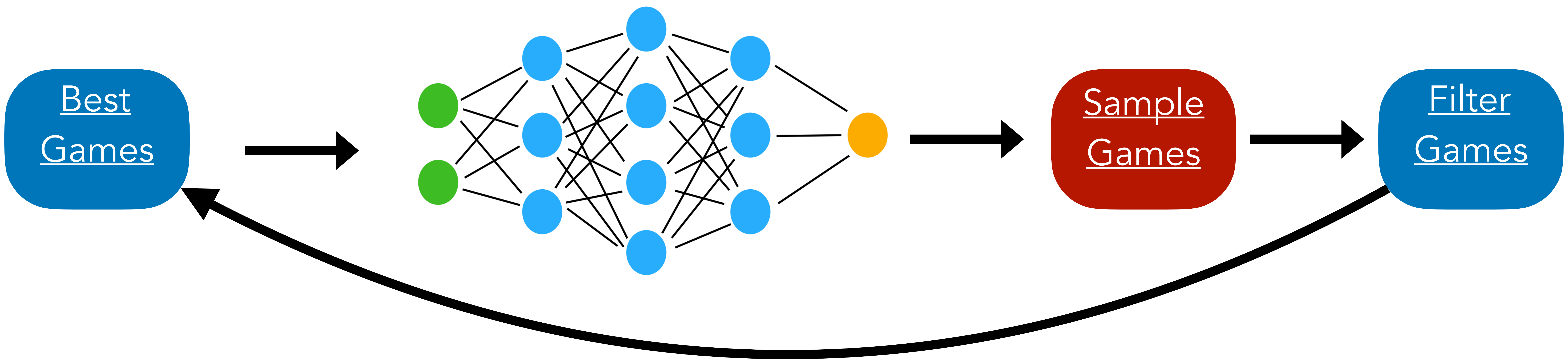
Define a neural network that takes in the current state of the grid (encoded as a vector) and an index to be considered on the grid and output a scalar p in $[0,1]$. Add the point with probability p. Repeat with the next index and so on until we have checked every index.

The evaluation

Sample lots of games (~2000) and assign a score to each one. Filter out the top $k\,\%$.

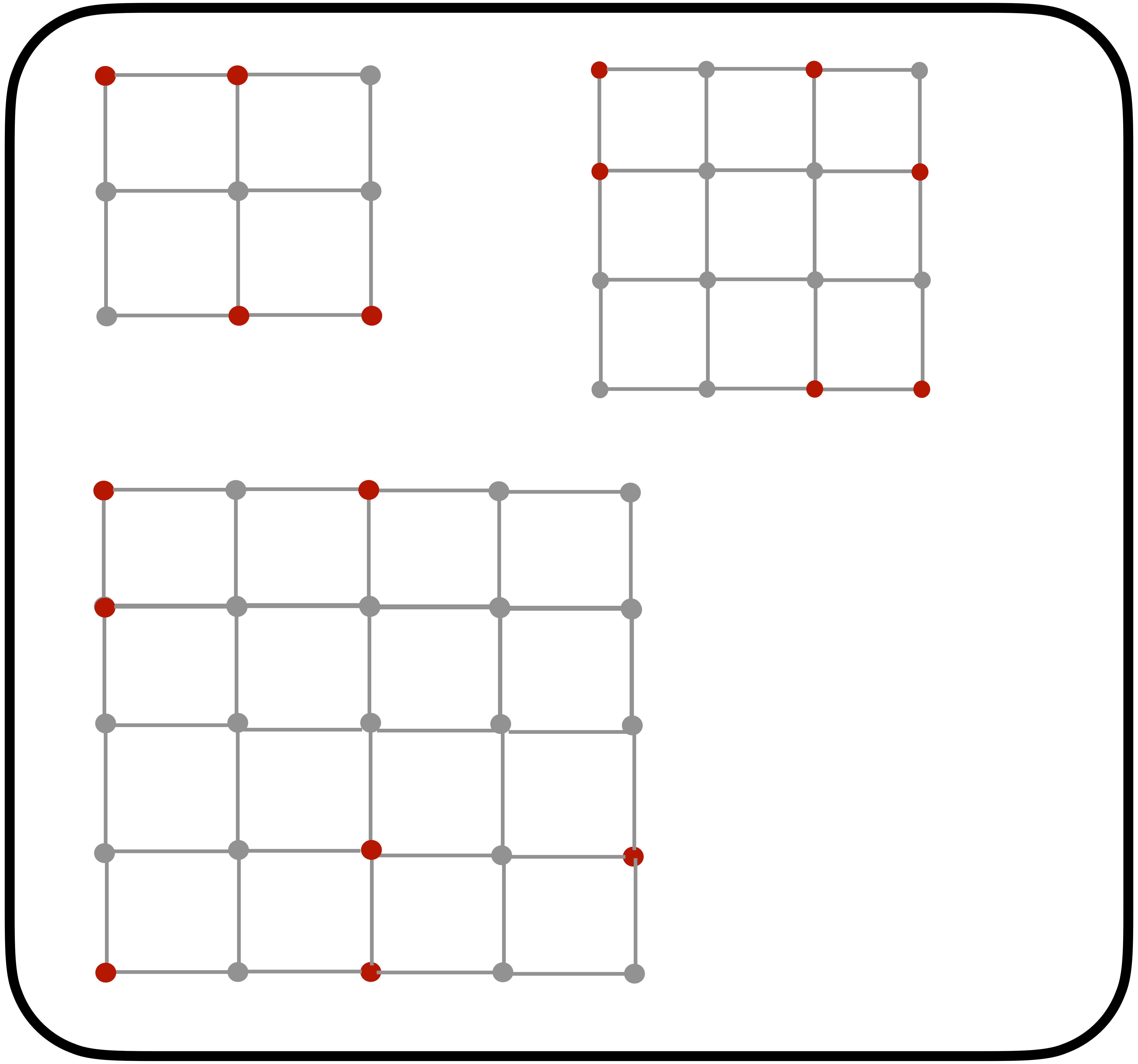
The Reinforcement Learning

Treat the best games as 'ground truth' and train the network on them.

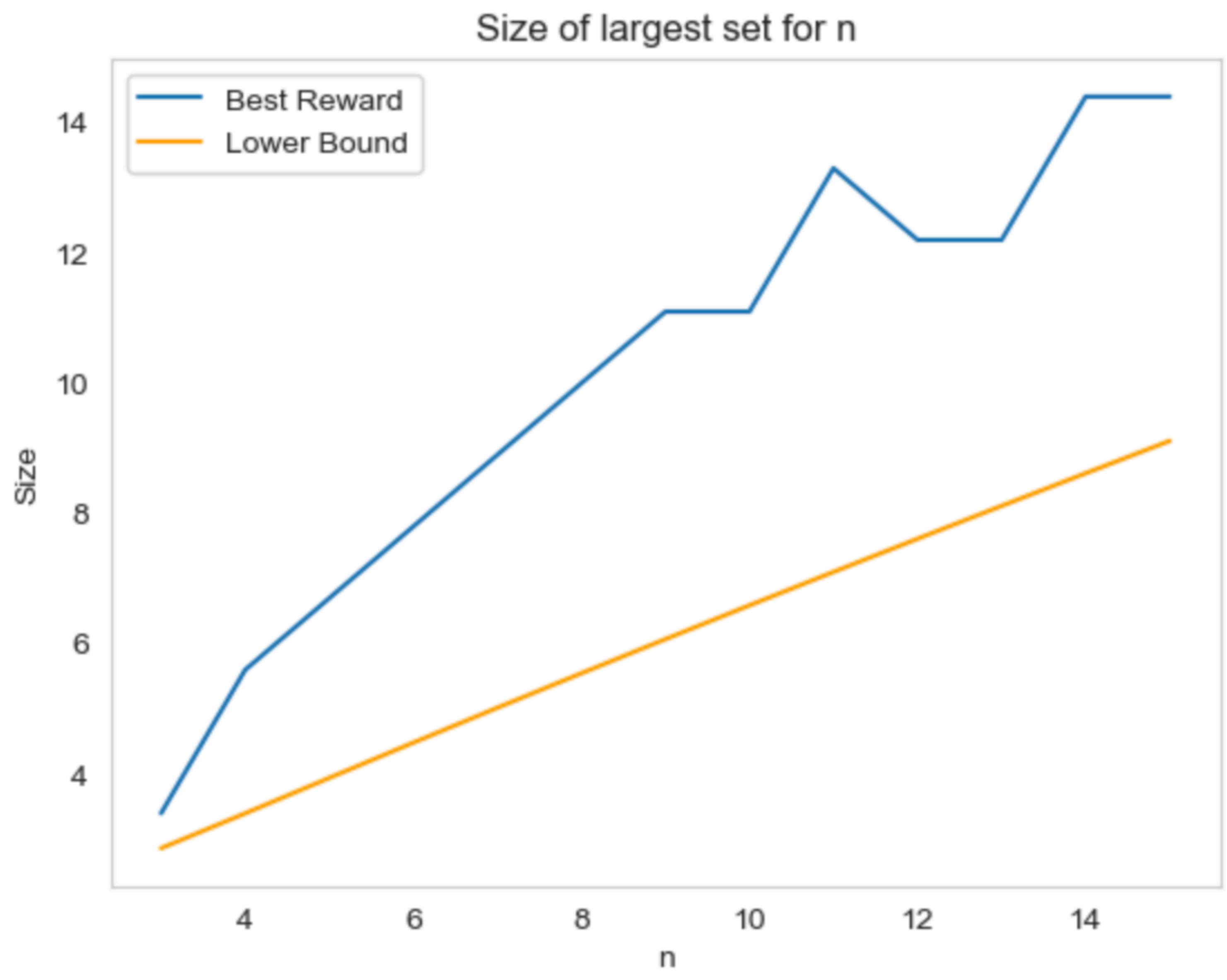


Approach 1: Reinforcement Learning

Results



N	3	4	5	6	10	15	64
Size	4	6	7	9	12	17	80



Seems better than known lower bound

$$\text{Size} > \frac{cN}{\sqrt{\log N}}$$

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 1

For any graph G with n vertices, we have,

$$\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$$

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

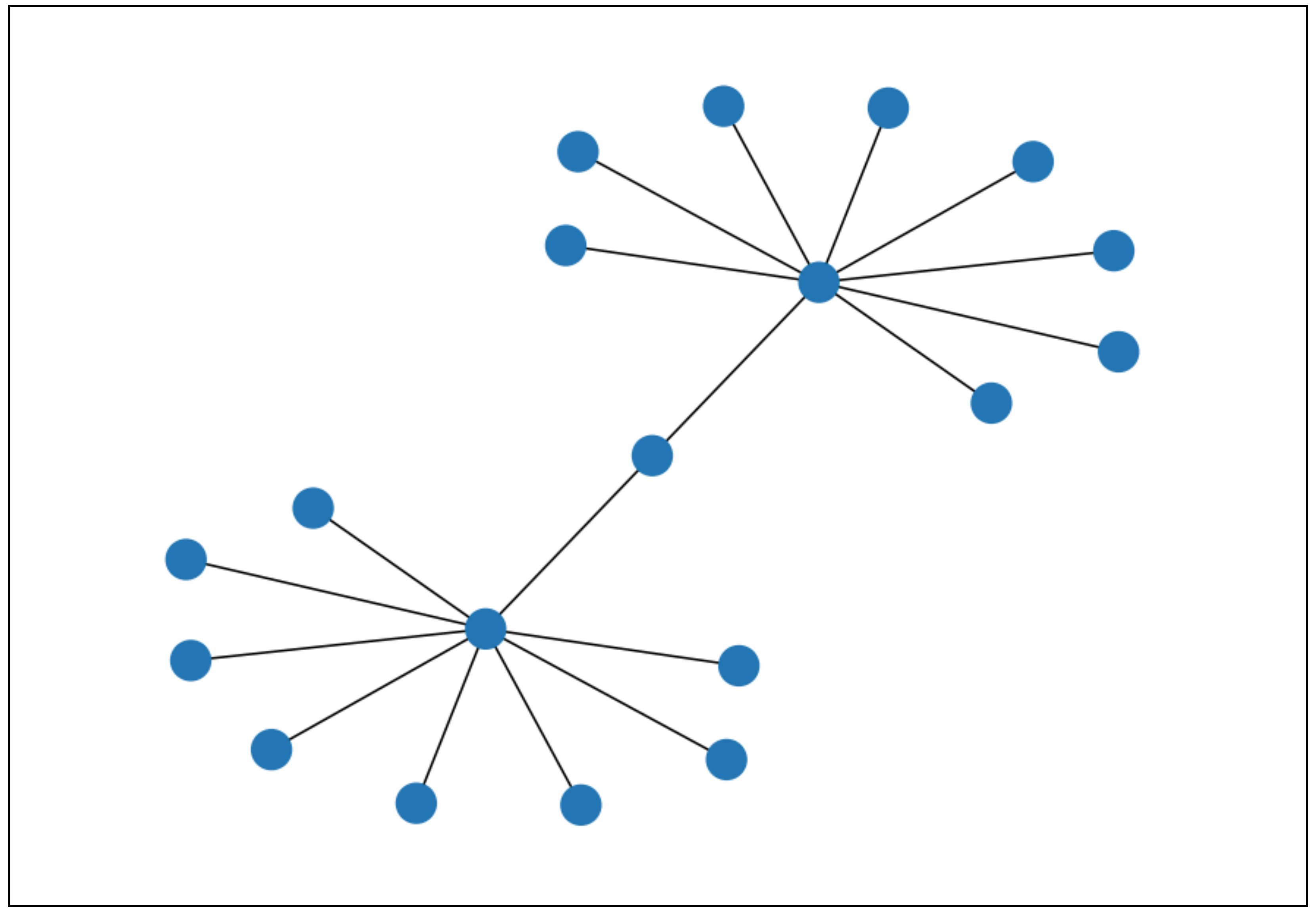
Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 1

For any graph G with n vertices, we have,

$$\lambda_1(G) + \mu(G) \geq \sqrt{n-1} + 1$$



Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

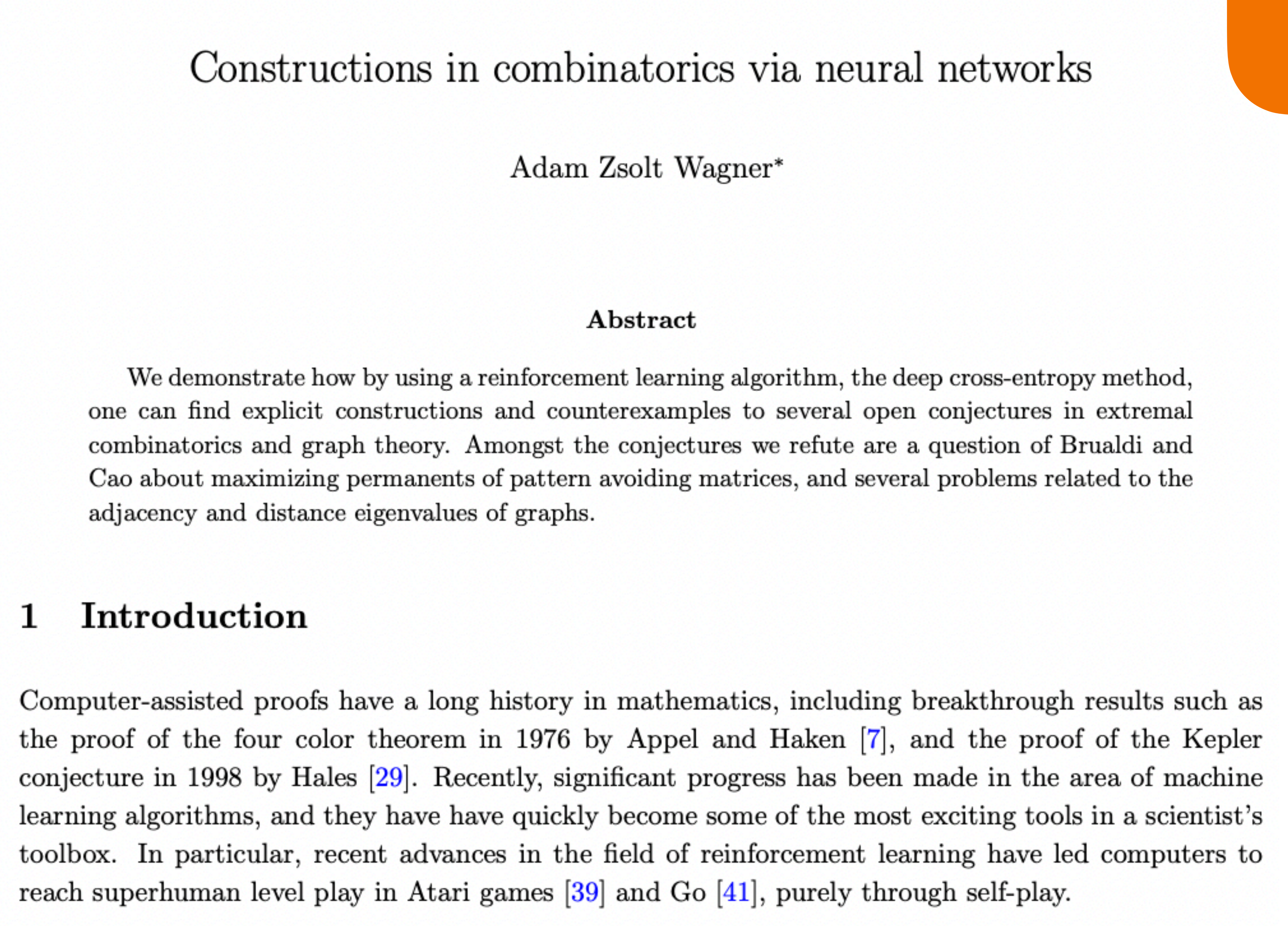
Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 2

For any graph G, $K_4(G) + K_4(\bar{G})$ is asymptotically minimized by random graphs.

Algorithm Background

Adapted from work by Adam Wagner[5]:



Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

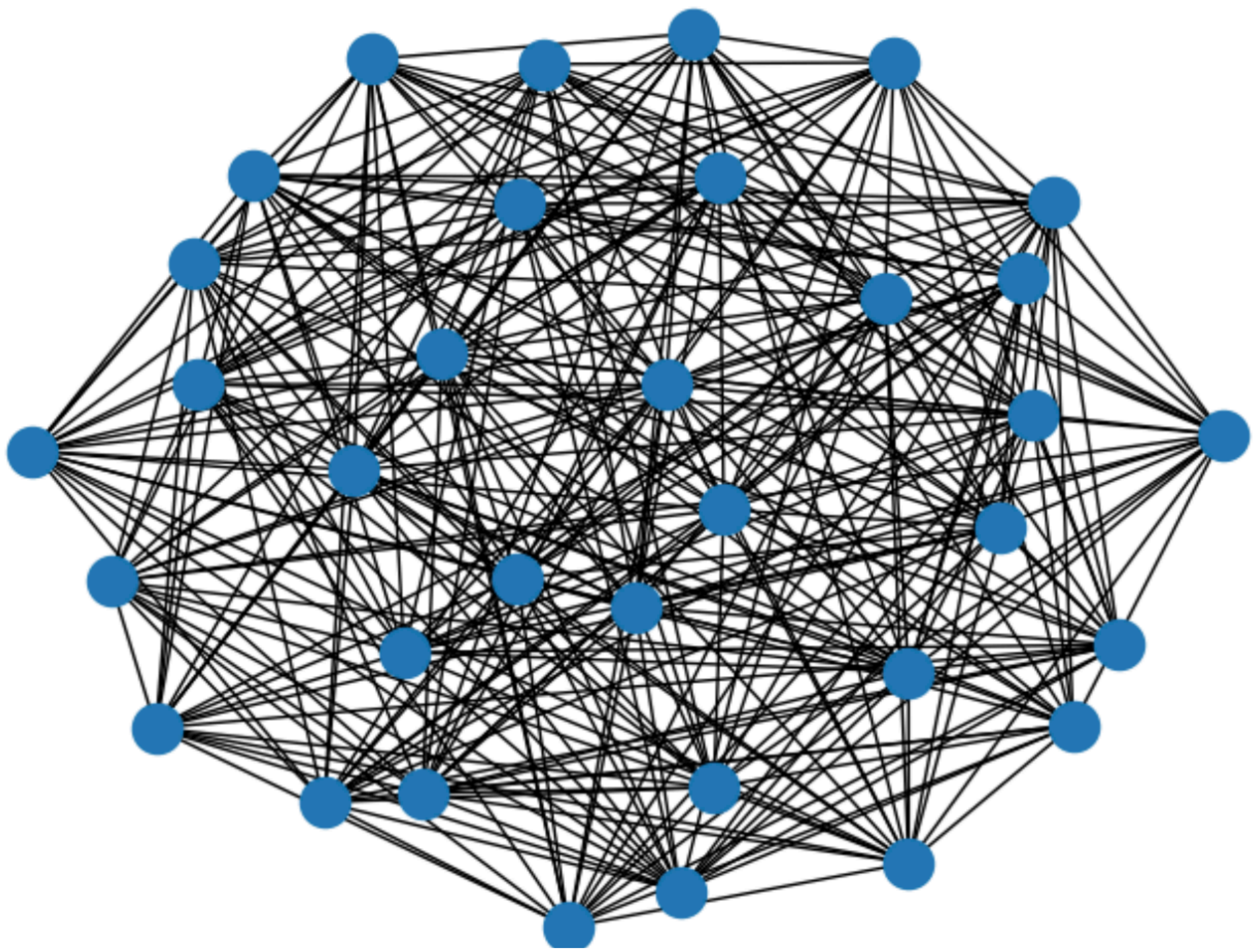
1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 2

For any graph G, $K_4(G) + K_4(\bar{G})$ is asymptotically minimized by random graphs.



Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 3

Let G be a graph with diameter D, proximity π, and distance spectrum $\partial_1 \geq \ldots \geq \partial_n$, then

$$\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} > 0$$

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

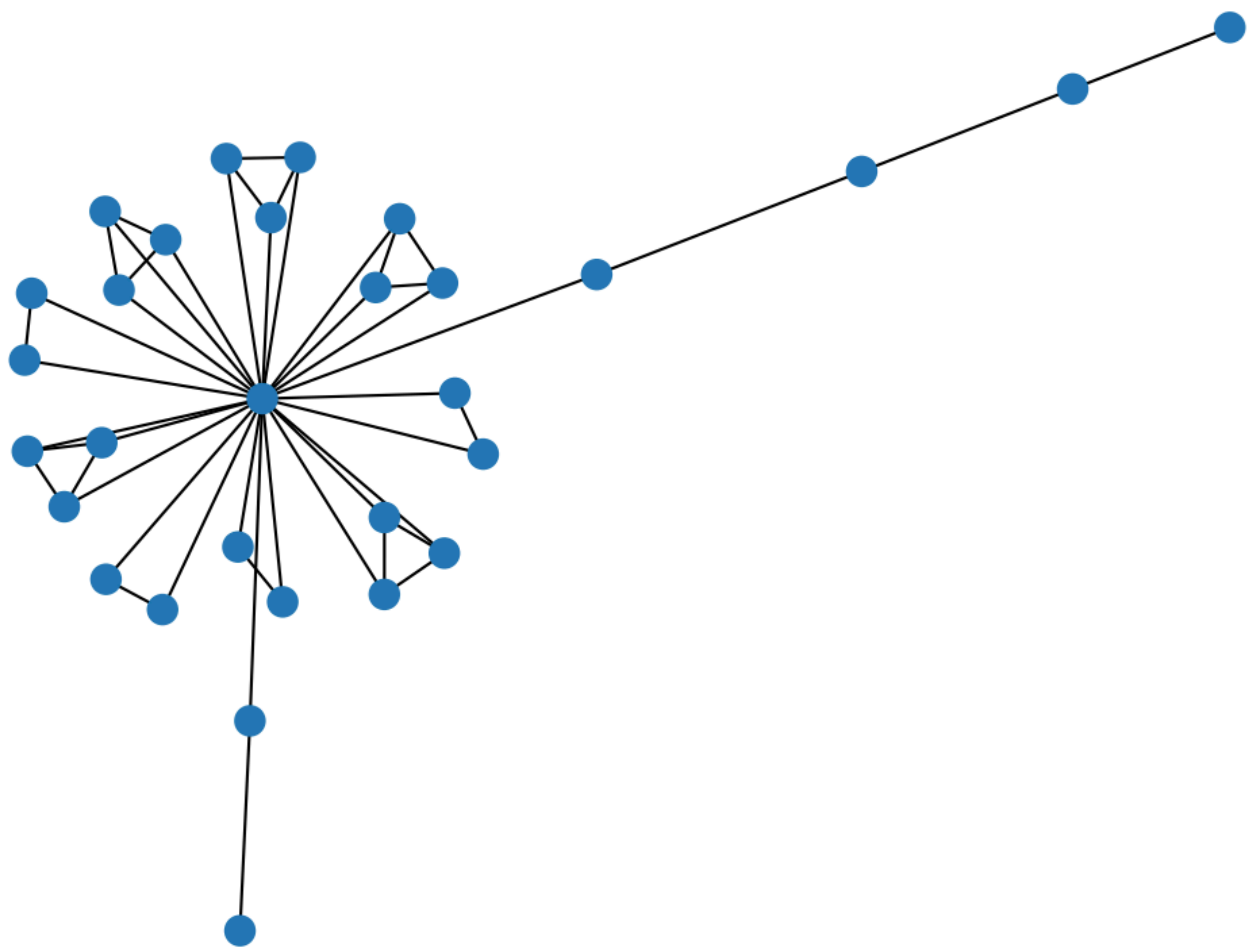
Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 3

Let G be a graph with diameter D, proximity π, and distance spectrum $\partial_1 \geq \ldots \geq \partial_n$, then

$$\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} > 0$$



Not a counterexample…..

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

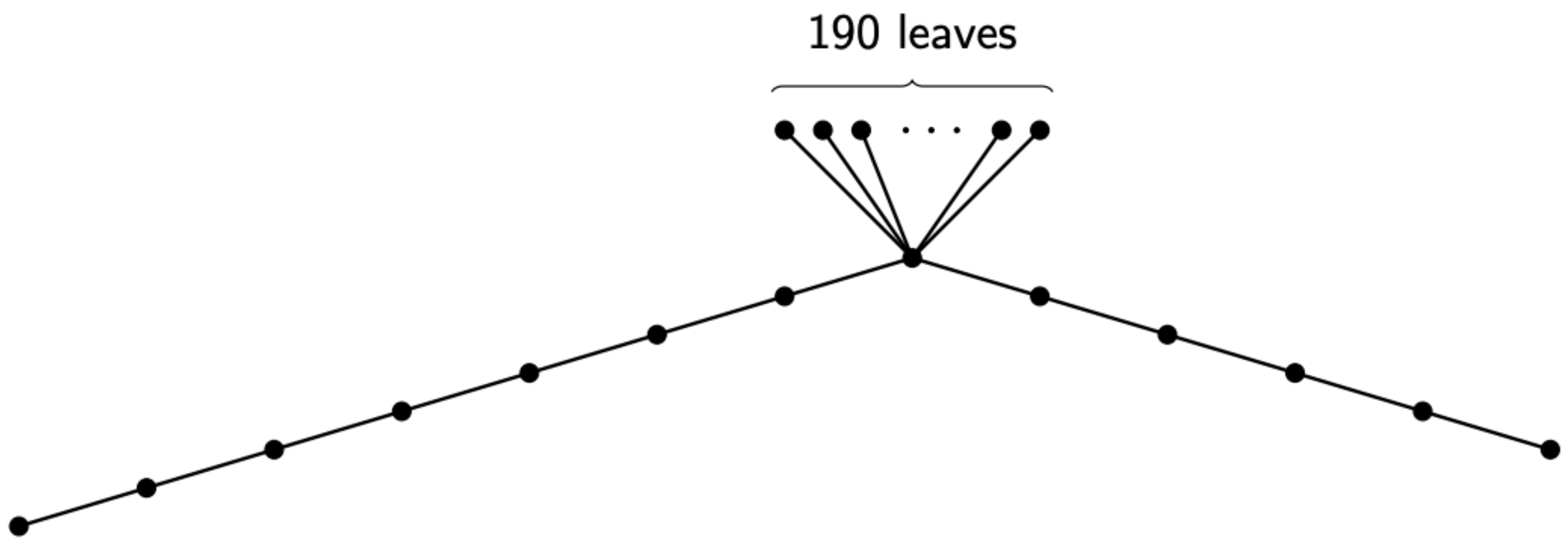
Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

Example Conjecture 3

Let G be a graph with diameter D, proximity π, and distance spectrum $\partial_1 \geq \ldots \geq \partial_n$, then

$$\pi + \partial_{\lfloor \frac{2D}{3} \rfloor} > 0$$



Not a counterexample….. but it leads to one

Algorithm Background

Adapted from work by Adam Wagner[5]:

Constructions in combinatorics via neural networks

Adam Zsolt Wagner*

Abstract

We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

1 Introduction

Computer-assisted proofs have a long history in mathematics, including breakthrough results such as the proof of the four color theorem in 1976 by Appel and Haken [7], and the proof of the Kepler conjecture in 1998 by Hales [29]. Recently, significant progress has been made in the area of machine learning algorithms, and they have have quickly become some of the most exciting tools in a scientist's toolbox. In particular, recent advances in the field of reinforcement learning have led computers to reach superhuman level play in Atari games [39] and Go [41], purely through self-play.

Aim: Use this algorithm to generate counterexamples to conjectures in combinatorics

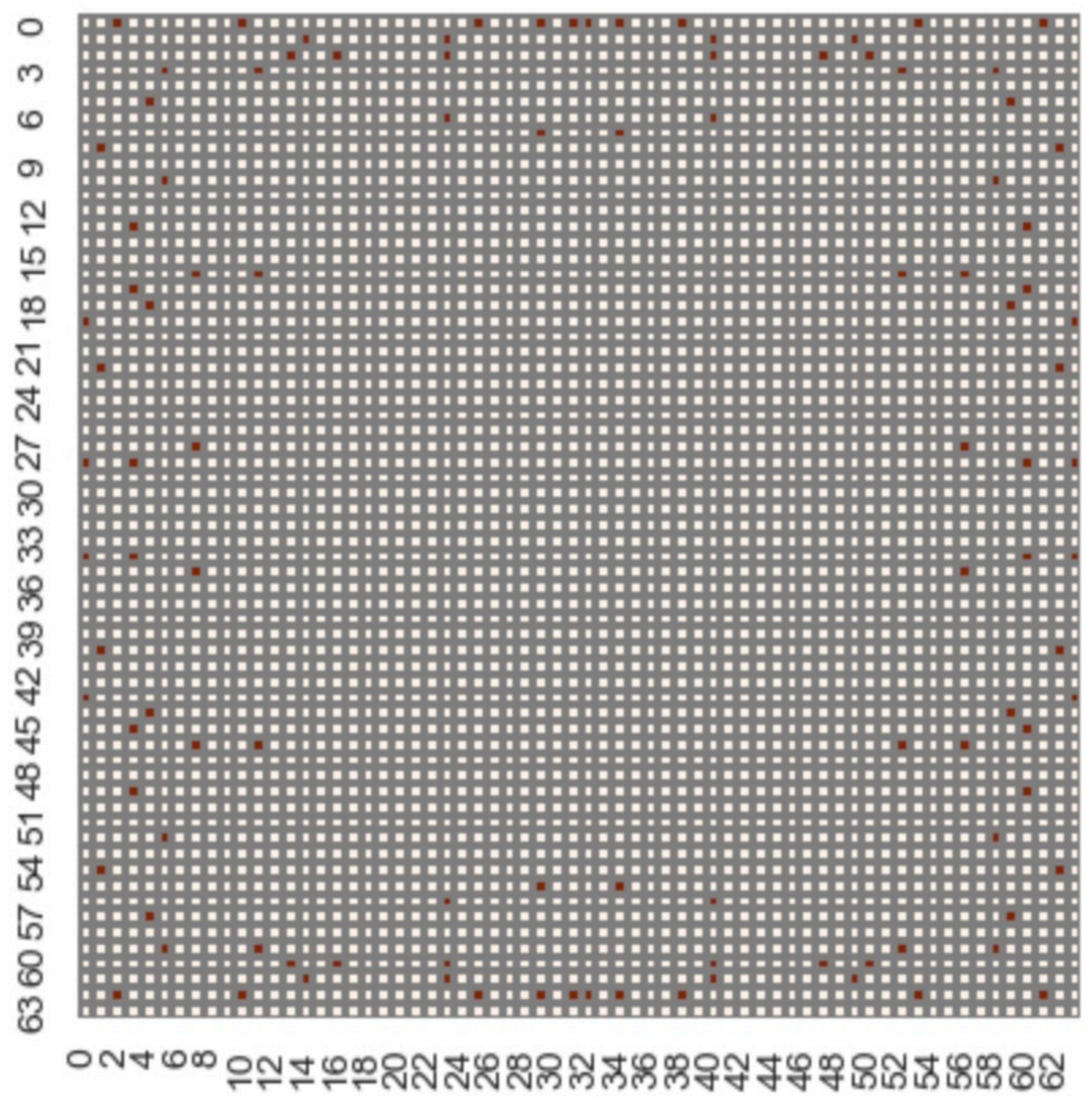
Immediate Counterexample

Not a Counterexample and / or not insightful

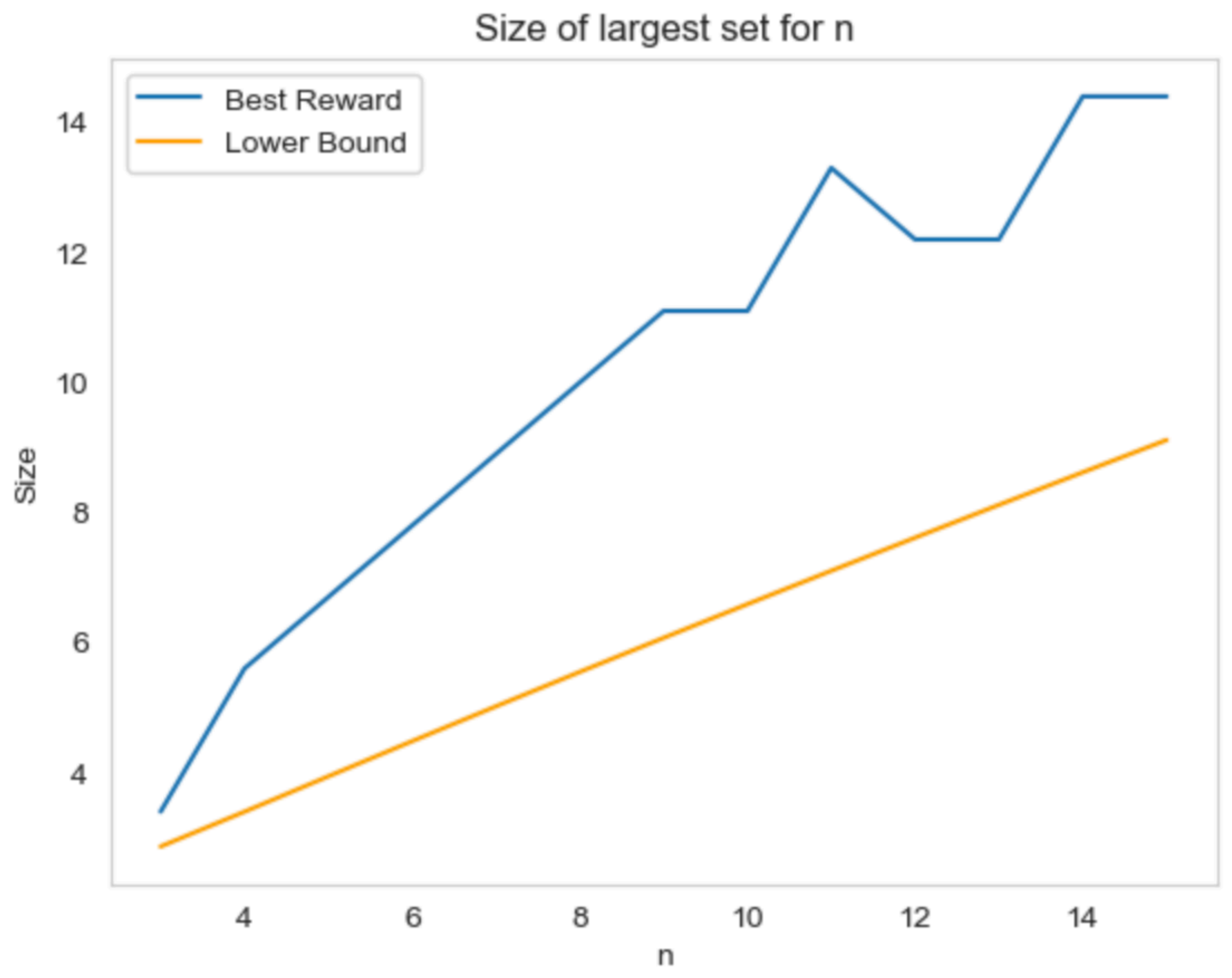
Almost a Counterexample
But was able to extend to counterexample

Approach 1: Reinforcement Learning

Results



N	3	4	5	6	10	15	64
Size	4	6	7	9	12	17	80

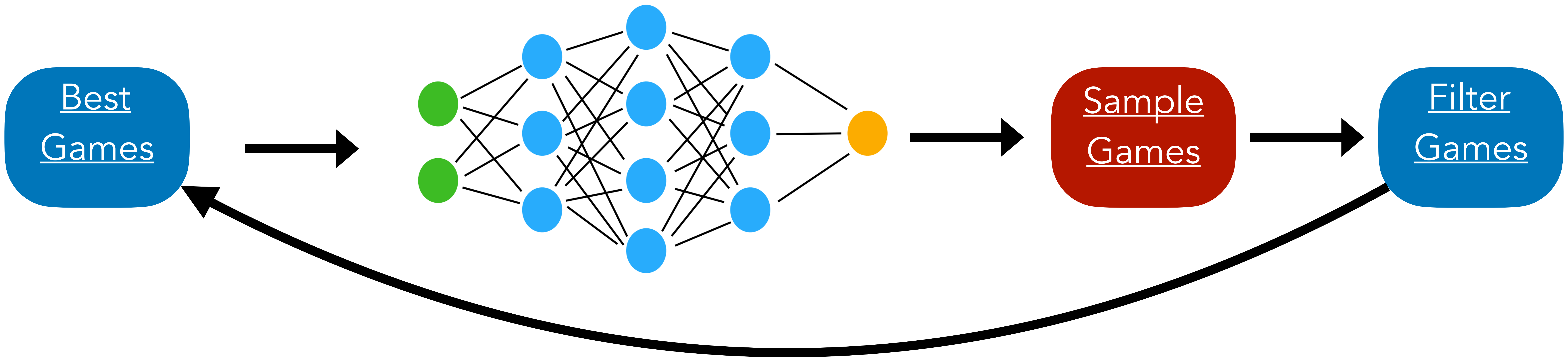


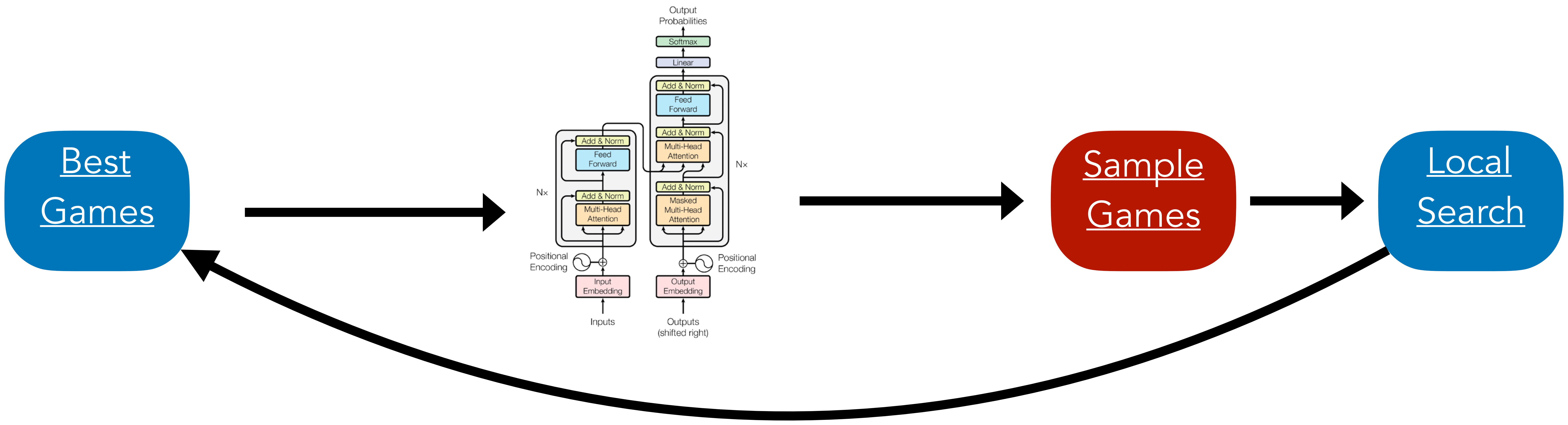
Seems better than known lower bound

Size $> \frac{cN}{\sqrt{\log N}}$

Approach 1: Reinforcement Learning

Alternative: Use a transformer + local search instead





Approach 1: Reinforcement Learning

Alternative: Use a transformer + local search instead[6]

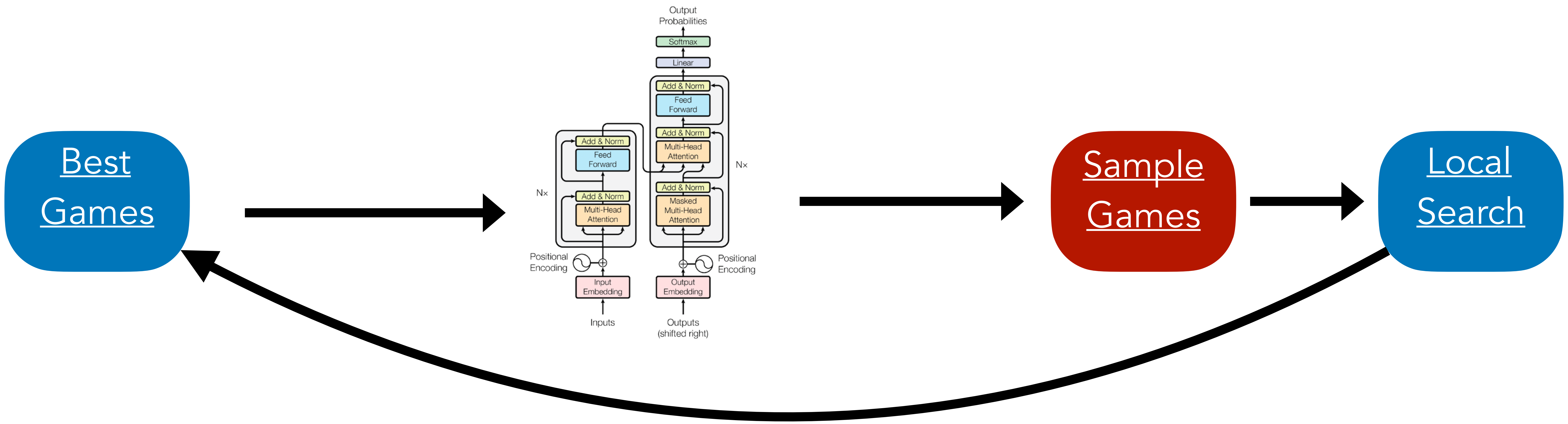
PatternBoost: Constructions in Mathematics with a Little Help from AI

François Charton* Jordan Ellenberg† Adam Zsolt Wagner‡
Geordie Williamson§

November 4, 2024

Abstract

We introduce PatternBoost, a flexible method for finding interesting constructions in mathematics. Our algorithm alternates between two phases. In the first "local" phase, a classical search algorithm is used to produce many desirable constructions. In the second "global" phase, a transformer neural network is trained on the best such constructions. Samples from the trained transformer are then used as seeds for the first phase, and the process is repeated. We give a detailed introduction to this technique, and discuss the results of its application to several problems in extremal combinatorics. The performance of PatternBoost varies across different problems, but there are many situations where its performance is quite impressive. Using our technique, we find the best known solutions to several long-standing problems, including the construction of a counterexample to a conjecture that had remained open for 30 years.



Approach 1: Reinforcement Learning

<u>**Alternative: Use a transformer + local search instead**</u>[6]

<u>**Results**</u>[6]

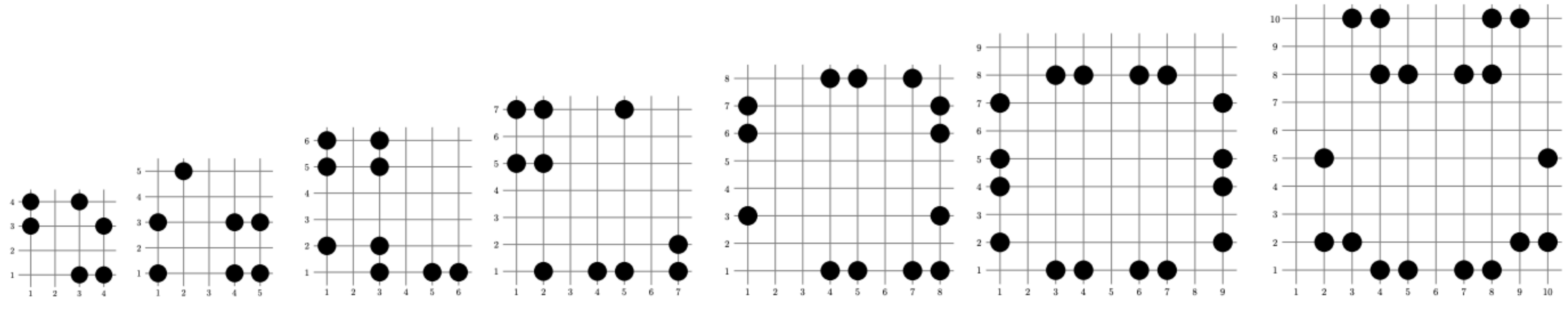


Figure 11: The best constructions for $n = 4$ to 10

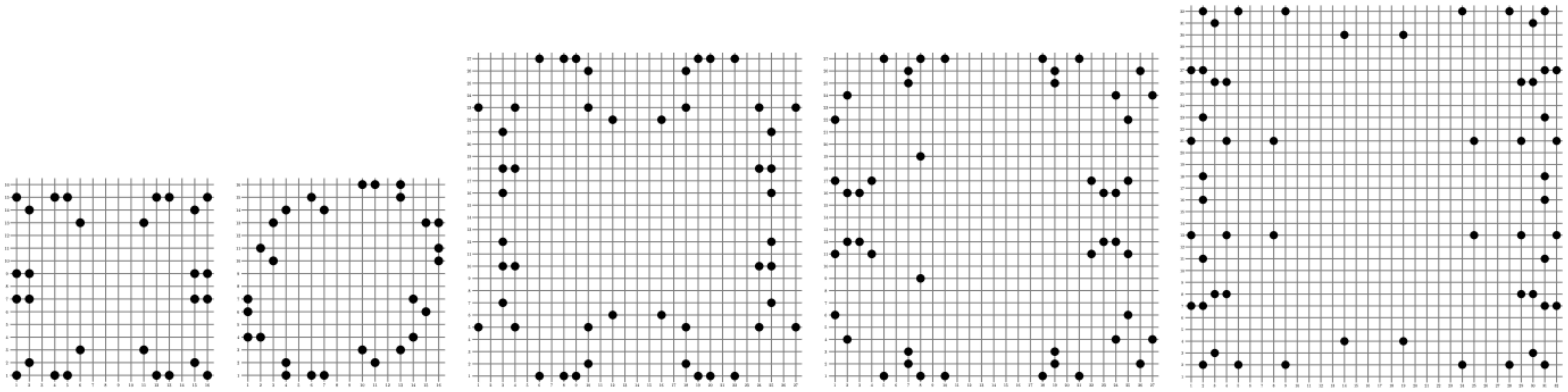
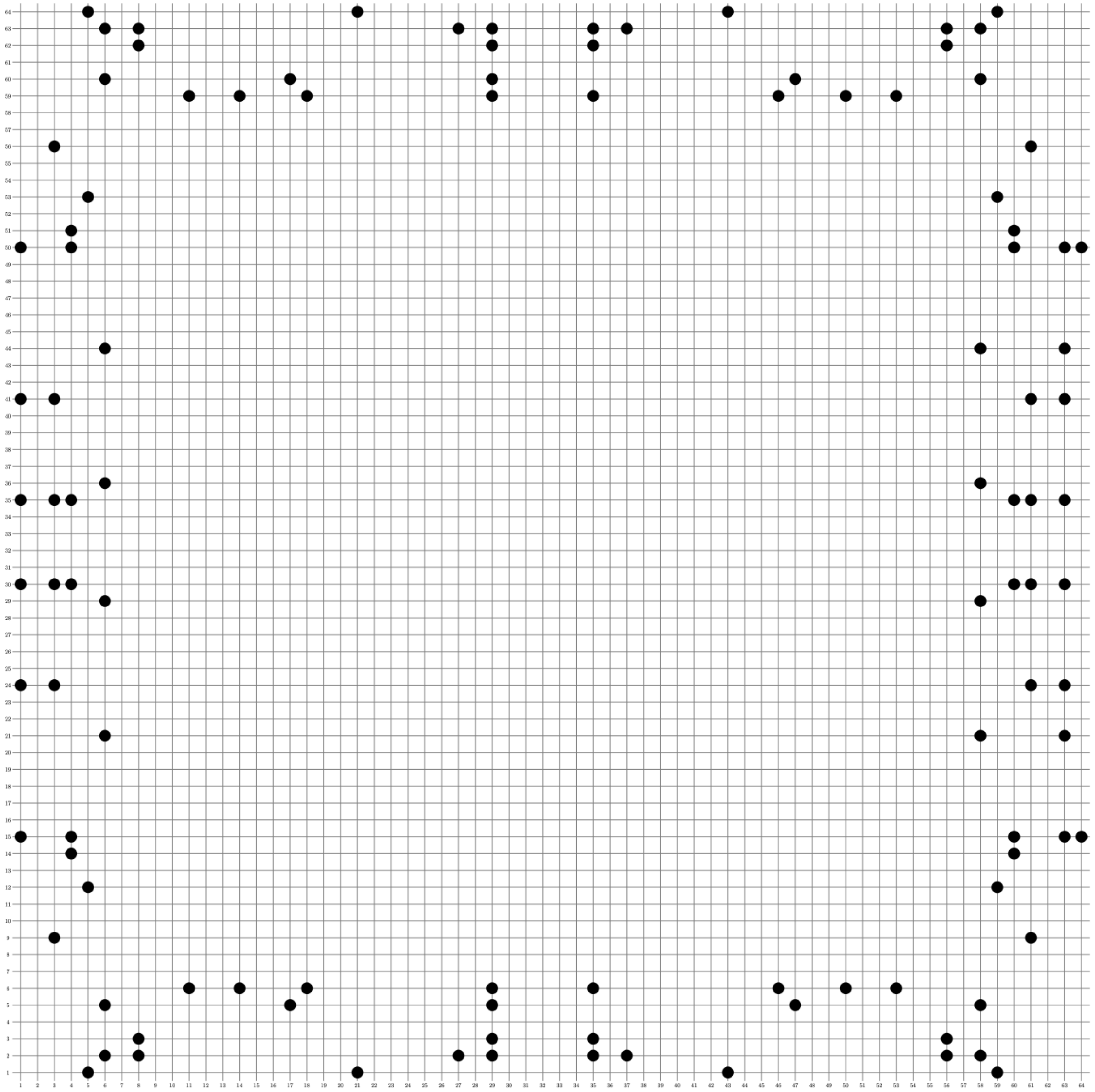


Figure 12: The best constructions for $n = 16$, 27, and 32. For some n, there are many optimal solutions that look very different from each other.

Approach 1: Reinforcement Learning

Alternative: Use a transformer + local search instead[6]

Results[6]



N=64

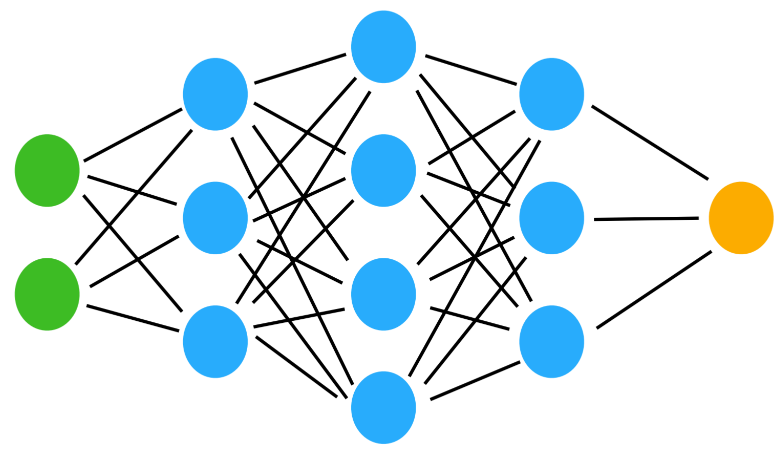
Size = 108

Drawback: These results are great, but they are hard to draw insights from!

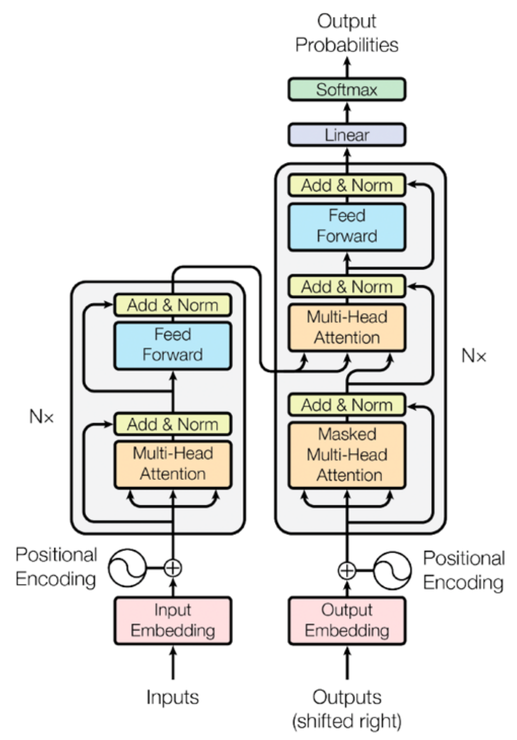
Traditional Neural Approaches have drawbacks.

Data-Driven Methods

Neural Networks



Transformers

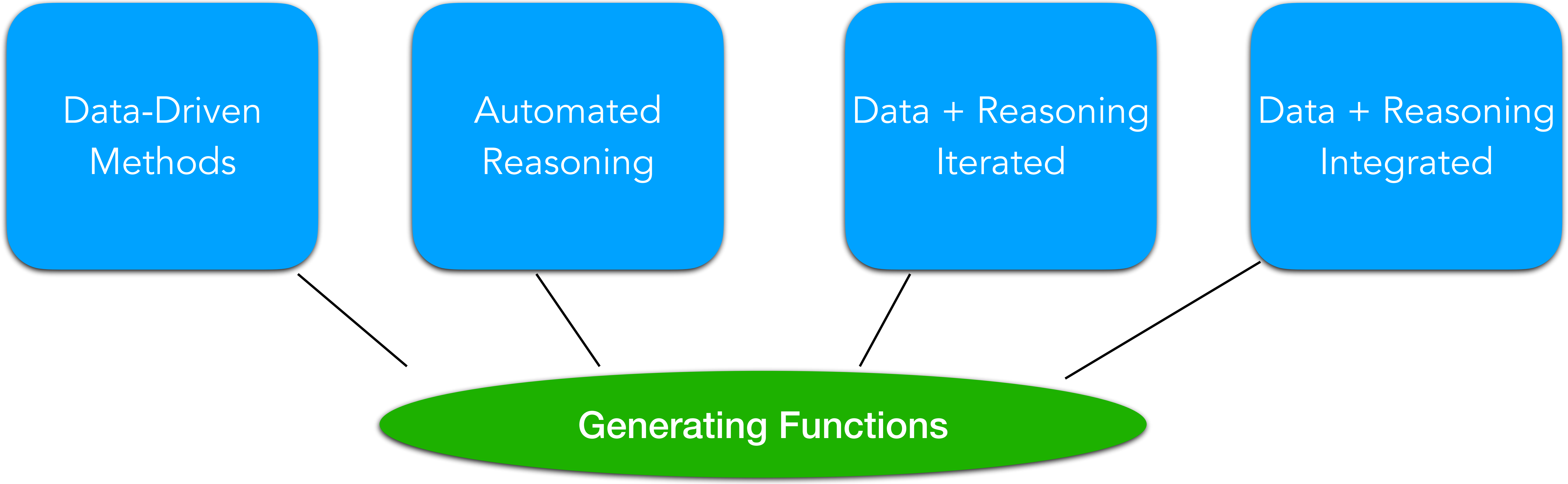


Drawback

1. Results are usually less interpretable. The mechanism behind the generation is difficult to understand.

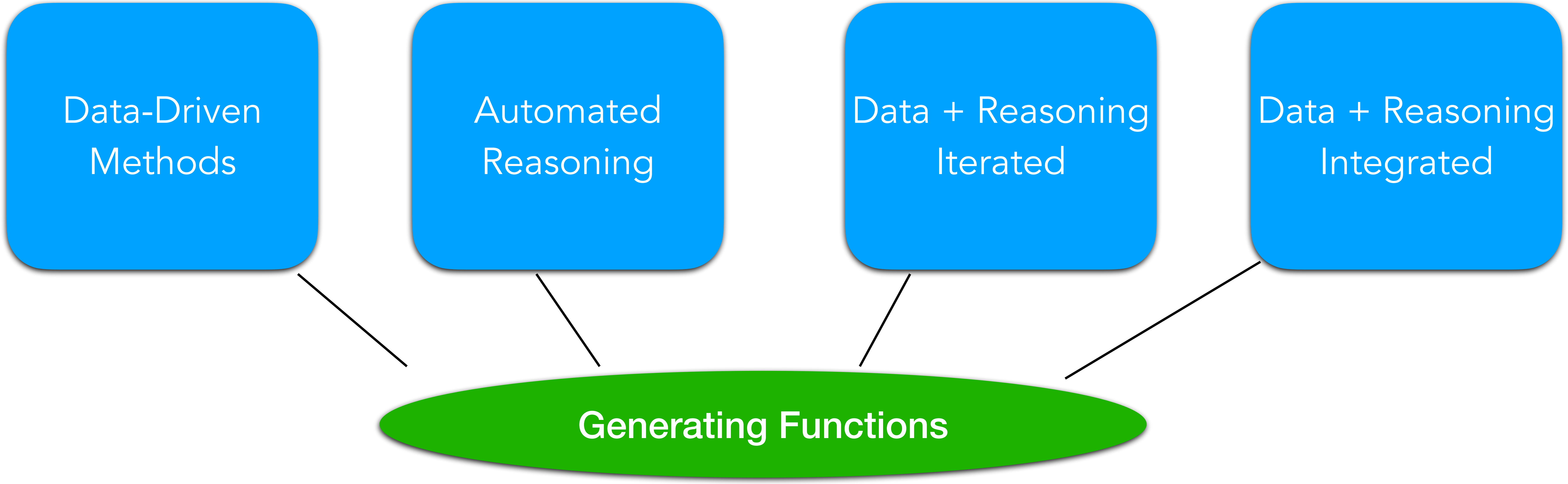
2. For different input sizes, we might have to train a new model. So difficult to test generalization.

Change of Approach: Functions to Algorithms

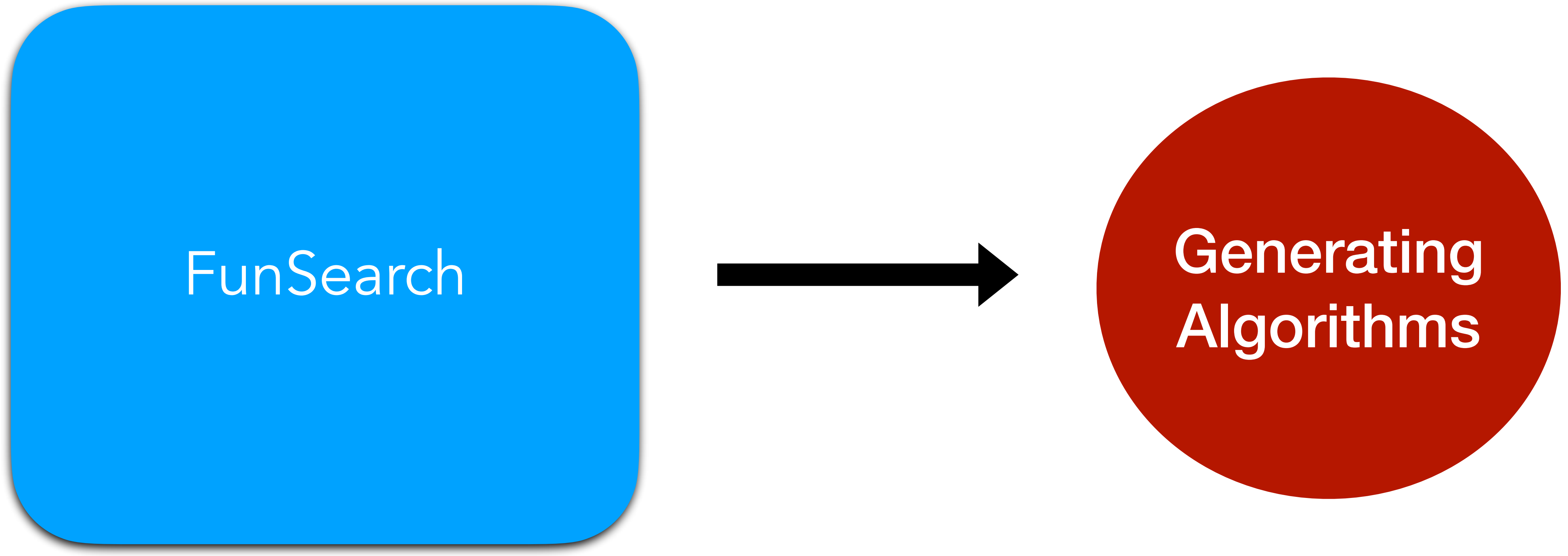


The methods we have looked at are approaches for generating functions. Instead of generating functions, we will focus on generating algorithms.

Change of Approach: Functions to Algorithms



The methods we have looked at are approaches for generating functions. Instead of generating functions, we will focus on generating algorithms.



Aim: Train a large language model to generate code that we can use to construct examples.

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature > articles > article

Article | Open access | Published: 14 December 2023

Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes ✉, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli ✉ & Alhussein Fawzi ✉

Nature **625**, 468–475 (2024) | Cite this article

Abstract

Large language models (LLMs) have demonstrated tremendous capabilities in solving complex tasks, from quantitative reasoning to understanding natural language. However, LLMs sometimes suffer from confabulations (or hallucinations), which can result in them making plausible but incorrect statements[1,2]. This hinders the use of current large models in scientific discovery. Here we introduce FunSearch (short for searching in the function space),

For the purposes of this talk, we will follow the work on FunSearch, by DeepMind[8]

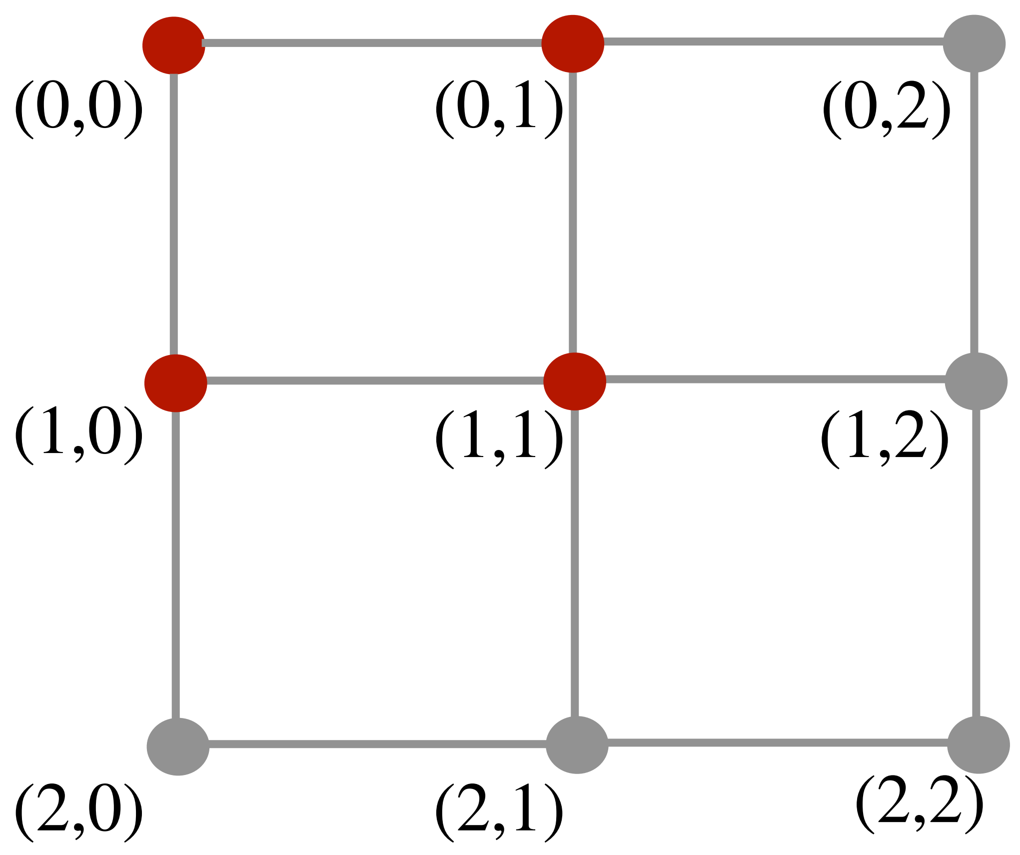
Algorithms and LLMs

Given the Capset problem, we can represent the solution to the problem in two ways.

Algorithms and LLMs

Given the Capset problem, we can represent the solution to the problem in two ways.

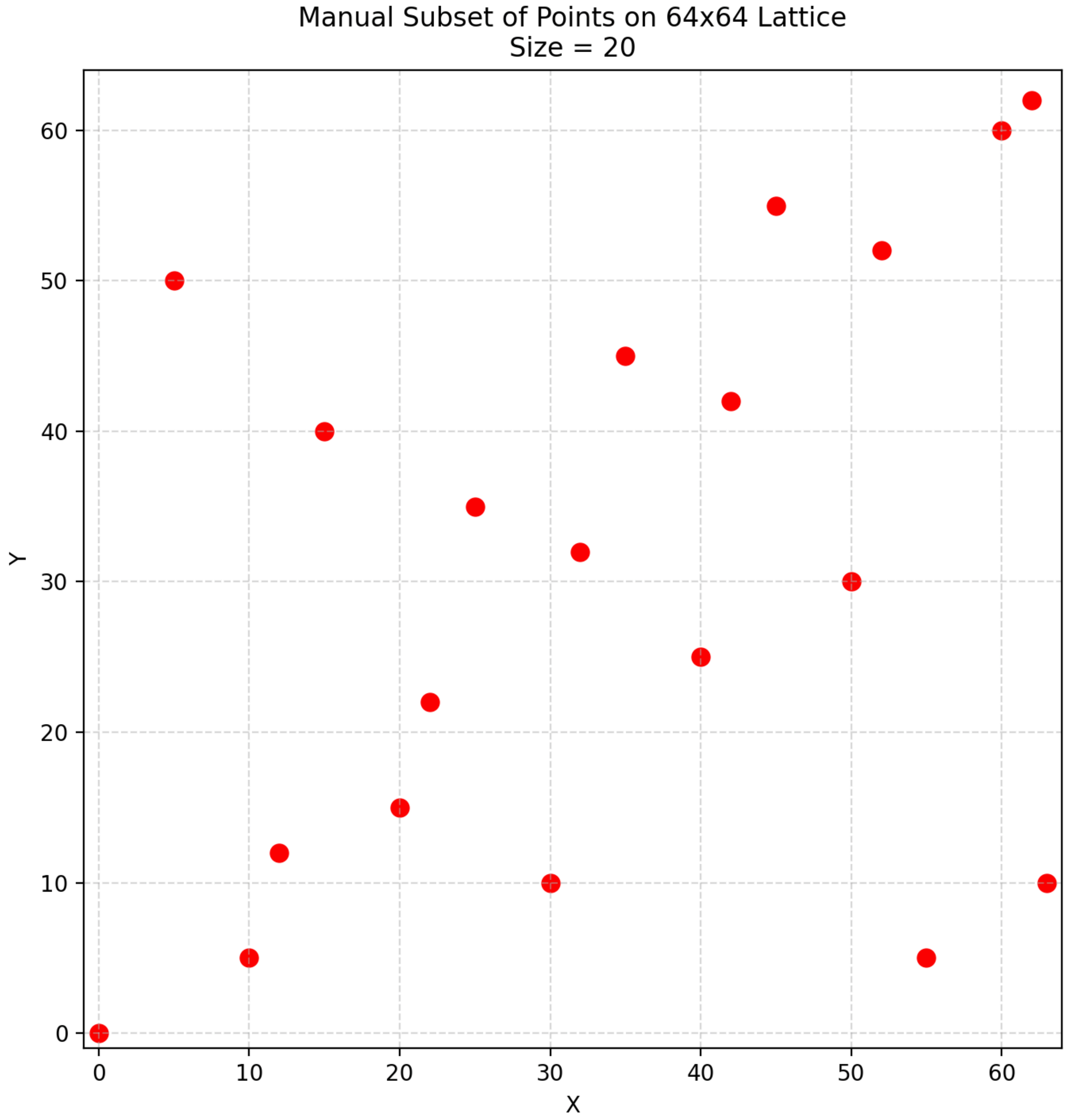
Via a set of points



Algorithms and LLMs

Need to be careful: LLMs by themselves cannot solve complex reasoning tasks. Solutions may not be plausible / might hallucinate

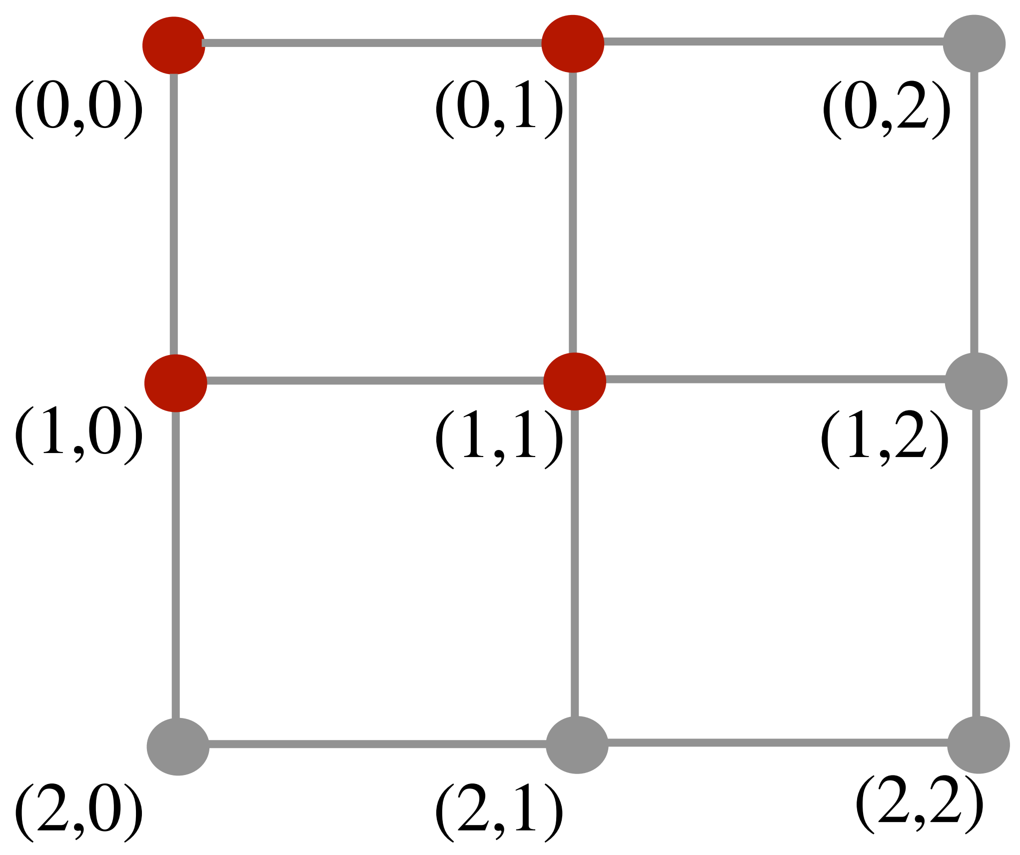
GPT4o attempt at solving the isosceles free problem for n=64 with chain of thought reasoning.



Algorithms and LLMs

Given the Capset problem, we can represent the solution to the problem in two ways.

Via a set of points



Via an algorithm

```
def build_cap_set(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset(elem):
    return (elem[0] == 0 and elem[1] <= 1)
```

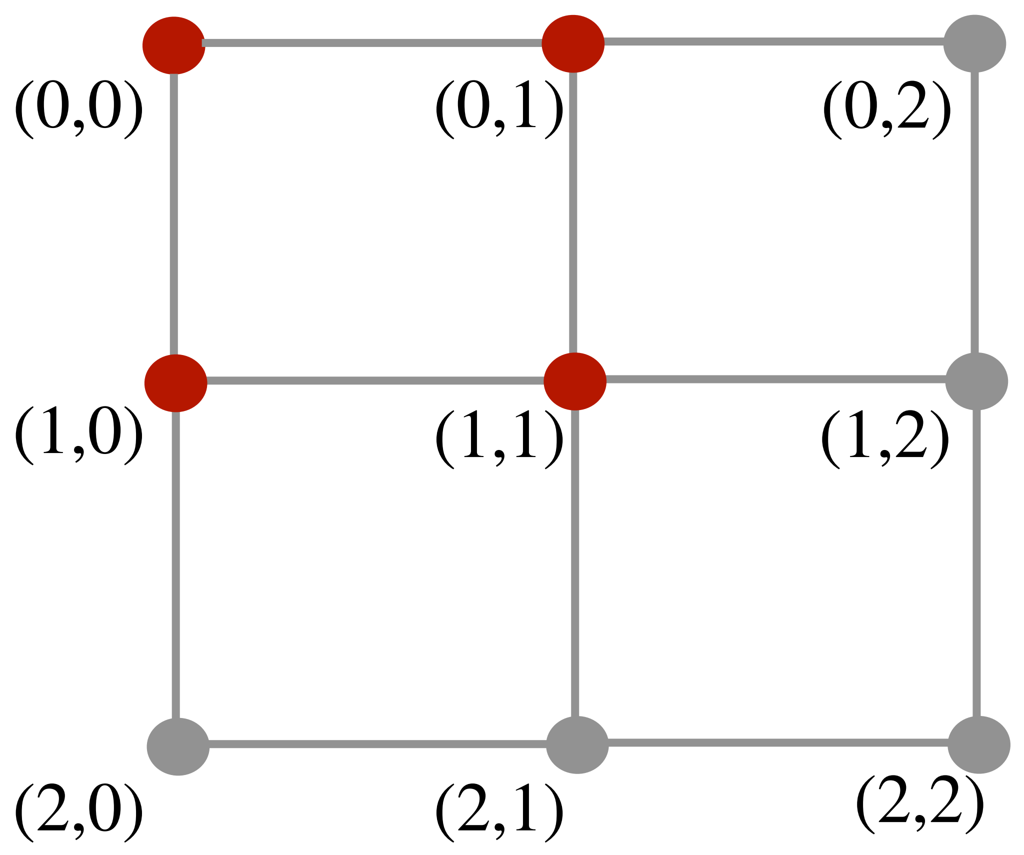
Advantage of expressing as an algorithm:

1. We can verify the correctness of an algorithm quickly (evaluation is easy)

Algorithms and LLMs

Given the Capset problem, we can represent the solution to the problem in two ways.

Via a set of points



Via an algorithm

```python
def build_cap_set(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset(elem):
    return (elem[0] == 0 and elem[1] <= 1)
```

Advantage of expressing as an algorithm:

1. We can verify the correctness of an algorithm quickly (evaluation is easy)

2. We can test the generating mechanism on cases out of the training distribution (can study generalization)

Algorithms and LLMs

Key tool for generating code: Large Language Models.

Large Language Models have shown widespread success in code generation. Example below: Evaluation of various models on the HumanEval and MBPP benchmarks[6].

Model	Size	HumanEval			MBPP		
		pass@1	pass@10	pass@100	pass@1	pass@10	pass@100
code-cushman-001	12B	33.5%	-	-	45.9%	-	-
GPT-3.5 (ChatGPT)	-	48.1%	-	-	52.2%	-	-
GPT-4	-	67.0%	-	-	-	-	-
PaLM	540B	26.2%	-	-	36.8%	-	-
PaLM-Coder	540B	35.9%	-	88.4%	47.0%	-	-
PaLM 2-S	-	37.6%	-	88.4%	50.0%	-	-
StarCoder Base	15.5B	30.4%	-	-	49.0%	-	-
StarCoder Python	15.5B	33.6%	-	-	52.7%	-	-
StarCoder Prompted	15.5B	40.8%	-	-	49.5%	-	-
Llama 2	7B	12.2%	25.2%	44.4%	20.8%	41.8%	65.5%
	13B	20.1%	34.8%	61.2%	27.6%	48.1%	69.5%
	34B	22.6%	47.0%	79.5%	33.8%	56.9%	77.6%
	70B	30.5%	59.4%	87.0%	45.4%	66.2%	83.1%
Code Llama	7B	33.5%	59.6%	85.9%	41.4%	66.7%	82.5%
	13B	36.0%	69.4%	89.8%	47.0%	71.7%	87.1%
	34B	48.8%	76.8%	93.0%	55.0%	76.2%	86.6%
	70B	53.0%	84.6%	96.2%	62.4%	81.1%	91.9%
Code Llama - Instruct	7B	34.8%	64.3%	88.1%	44.4%	65.4%	76.8%
	13B	42.7%	71.6%	91.6%	49.4%	71.2%	84.1%
	34B	41.5%	77.2%	93.5%	57.0%	74.6%	85.4%
	70B	67.8%	90.3%	97.3%	62.2%	79.6%	89.2%
Unnatural Code Llama	34B	62.2%	85.2%	95.4%	61.2%	76.6%	86.7%
Code Llama - Python	7B	38.4%	70.3%	90.6%	47.6%	70.3%	84.8%
	13B	43.3%	77.4%	94.1%	49.0%	74.0%	87.6%
	34B	53.7%	82.8%	94.7%	56.2%	76.4%	88.2%
	70B	57.3%	89.3%	98.4%	65.6%	81.5%	91.9%

Algorithms and LLMs

Key tool for generating code: Large Language Models.

Large Language Models have shown widespread success in code generation. Example below: Evaluation of various models on the HumanEval and MBPP benchmarks[6].

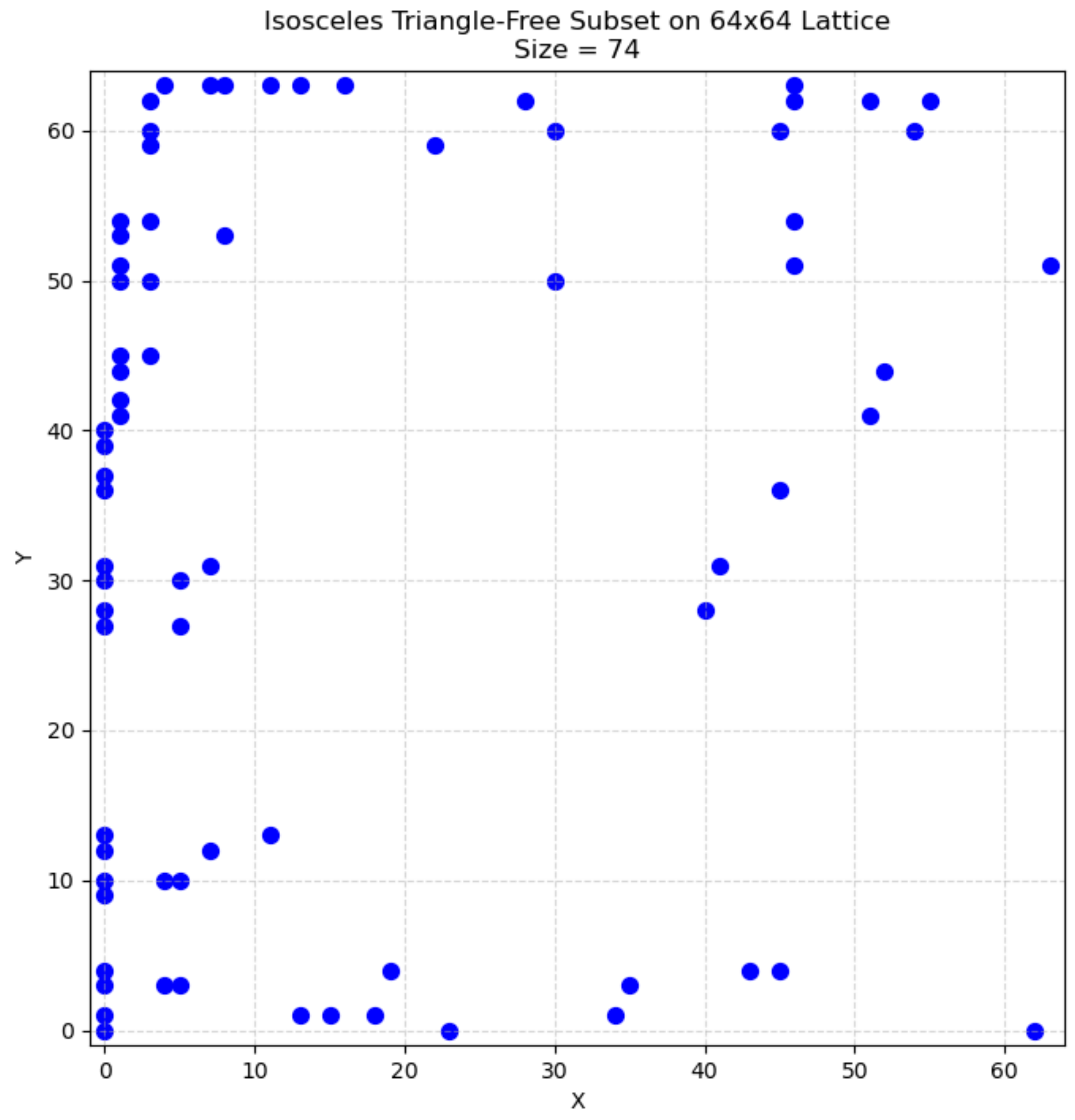
Model	Size	HumanEval			MBPP		
		pass@1	pass@10	pass@100	pass@1	pass@10	pass@100
code-cushman-001	12B	33.5%	-	-	45.9%	-	-
GPT-3.5 (ChatGPT)	-	48.1%	-	-	52.2%	-	-
GPT-4	-	67.0%	-	-	-	-	-
PaLM	540B	26.2%	-	-	36.8%	-	-
PaLM-Coder	540B	35.9%	-	88.4%	47.0%	-	-
PaLM 2-S	-	37.6%	-	88.4%	50.0%	-	-
StarCoder Base	15.5B	30.4%	-	-	49.0%	-	-
StarCoder Python	15.5B	33.6%	-	-	52.7%	-	-
StarCoder Prompted	15.5B	40.8%	-	-	49.5%	-	-
Llama 2	7B	12.2%	25.2%	44.4%	20.8%	41.8%	65.5%
	13B	20.1%	34.8%	61.2%	27.6%	48.1%	69.5%
	34B	22.6%	47.0%	79.5%	33.8%	56.9%	77.6%
	70B	30.5%	59.4%	87.0%	45.4%	66.2%	83.1%
Code Llama	7B	33.5%	59.6%	85.9%	41.4%	66.7%	82.5%
	13B	36.0%	69.4%	89.8%	47.0%	71.7%	87.1%
	34B	48.8%	76.8%	93.0%	55.0%	76.2%	86.6%
	70B	53.0%	84.6%	96.2%	62.4%	81.1%	91.9%
Code Llama - Instruct	7B	34.8%	64.3%	88.1%	44.4%	65.4%	76.8%
	13B	42.7%	71.6%	91.6%	49.4%	71.2%	84.1%
	34B	41.5%	77.2%	93.5%	57.0%	74.6%	85.4%
	70B	67.8%	90.3%	97.3%	62.2%	79.6%	89.2%
Unnatural Code Llama	34B	62.2%	85.2%	95.4%	61.2%	76.6%	86.7%
Code Llama - Python	7B	38.4%	70.3%	90.6%	47.6%	70.3%	84.8%
	13B	43.3%	77.4%	94.1%	49.0%	74.0%	87.6%
	34B	53.7%	82.8%	94.7%	56.2%	76.4%	88.2%
	70B	57.3%	89.3%	98.4%	65.6%	81.5%	91.9%

So…. can we just get GPT to write us a script to solve it?

Algorithms and LLMs

Need to be careful: LLMs by themselves cannot just solve complex reasoning tasks.

```
import itertools

def squared_dist(p1, p2):
    return (p1[0] - p2[0])**2 + (p1[1] - p2[1])**2

def forms_isosceles(p, subset):
    # Check if adding point p to subset creates any isosceles triangle
    # Iterate all pairs in subset and check distances
    for a, b in itertools.combinations(subset, 2):
        d1 = squared_dist(p, a)
        d2 = squared_dist(p, b)
        d3 = squared_dist(a, b)
        # Check if any two sides are equal (including zero distance means points coincide)
        # Skip degenerate triangles where points coincide (distance zero)
        sides = [d1, d2, d3]
        if 0 in sides:
            continue
        if d1 == d2 or d1 == d3 or d2 == d3:
            return True
    return False

def generate_isosceles_free_subset(n):
    lattice_points = [(x, y) for x in range(n) for y in range(n)]
    subset = []
    for p in lattice_points:
        if not forms_isosceles(p, subset):
            subset.append(p)
    return subset
```



GPT4o one-shot attempt at writing code to generate an algorithm that generates isosceles-free sets. While a human might not have written that example by hand, the code shows that it's a greedy algorithm! One could have certainly done that!

Funsearch Overview

Key features of our problems:

Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs

Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement

Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code

Funsearch Overview

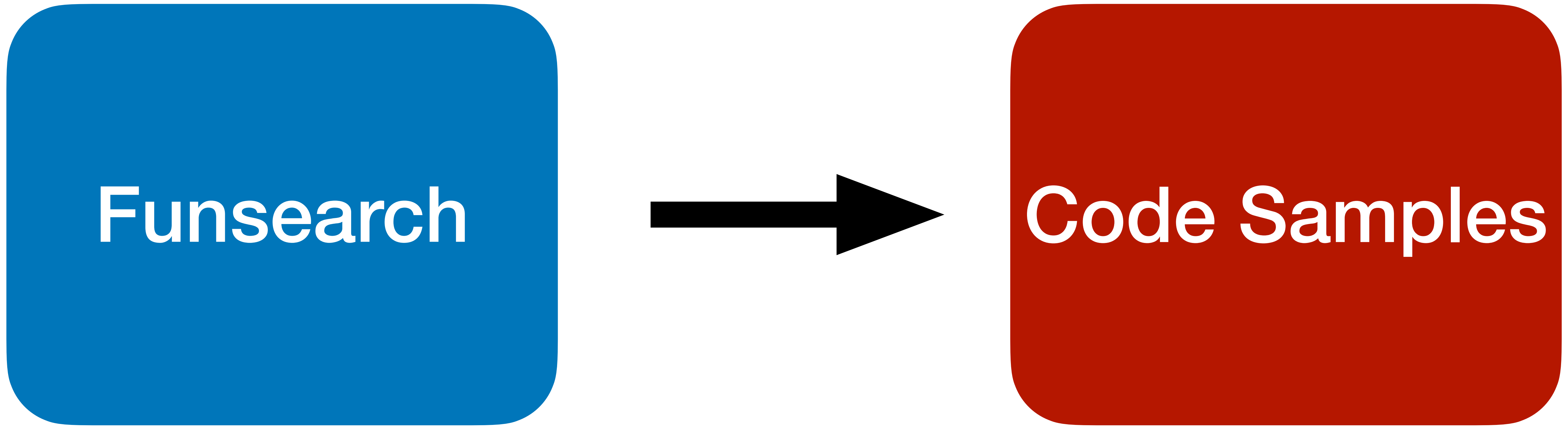
Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code

Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code



Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code

Funsearch Overview

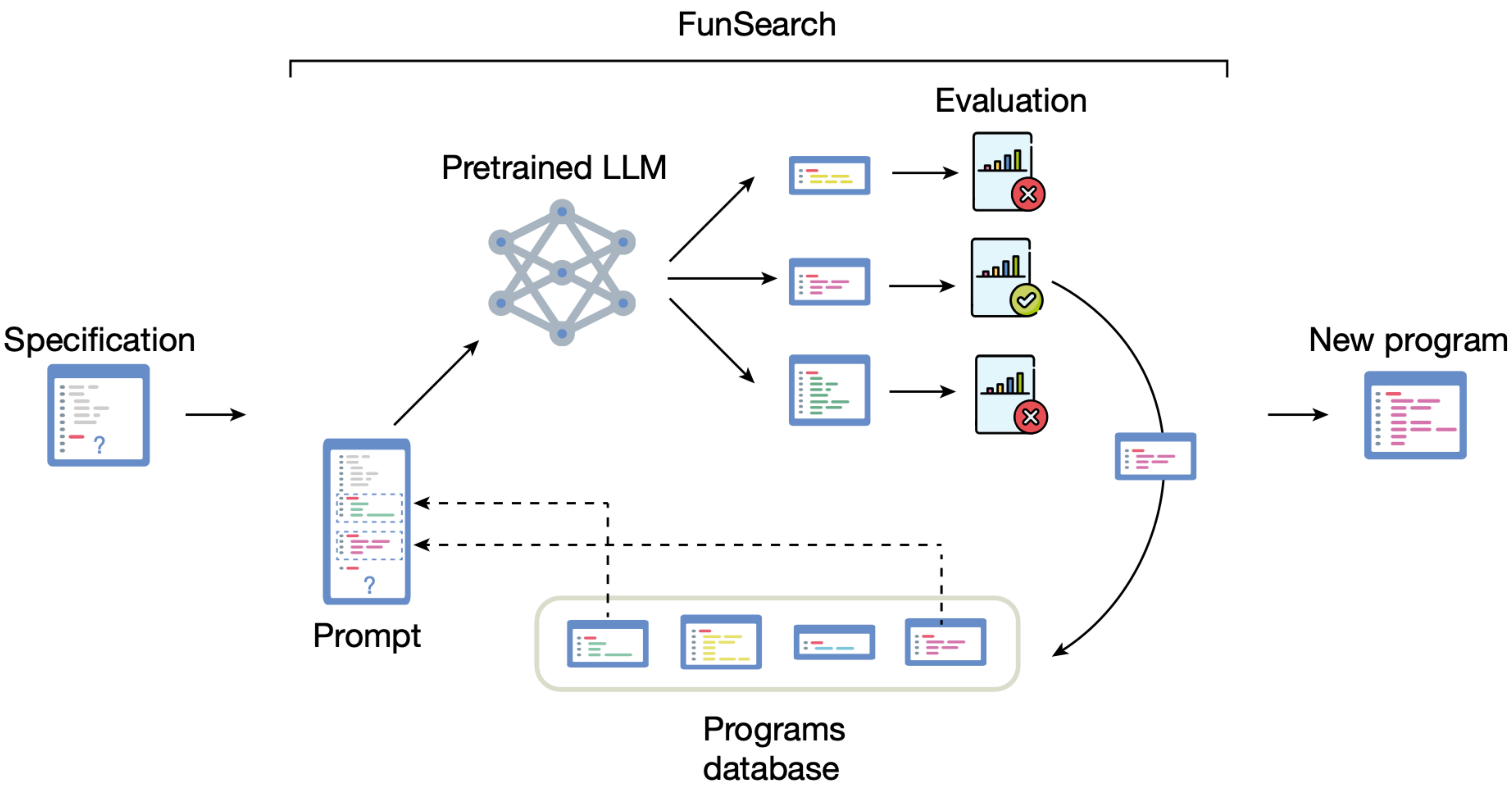
Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code

Funsearch Overview

Key features of our problems:

- Easy to verify - We can build a filter for incorrect / bad programs
- No data available - We will rely on some self improvement
- Smooth Scoring - We can make gradual improvements to the code



Funsearch Code Improvement

The role of the Large Language Model

Take a code that works, make a small improvement to it.

Funsearch Code Improvement

We'll start with a code that builds capsets (likely sub-optimally)

```
def evaluator(dim):
    subset = build_capset(dim)
    return len(subset) if is_capset(subset) else 0

def build_capset(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset_v0(elem):
    return (elem[0]==0 and elem[1]<=1)
```

The role of the Large Language Model

Take a code that works, make a small improvement to it.

Example Slide from Matej Balog CMSA talk[9]

Funsearch Code Improvement

We'll prompt the LLM to make an improvement in the code.

```
def evaluator(dim):
    subset = build_capset(dim)
    return len(subset) if is_capset(subset) else 0

def build_capset(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset_v0(elem):
    return (elem[0]==0 and elem[1]<=1)

def is_in_capset_v1(elem):
    """Improved version of is_in_capset_v0"""
    return (elem[0] <=1 ) and elem( [0] <= 1)
```

Generated by LLM

The role of the
Large Language Model

Take a code that works, make a small improvement to it.

Example Slide from Matej Balog CMSA talk[9]

Funsearch Code Improvement

We'll prompt the LLM to make an improvement in the code.

The role of the
Large Language Model

Take a code that works, make a small improvement to it.

```
def evaluator(dim):
    subset = build_capset(dim)
    return len(subset) if is_capset(subset) else 0

def build_capset(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset_v0(elem):
    return (elem[0]==0 and elem[1]<=1)

def is_in_capset_v1(elem):
    """Improved version of is_in_capset_v0"""
    return (elem[0] <=1 ) and elem( [0] <= 1)
```

Gives size 2 capset in dim 2

Gives size 4 capset in dim 2

Example Slide from Matej Balog CMSA talk[9]

Funsearch Code Improvement

We'll substitute the new code in and repeat!

```python
def evaluator(dim):
    subset = build_capset(dim)
    return len(subset) if is_capset(subset) else 0

def build_capset(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset_v0(elem):
    return (elem[0] <=1 ) and elem( [0] <= 1)

def is_in_capset_v1(elem):
    """Improved version of is_in_capset_v0"""
```

The role of the
Large Language Model

Take a code that works, make a small improvement to it.

Example Slide from Matej Balog CMSA talk[9]

Funsearch Code Improvement

The role of the
Large Language Model

Take a code that works, make a small improvement to it.

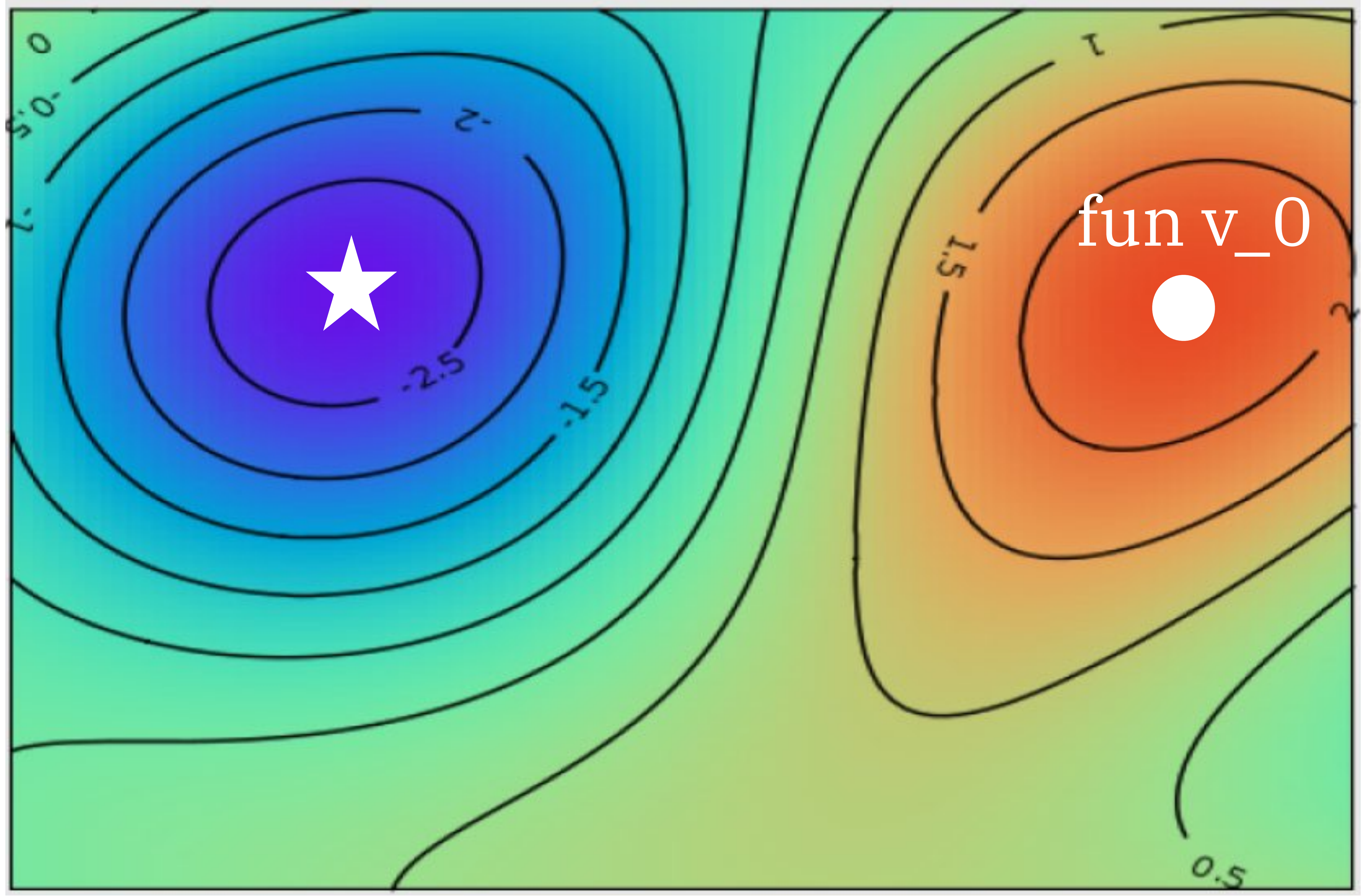
We'll substitute the new code in and repeat!

```
def evaluator(dim):
    subset = build_capset(dim)
    return len(subset) if is_capset(subset) else 0

def build_capset(dim):
    return [elem for elem in grid(dim) if is_in_capset(elem)]

def is_in_capset_v0(elem):
    return (elem[0] <=1 ) and elem( [0] <= 1)

def is_in_capset_v1(elem):
    """Improved version of is_in_capset_v0"""
```

But we need to be careful: Iteratively improving the same candidate can lead to local minima.

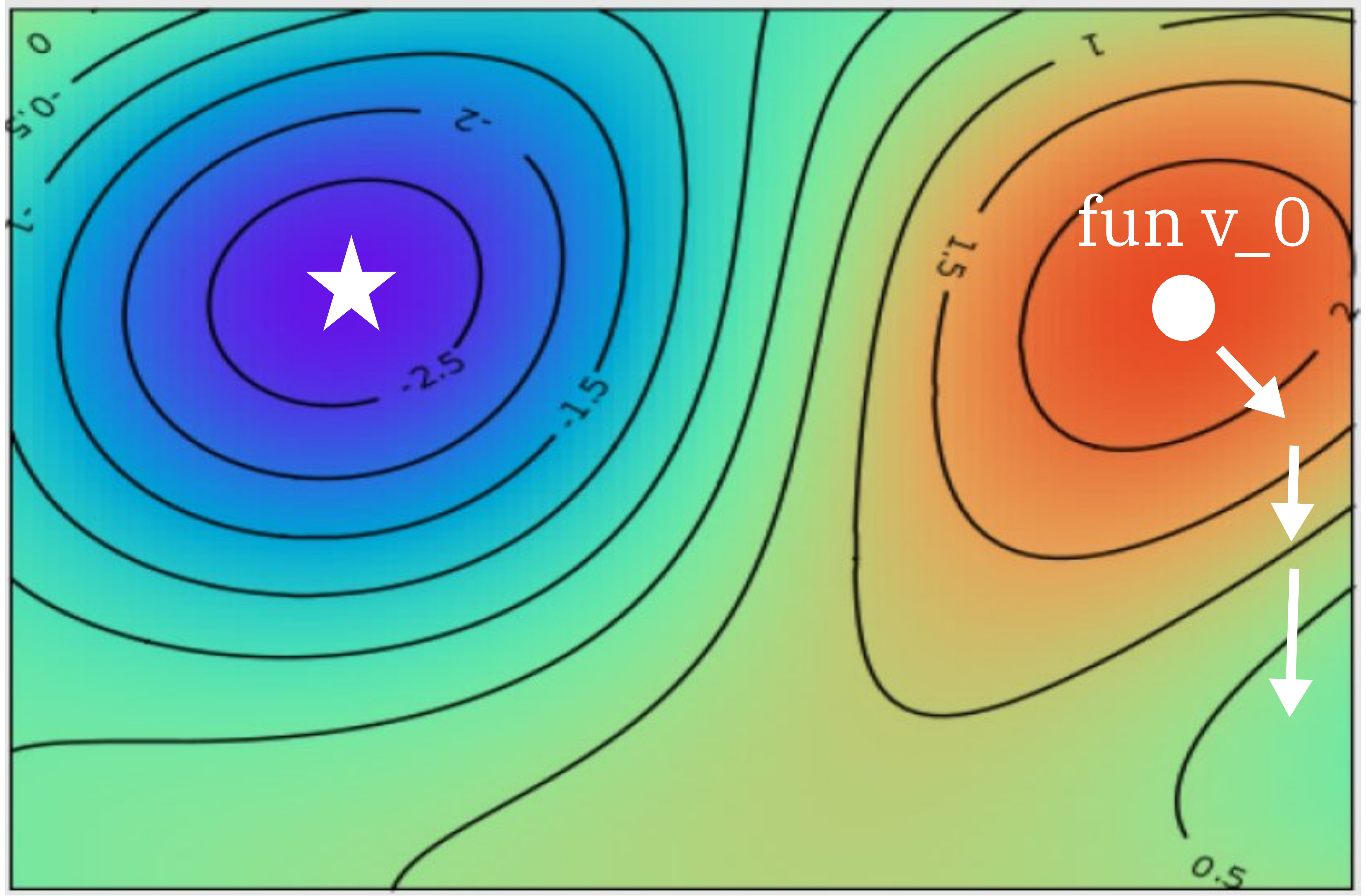
Example Slide from Matej Balog CMSA talk[9]

Funsearch Code Improvement



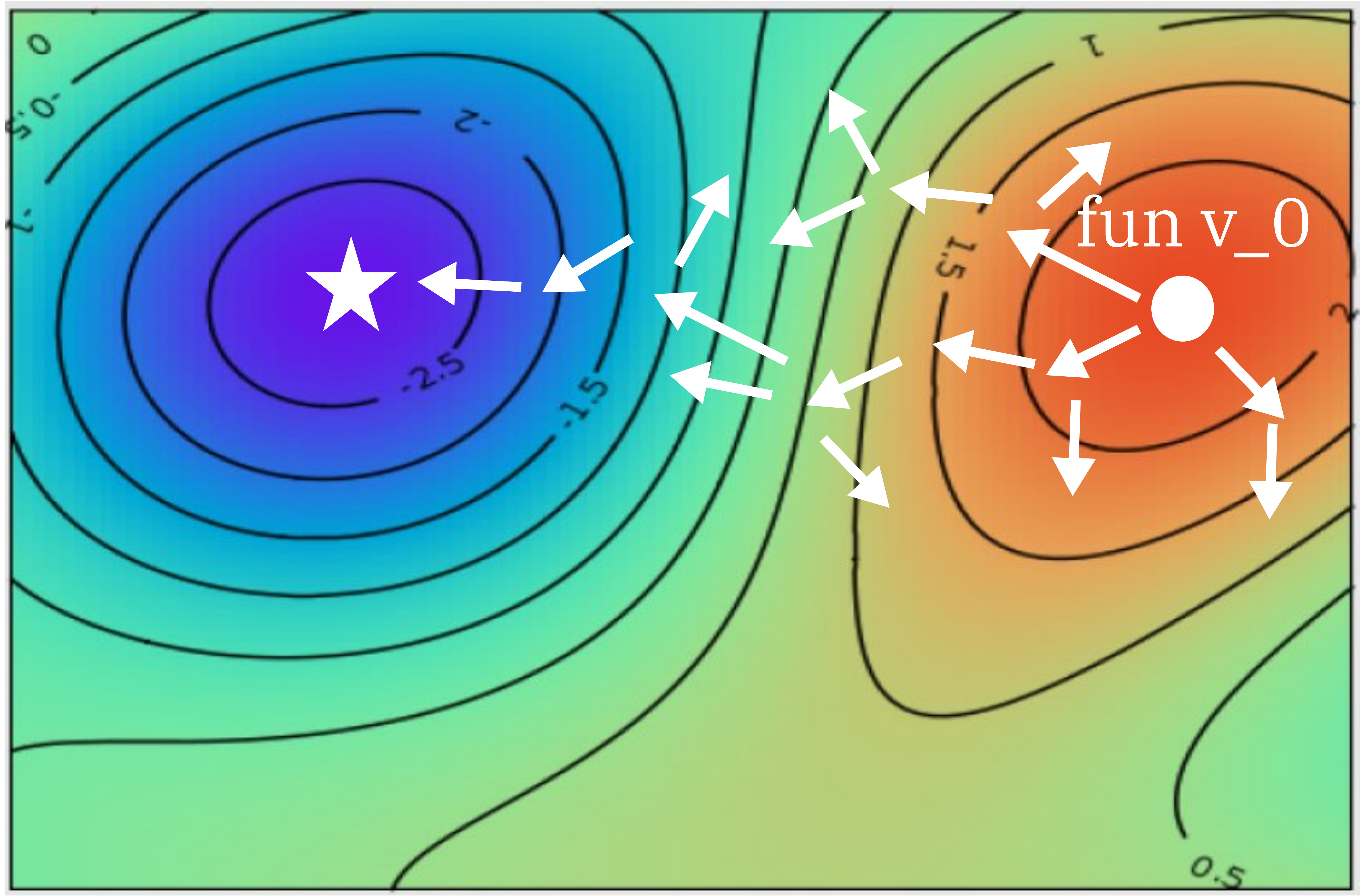
But we need to be careful: Iteratively improving the same candidate can lead to local minima.

Funsearch Code Improvement



But we need to be careful: Iteratively improving the same candidate can lead to local minima.

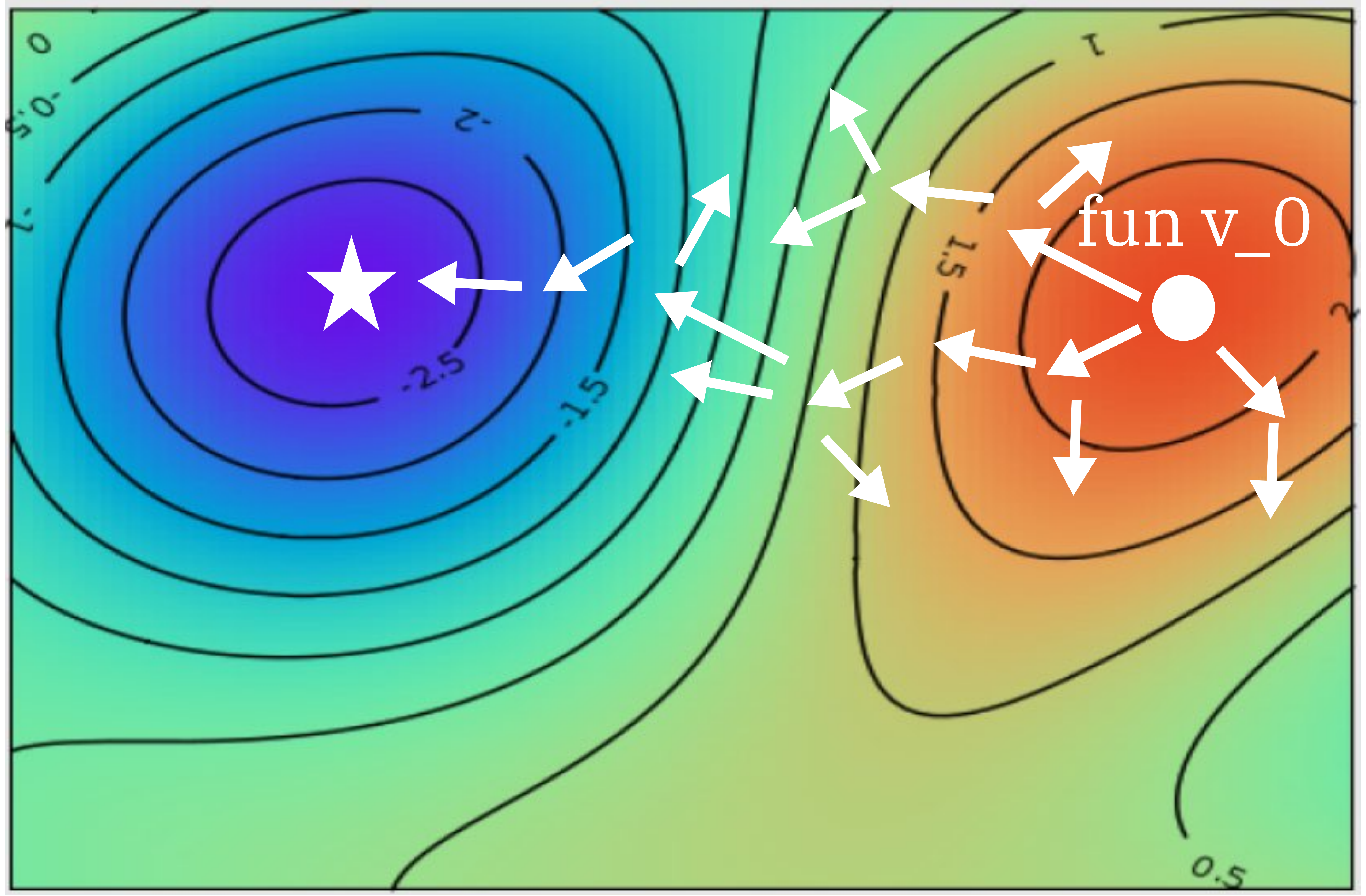
Funsearch Code Improvement



But we need to be careful: Iteratively improving the same candidate can lead to local minima.

We want to have multiple different perturbations of the code that we can study over time.

Funsearch Code Improvement



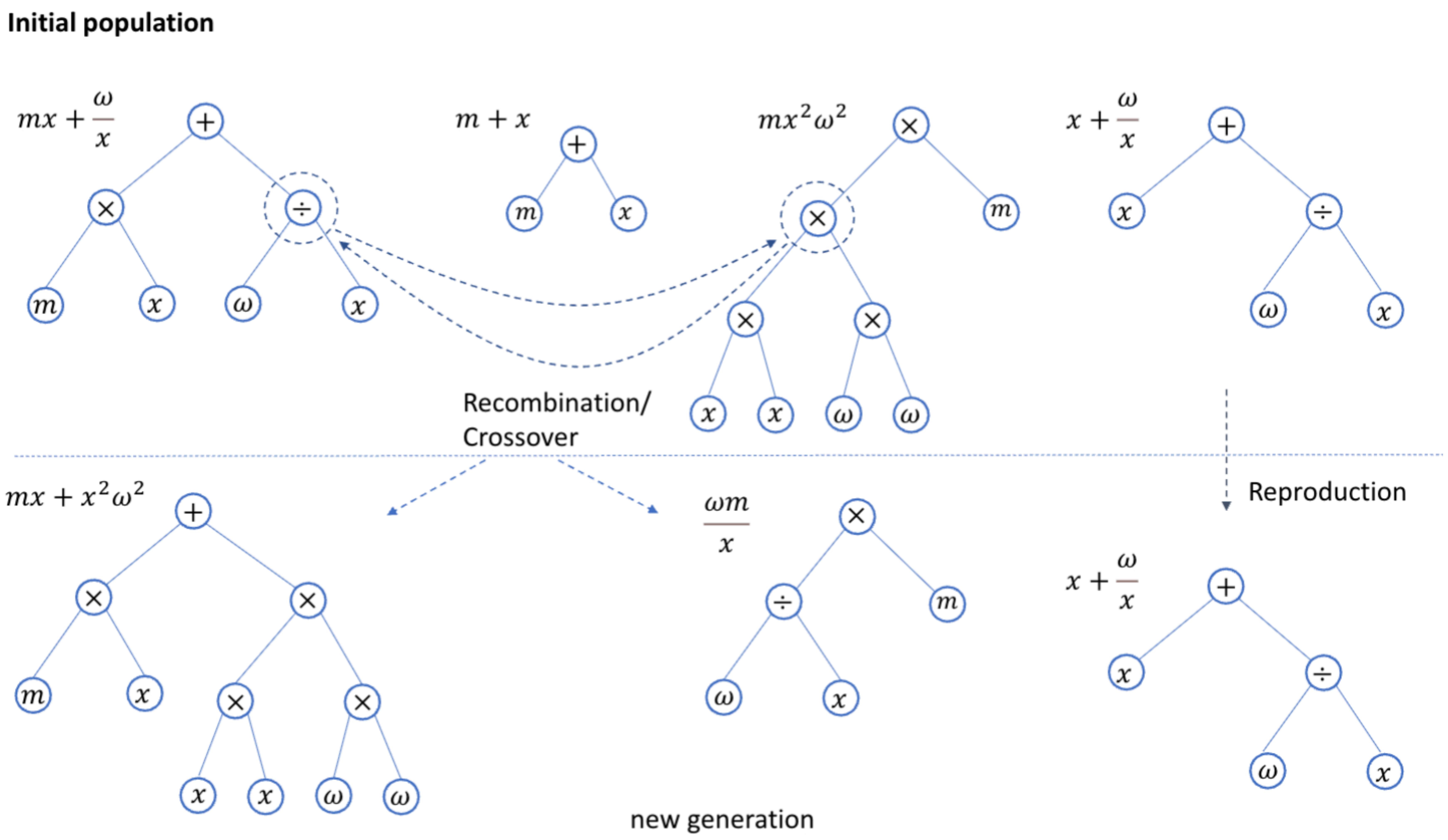
<u>Solution</u>
Genetic
Evolution

But we need to be careful: Iteratively improving the same candidate can lead to local minima.

We want to have multiple different perturbations of the code that we can study over time.

Funsearch Genetic Algorithm

Earlier in the summer school, we saw an application of a genetic algorithm:



Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:

1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

Funsearch Genetic Algorithm

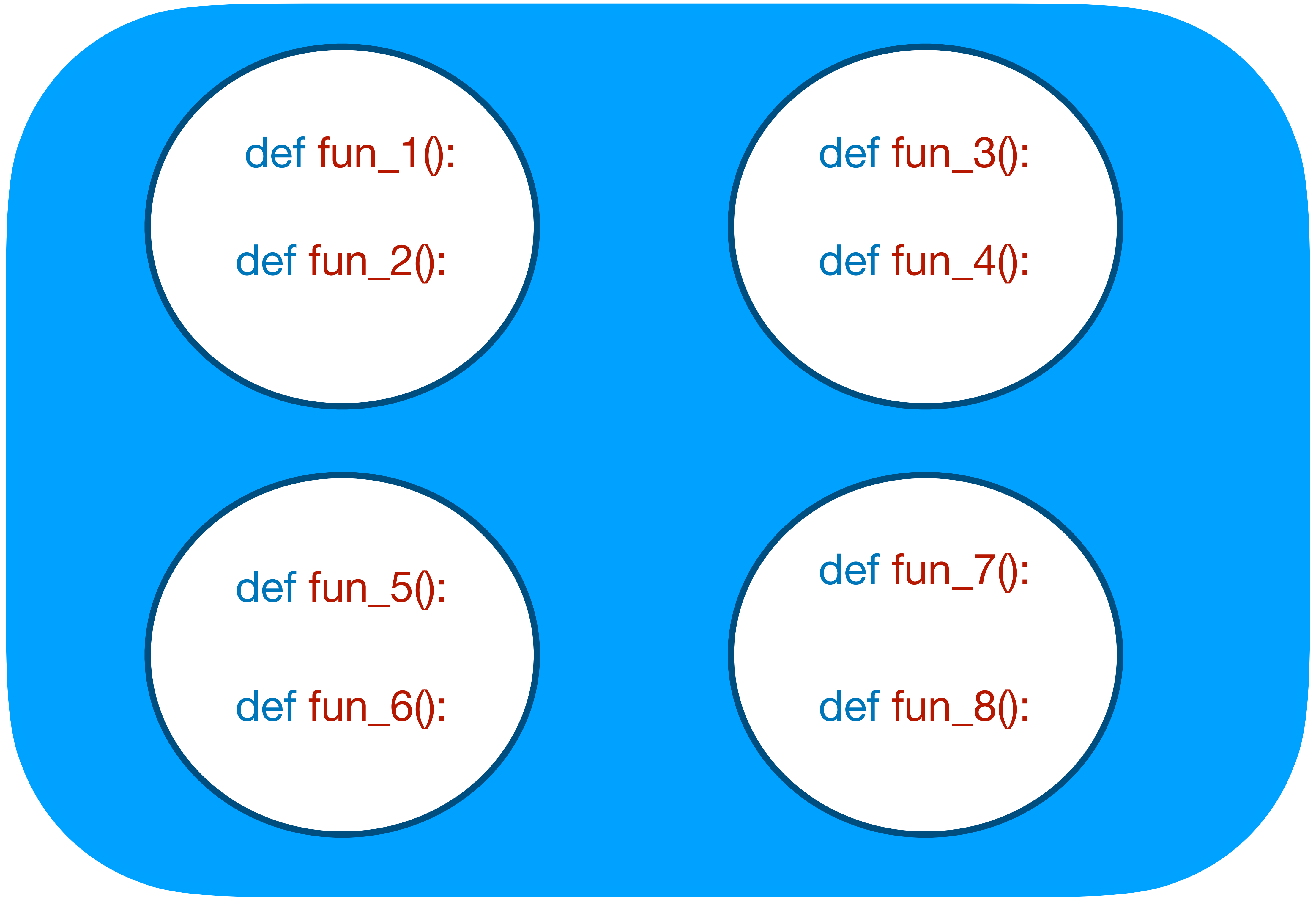
In order to implement a genetic evolution algorithm, we need:

1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

Initial Program

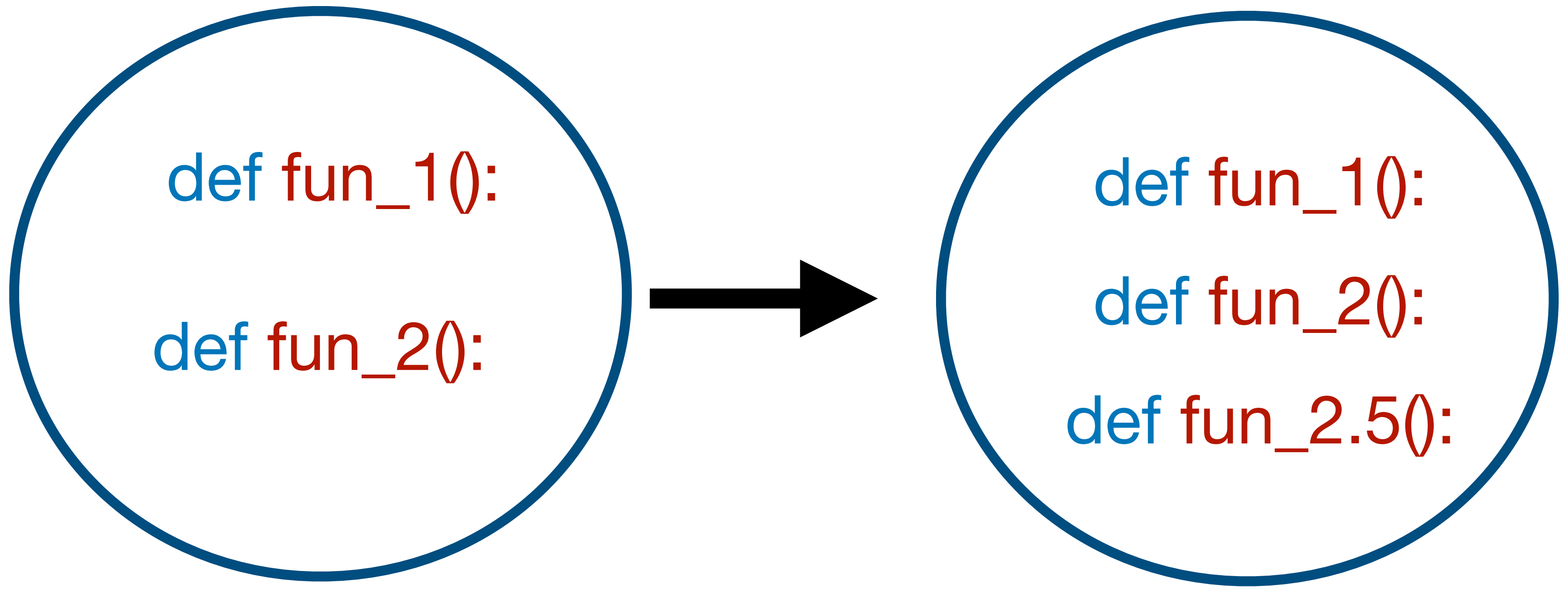
```
def function_to_evolve(inputs):
    return math.random()
```

Database of programs partitioned into islands

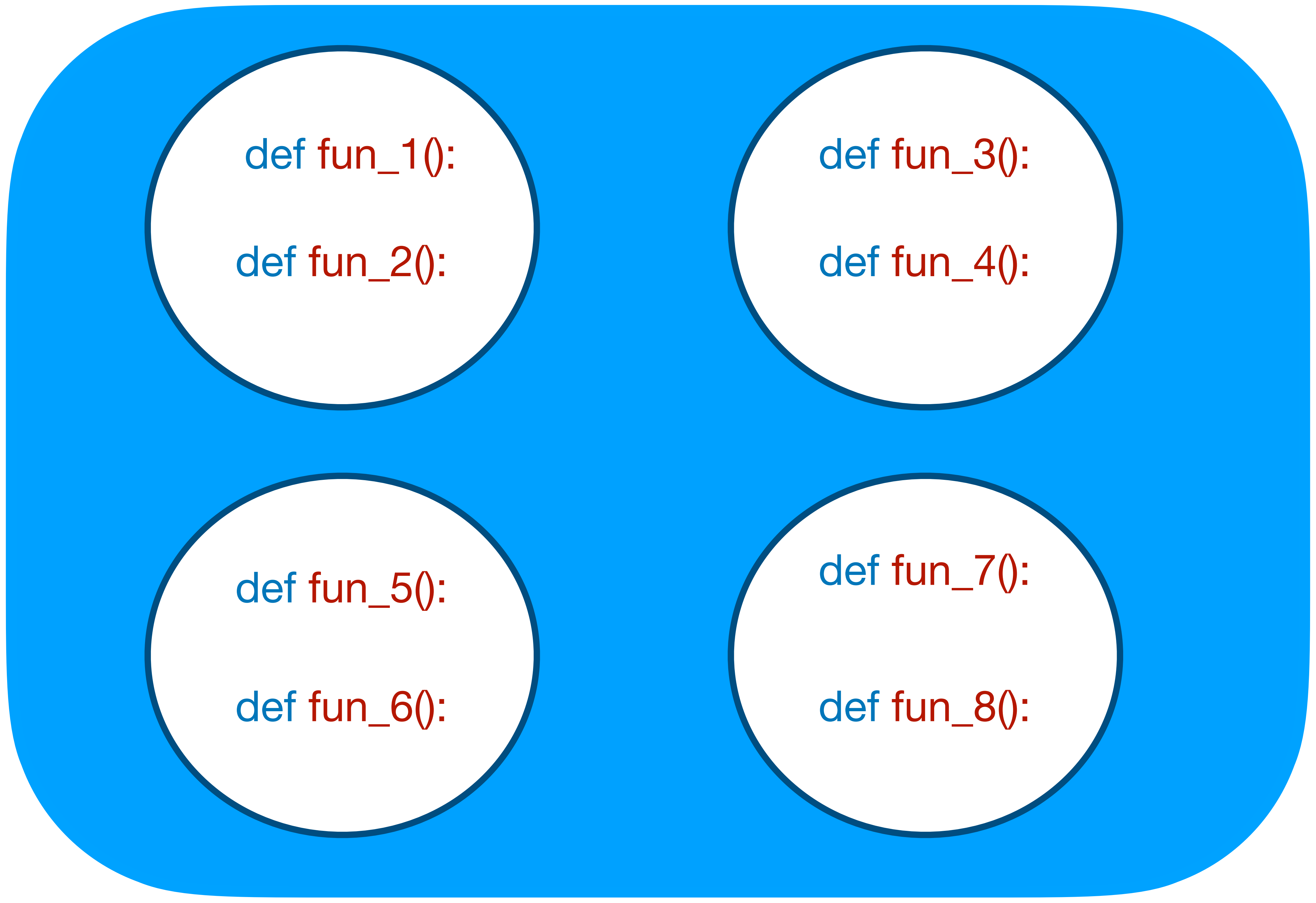


Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:
1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive



Database of programs partitioned into islands

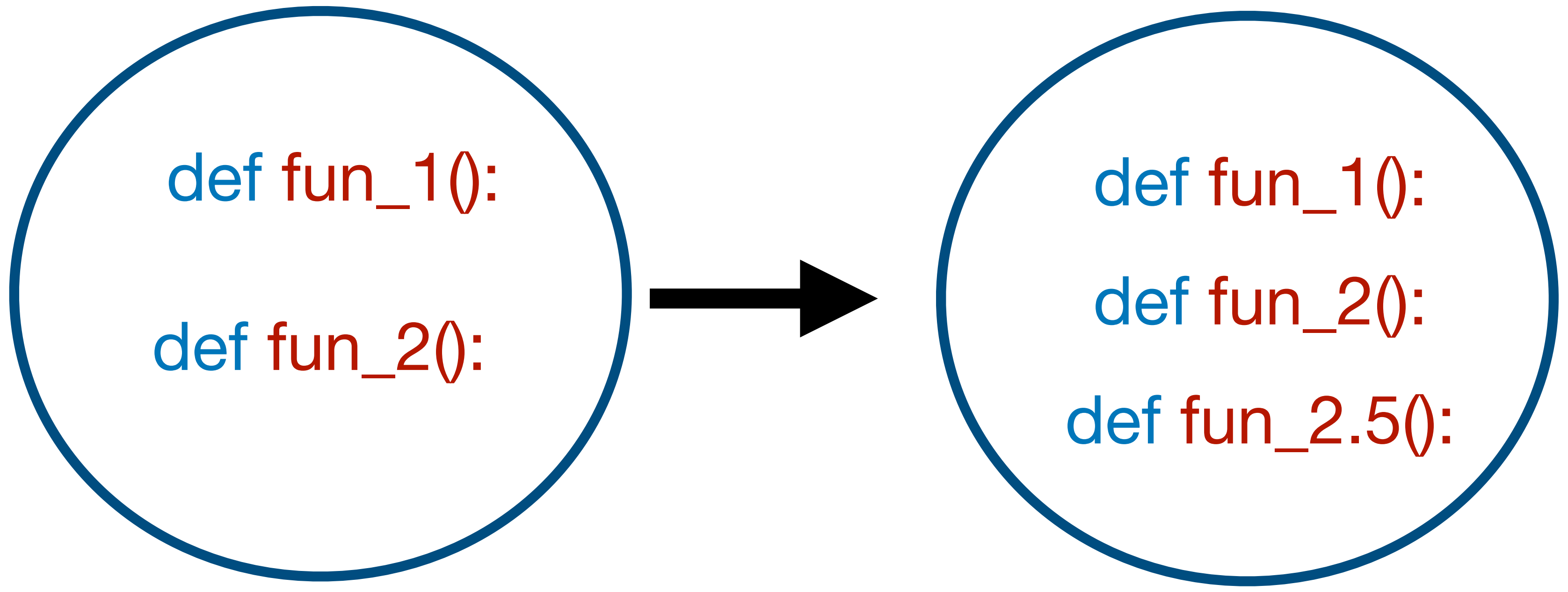



```python
def is_in_capset_v0(elem):
    return (elem[0]==0 and elem[1]<=1)

def is_in_capset_v1(elem):
    """Improved version of is_in_capset_v0"""
    return (elem[0] <=1 ) and elem([0] <= 1)
```

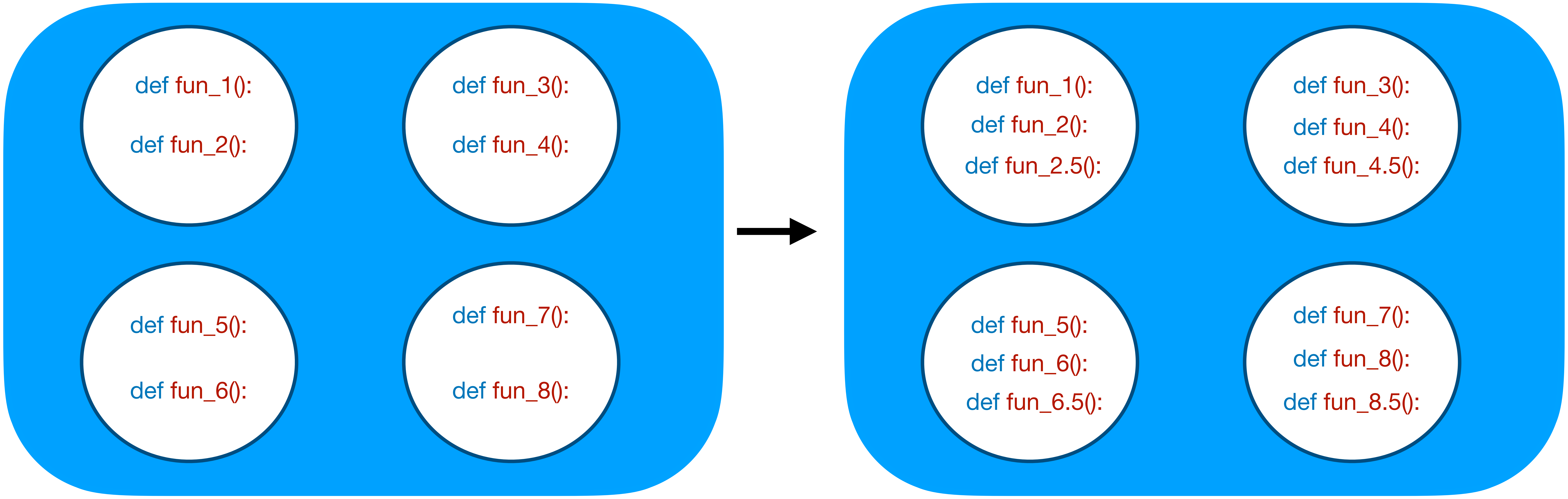
Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:

1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

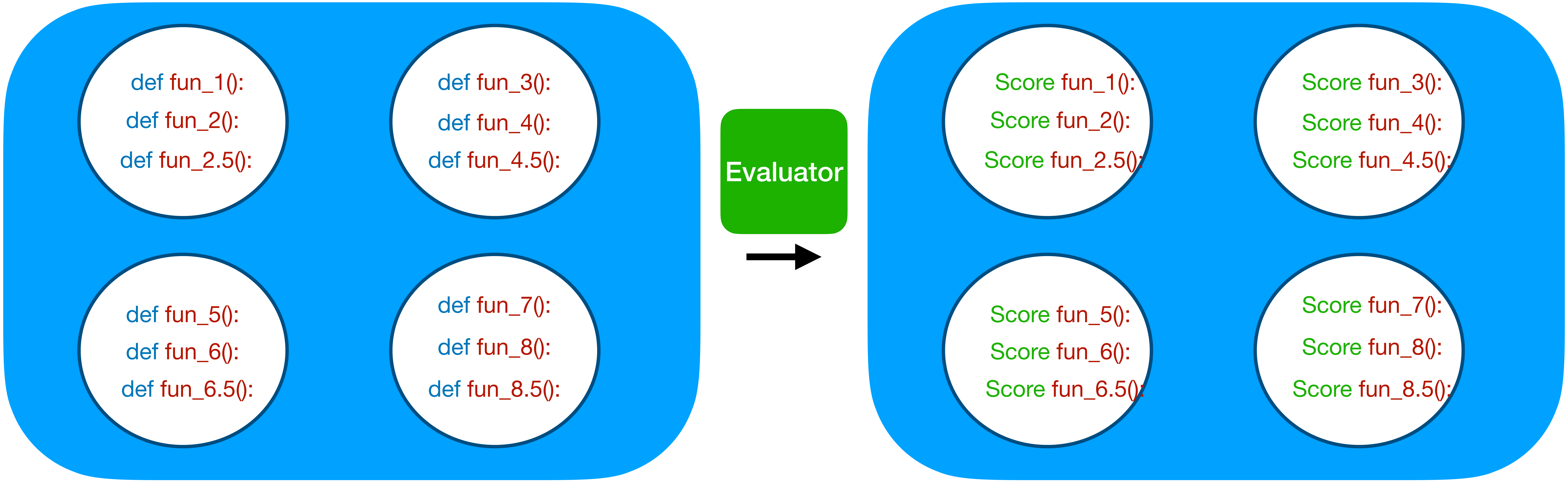


Database of programs partitioned into islands



Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:
1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

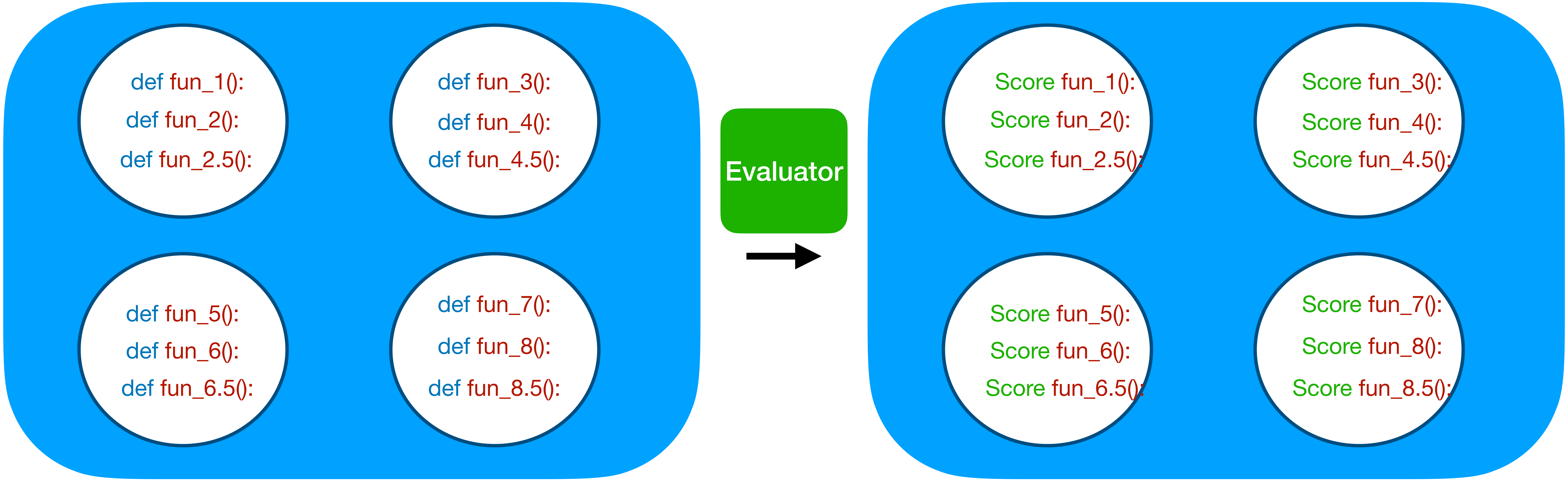


Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:

1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

Repeat until satisfied!

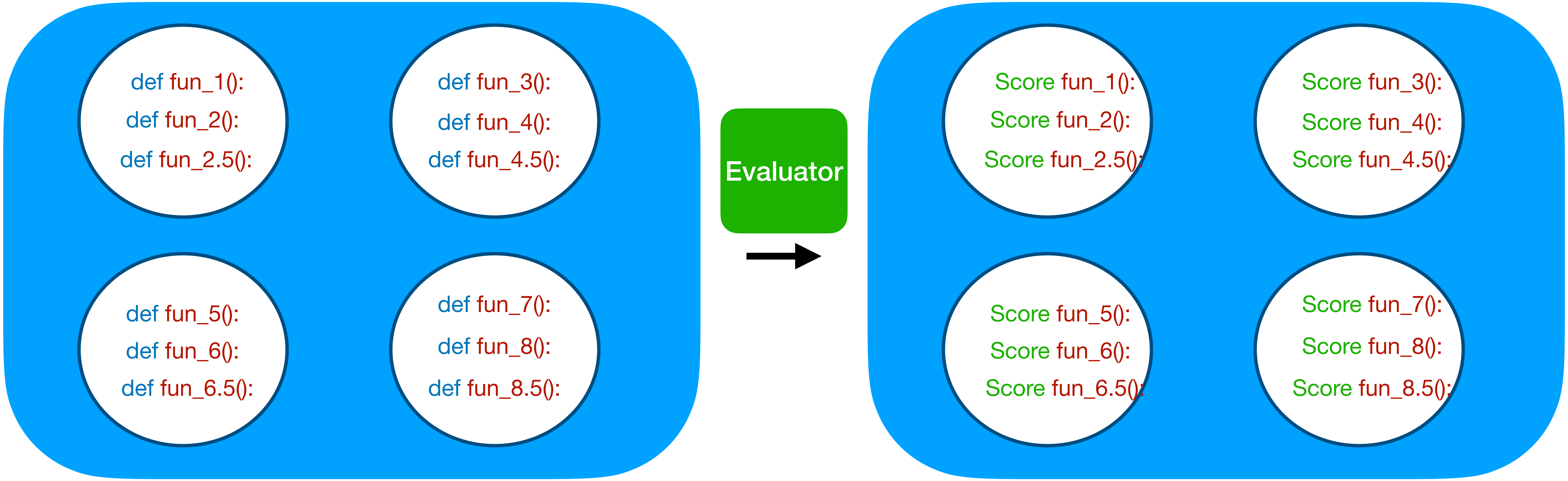


Funsearch Genetic Algorithm

In order to implement a genetic evolution algorithm, we need:

1. An initialization of the population
2. A mutation mechanism for each population
3. A score to test for which species survive

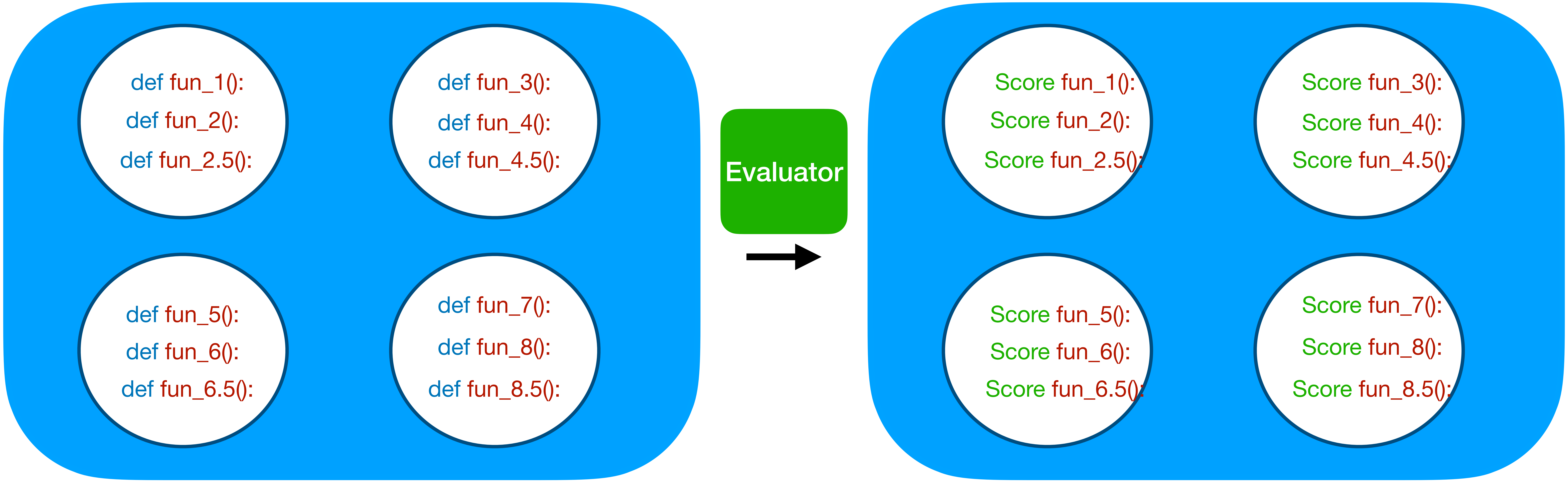
Repeat until satisfied!



Funsearch Genetic Algorithm

Leveraging three things here:

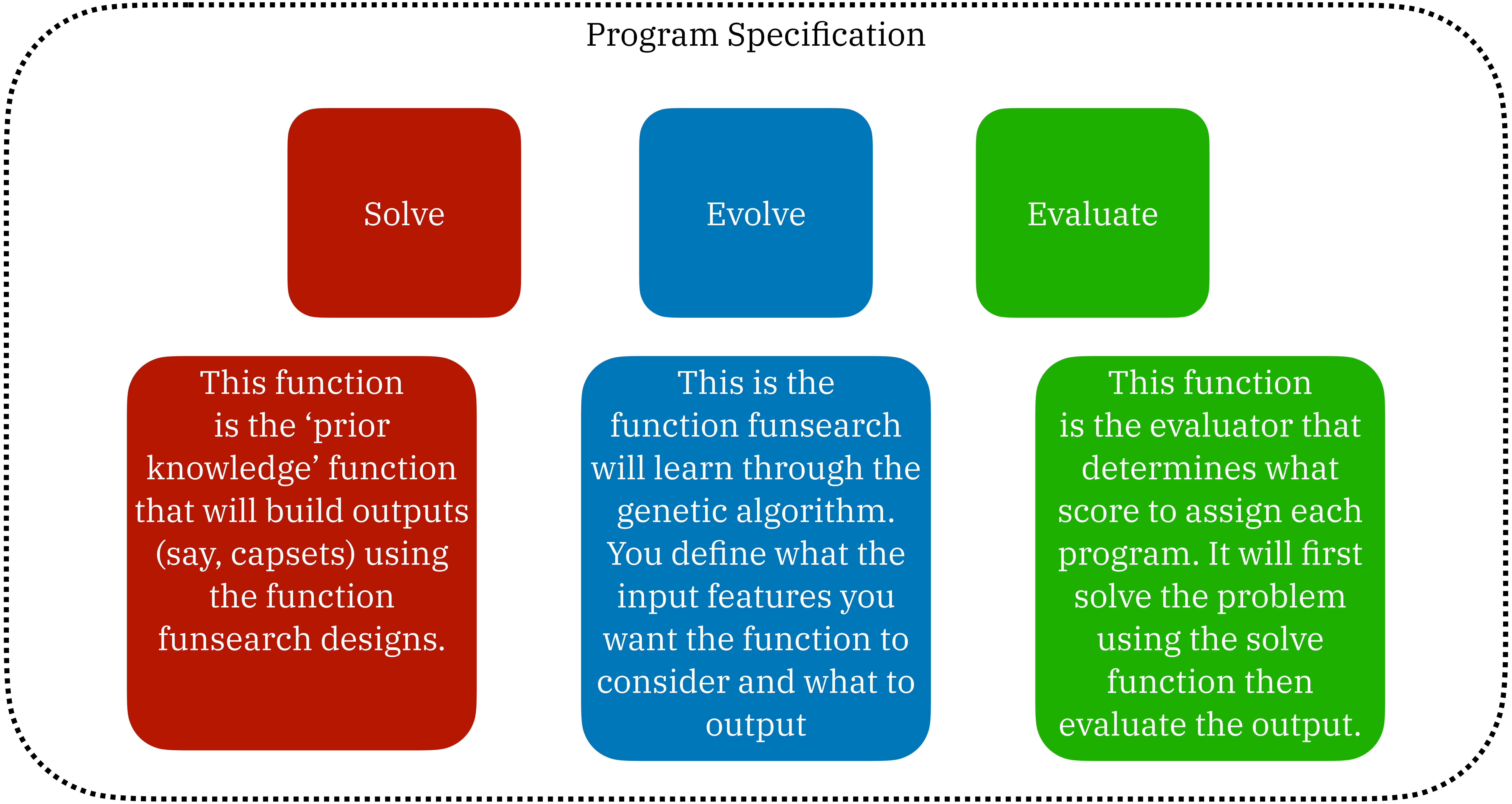
1. Our ability to run inference in parallel
2. Our ability to evaluate fast in parallel
3. LLM ‘creativity’.



Let’s see how this works from input to output!

Funsearch Input

We input three functions.



Funsearch Input

Solve

This function is the ‘prior knowledge’ function that will build outputs (say, capsets) using the function funsearch designs.

```
def solve(n: int):
    "Returns a large isosceles-free subset of the integer lattice"
    #Generate list of all lattice coordinates
    all_points = list(itertools.product({0,1,2}, repeat=n))
    #Assign a weight to each point
    priorities = [priority(point) for point in all_points]
    #Initialize capset
    capset = []
    while np.any(priorities != -np.inf):
        #Find the highest priority point
        max_index = argmax(priorities)
        max_point = all_points[max_index]
        #Add it to the set if we don't form a capset
        if is_capset(capset.copy()append(max_point)):
            capset.append(max_point)
        #Remove the point from future consideration
        priorities[max_index] = -np.inf
    return capset
```

Funsearch Input

Evolve

This is the function funsearch will learn through the genetic algorithm. You define what the input features you want the function to consider and what to output

```
def solve(n: int):
    "Returns a large isosceles-free subset of the integer lattice"
    #Generate list of all lattice coordinates
    all_points = list(itertools.product({0,1,2}, repeat=n))
    #Assign a weight to each point
    priorities = [priority(point) for point in all_points]
    #Initialize capset
    capset = []
```

```
@funsearch.evolve
def priority(point):
    "Returns the weight to assign the point"
    "Higher weight -> higher priority to select point first"
    return 0.0
```

Funsearch Input

Evaluate

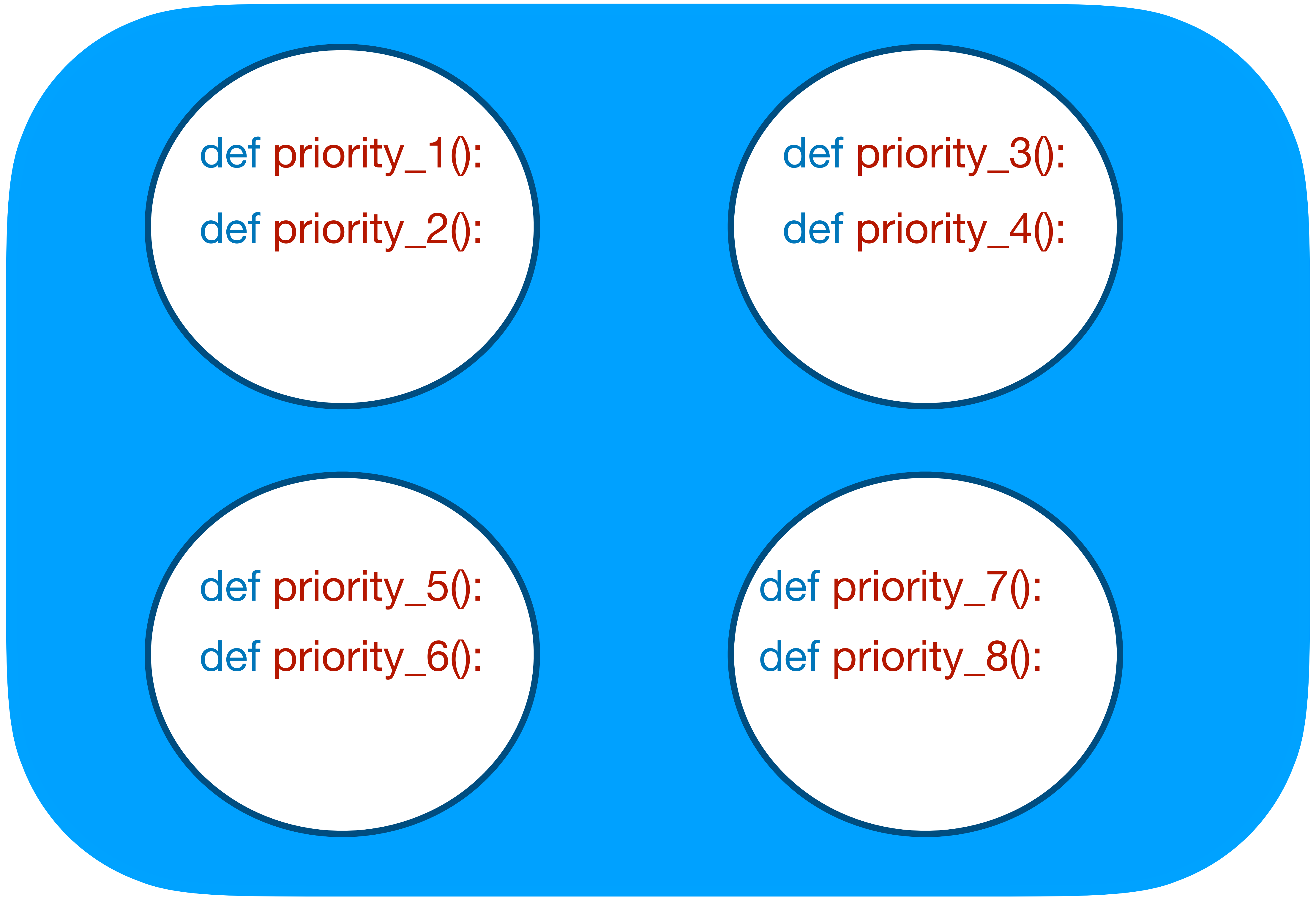
This function is the evaluator that determines what score to assign each program. It will first solve the problem using the solve function then evaluate the output.

```
def solve(n: int):
    "Returns a large isosceles-free subset of the integer lattice"
    #Generate list of all lattice coordinates
    all_points = list(itertools.product({0,1,2}, repeat=n))
    #Assign a weight to each point
    priorities = [priority(point) for point in all_points]
    #Initialize capset
    capset = []
```

```
@funsearch.run
def evaluator(n: int):
    "Returns the size of an n-dimensional capset"
    capset = solve(n)
    return len(capset)
```

Funsearch Evolution Step

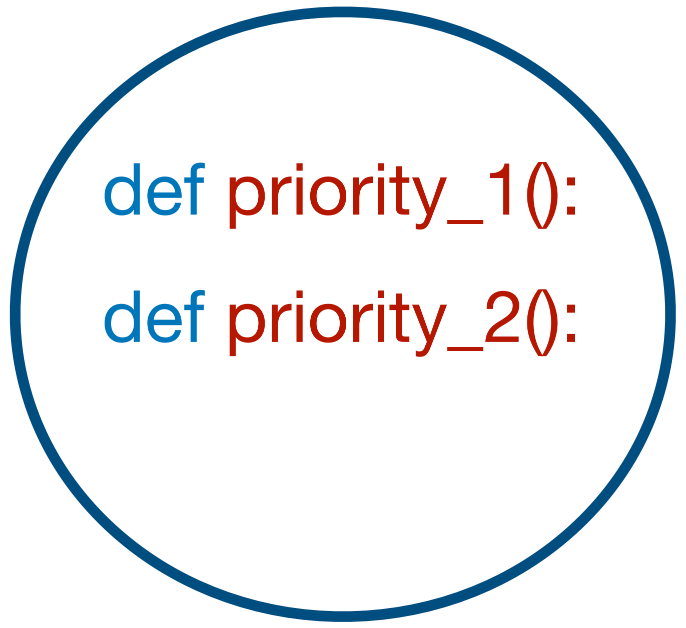
- We will initialize some number of islands/independent databases with copies of our initial function to evolve. For each island, we will develop a database of program.

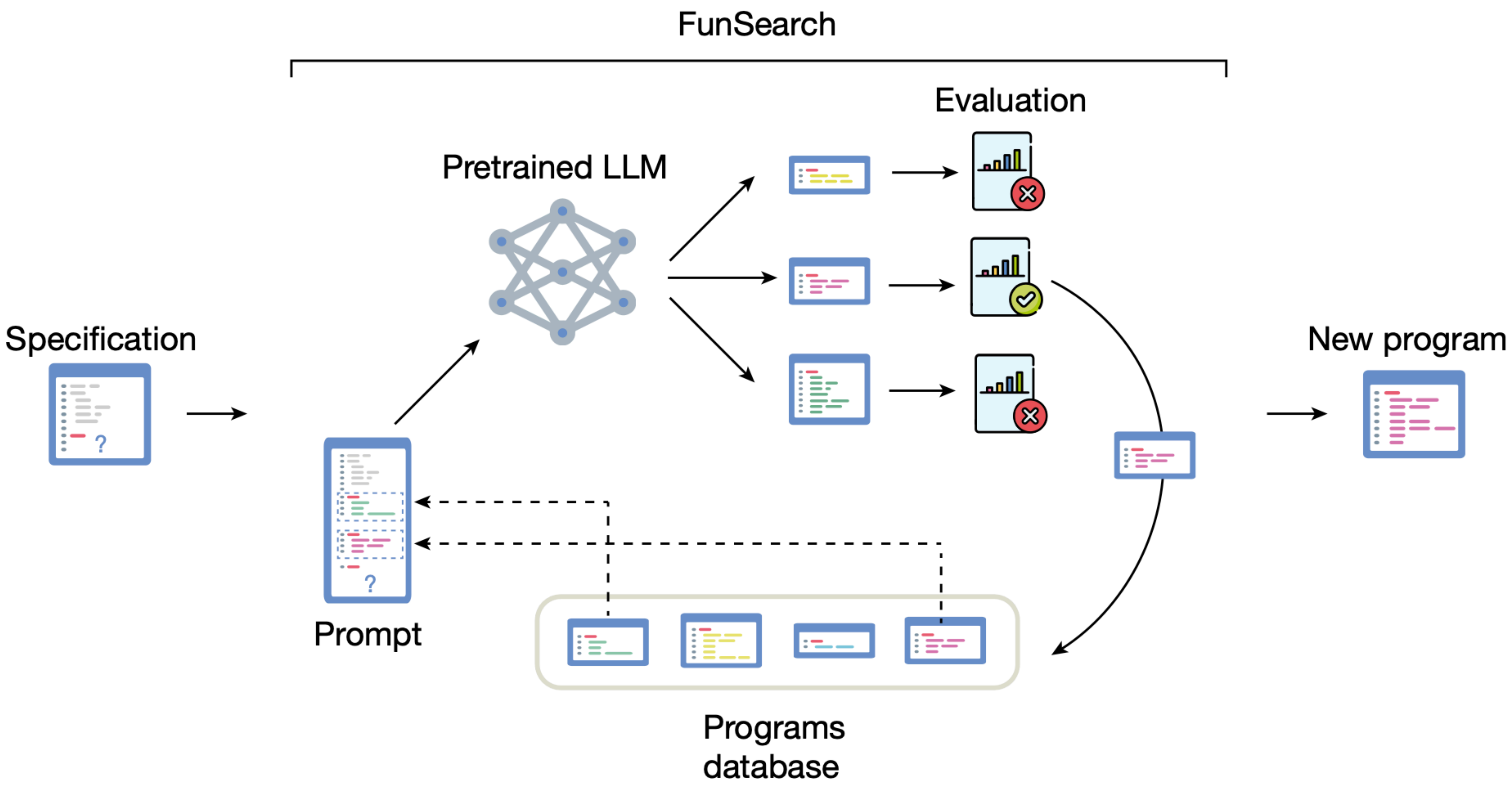


Funsearch Evolution Step

- We will initialize some number of islands/independent databases with copies of our initial function to evolve. For each island, we will develop a database of program.

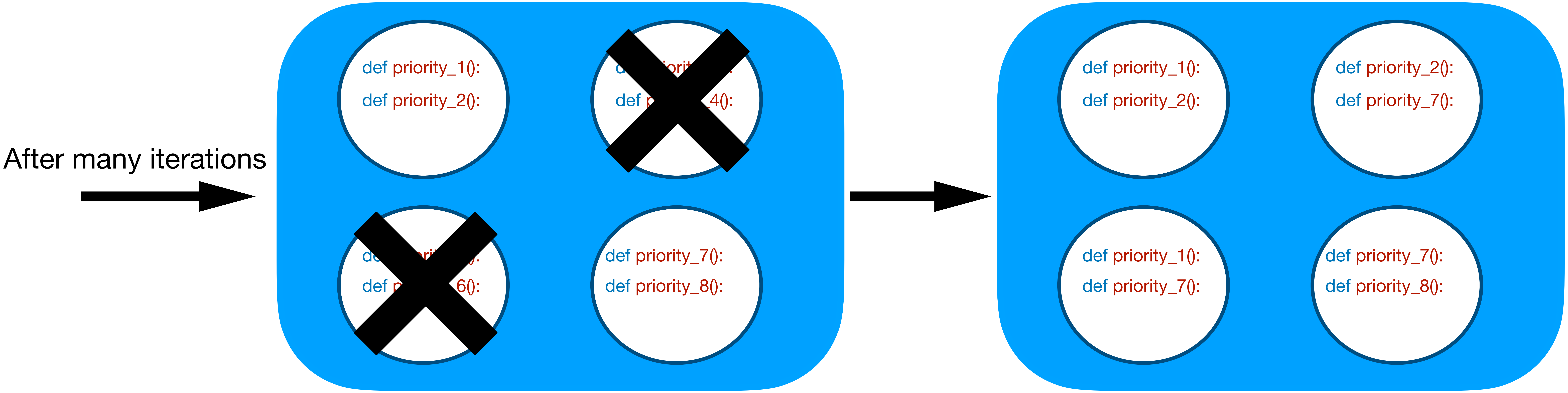
- We repeat the evolutionary algorithm for each island for as many iterations as desired.





Funsearch Evolution Step

- We will initialize some number of islands/independent databases with copies of our initial function to evolve. For each island, we will develop a database of program.
- We repeat the evolutionary algorithm for each island for as many iterations as desired.
- Some islands might converge to suboptimal solutions. We therefore periodically reset the islands and seed them with good performing programs from the surviving islands.



Funsearch Output and Results

- This method was used to discover the largest known Capset at size n=8.
- This was used to extend the best known lower bound at the time from 2.2173^n to the now best 2.2202^n.

Dimension *n*	2	3	4	5	6	7	8
Previous best construction	4	9	20	45	112	236	496
***FunSearch* construction**	**4**	**9**	**20**	**45**	**112**	**236**	**512**

Funsearch Output and Results

Priority function that is used to generate the largest known capset for n=8.

```
def priority(el: tuple[int,...],
↪   n: int) -> float:
  score = n
  in_el = 0
  el_count = el.count(0)

  if el_count == 0:
    score += n**2
    if el[1] == el[-1]:
      score *= 1.5
    if el[2] == el[-2]:
      score *= 1.5
    if el[3] == el[-3]:
      score *= 1.5
  else:
    if el[1] == el[-1]:
      score *= 0.5
    if el[2] == el[-2]:
      score *= 0.5

  for e in el:
    if e == 0:
      if in_el == 0:
        score *= n * 0.5
      elif in_el == el_count - 1:
        score *= 0.5
      else:
        score *= n * 0.5 ** in_el
      in_el += 1
    else:
      score += 1

  if el[1] == el[-1]:
    score *= 1.5
  if el[2] == el[-2]:
    score *= 1.5

  return score
```

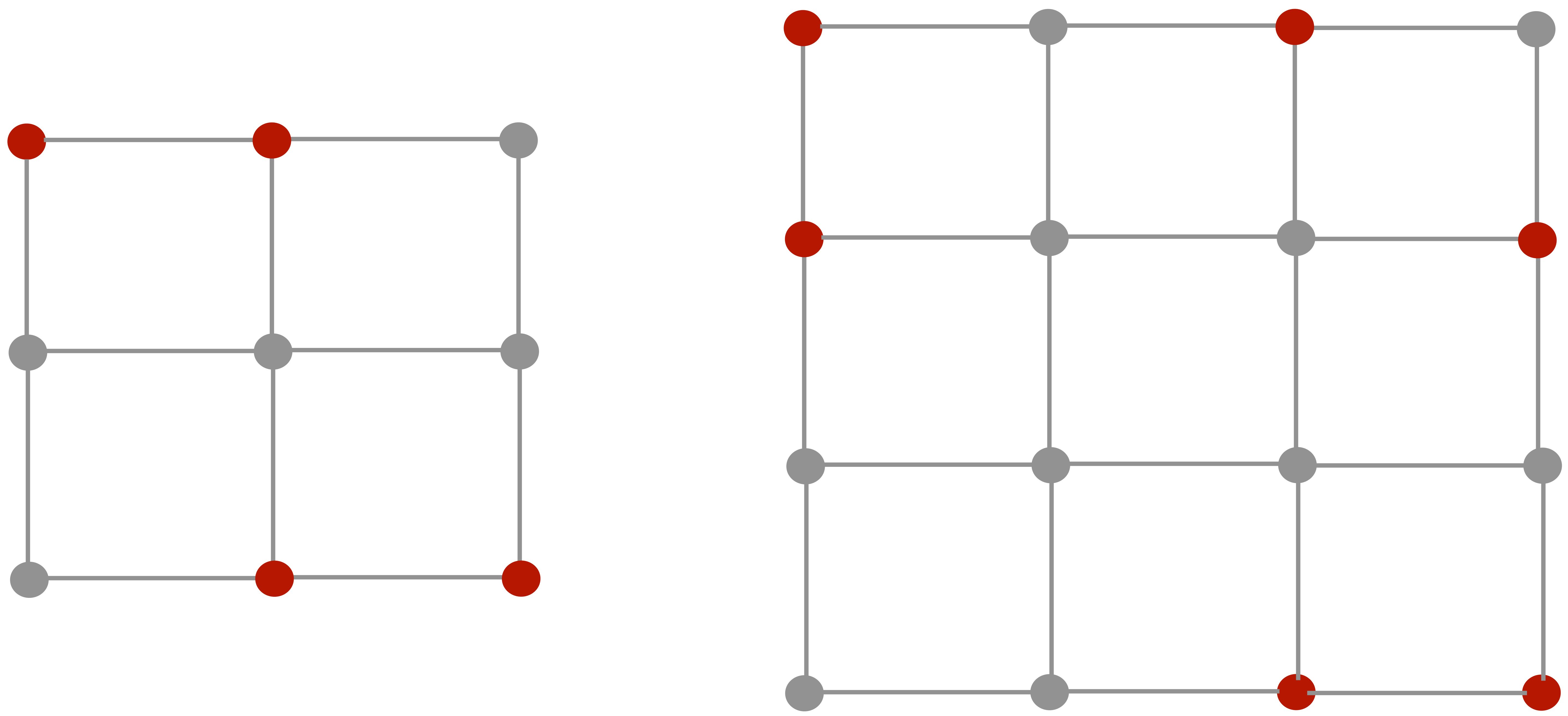
Funsearch Output and Results

Shorter code was found later using the same method

```python
def build_512_cap() -> list[tuple[int, ...]]:
  """Returns a cap set of size 512 in `n=8` dimensions."""
  n = 8
  support = lambda v: tuple(i for i in range(n) if v[i] == 0)
  stamp = lambda v: tuple(v[2 * i : 2 * i + 2].count(0) for i in range(n // 2))
  return [
      v for v in itertools.product(range(3), repeat=n) if
      (support(v) in [(1, 3, 5), (1, 2, 4), (0, 3, 4), (0, 2, 5)]) or
      (stamp(v) == (2, 0, 1, 0)) or
      (stamp(v) == (1, 2, 0, 1)) or
      (len(support(v)) == 4 and stamp(v)[:2] == (0, 1)) or
      (len(support(v)) == 1 and v[7] == 0 and v[:4].count(1) % 2 == 0) or
      (len(support(v)) == 1 and v[6] == 0 and v[:4].count(1) % 2 == 1)
  ]
```

Funsearch Output and Results

- What about the isosceles free subset problem?



Funsearch Output and Results

- Results on this will follow our recent arXiv submission[10].

ADV. THEOR. MATH. PHYS.
Volume NN, Number 1, 701–722, 2025

Generative Modeling for Mathematical Discovery

Jordan S. Ellenberg[a], Cristofero S. Fraser-Taliente[b], Thomas R. Harvey[c,d], Karan Srivastava[a], and Andrew V. Sutherland[c]

[a]University of Wisconsin-Madison
[b]University of Oxford
[c]Massachusetts Institute of Technology
[d]The NSF Institute for Artificial Intelligence and Fundamental Interactions

We present a new implementation of the LLM-driven genetic algorithm *funsearch*, whose aim is to generate examples of interest to mathematicians and which has already had some success in problems in extremal combinatorics. Our implementation is designed to be useful in practice for working mathematicians; it does not require expertise in machine learning or access to high-performance computing resources. Applying *funsearch* to a new problem involves modifying a small segment of Python code and selecting a large language model (LLM) from one of many third-party providers. We benchmarked our implementation on three different problems, obtaining metrics that may inform applications of *funsearch* to new problems. Our results demonstrate that *funsearch* successfully learns in a variety of combinatorial and number-theoretic settings, and in some contexts learns principles that generalize beyond the problem originally trained on.

Built a working, open source implementation of funsearch designed for working mathematicians

Tested the capabilities of funsearch on various models

Studied results for funsearch on the isosceles free problem - focussing on generalization outside the training set.

Funsearch Output and Results

We can design a very similar program specification!

The solve function is almost the exact same - a greedy algorithm moving in order of high to low priority weights assigned to each node.

The evaluation and initialization priority are the same.

```python
def solve(n: int) -> list[tuple[int, int]]:
    """Returns a large isosceles-free subset of an n by n integer lattice."""
    # Generate all possible points in the n x n lattice
    all_points = list(itertools.product(range(n), repeat=2))  # List of tuples (x, y)

    # Precompute priorities for all points
    priorities = np.array([priority(point, n) for point in all_points], dtype=float)

    # Initialize the isosceles-free subset
    subset = []

    # Add points to the subset in order of their priority
    while np.any(priorities != -np.inf):
        # Find the point with the highest priority
        max_index = np.argmax(priorities)
        point = all_points[max_index]

        # Check if adding this point creates an isosceles triangle
        if not forms_isosceles_triangle(subset, point):
            subset.append(point)

        # Mark this point as processed
        priorities[max_index] = -np.inf

    return subset
```

Funsearch Output and Results

- Basic models here refer to our funsearch runs with the specification files.

- The learned models don't outperform SOTA. However, we should note that the original funsearch paper reported to use that for reproducibility of some of their results, 15 instances of StarCoder-15B running on A100 40 GB GPU each and 5 CPU servers (each running 32 evaluators in parallel) for two days, with an estimate that when running on Google Cloud, the price of an experiment is around $800 – $1400[8]. We ran trained these models for 30 minutes with ~$2 worth of tokens on mistral-tiny with an 8-core M2 MacBook.

problem setup	n								
	12	**13**	**16**	**21**	**23**	**25**	**27**	**32**	**64**
maximum known	20	22	28	36	40	44	48	56	110
basic models	20	20	26	34	36	40	40	46	86*

So what can we say about the functions?

*We had a run improve to 96 when run with GPT4o with 10x the number of tokens.

Funsearch Output and Results

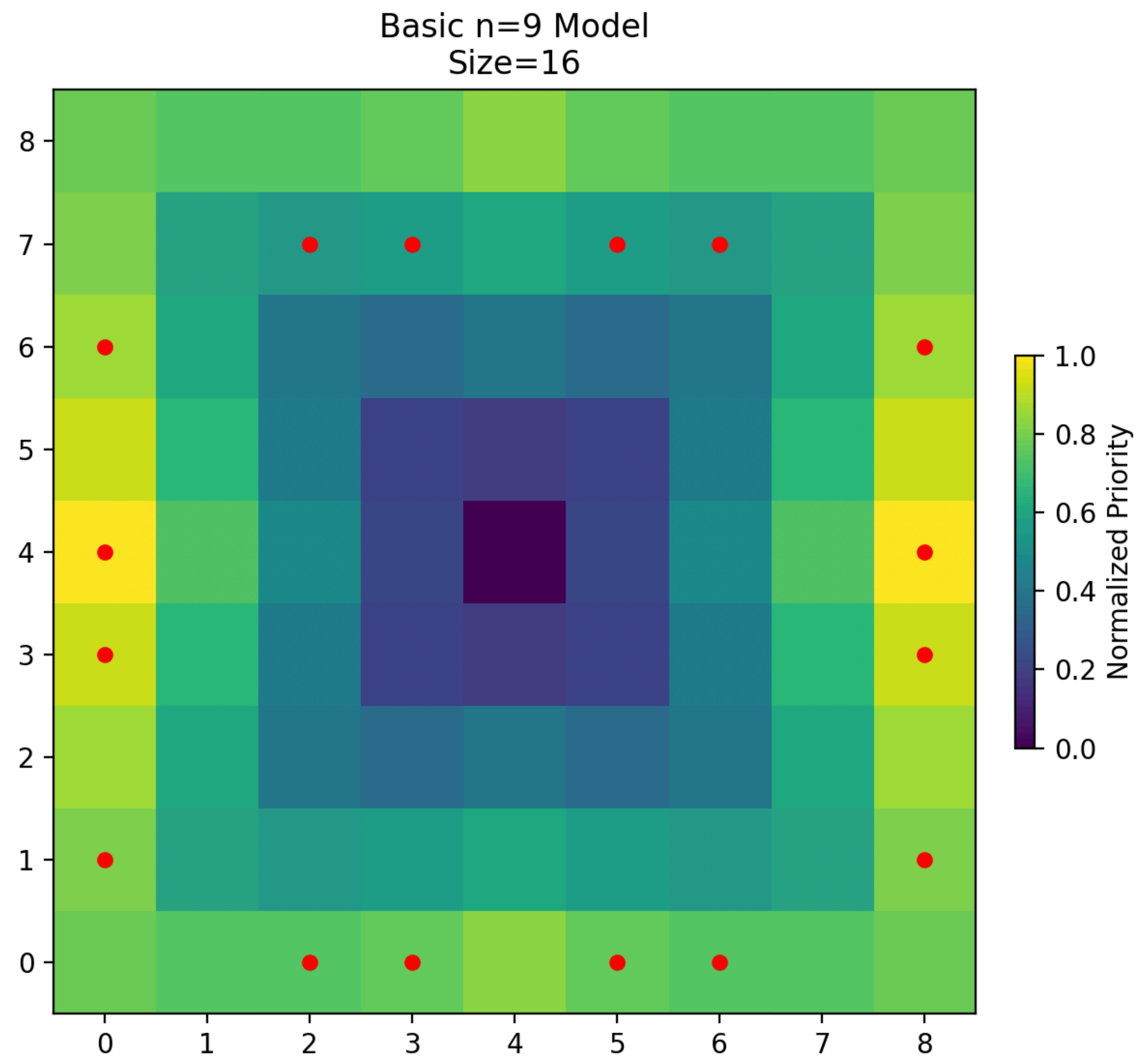
Advantage 1: We can see what the priority function looks like.

```python
def forw_9(v: tuple[int, ...], n: int) -> float:
    center = (n // 2, n // 2)
    dx, dy = abs(v[0] - center[0]), abs(v[1] - center[1])
    def weight_func(d: float) -> float:
        return 1 - (1 / (1 + np.exp(-0.1 * (d - 0.5 * n))))
    weight = weight_func(np.sqrt(dx**2 + dy**2))
    penalty = 1e-5 if dx == dy else 0
    if dx != dy:
        bonus = (np.abs(dx - dy) - 1) / (n - 1) if np.abs(dx - dy) < n - 1 else 0
    else:
        bonus = 0
    boundary_bonus = 0.1 * np.max([dx, dy])
    return ((dx**2 + dy**2 * weight) ** 0.5 + (1 - weight) * (abs(dx - dy) + penalty) + bonus + boundary_bonus)
```

This model was trained to construct isosceles-free sets on a 9x9 grid.

Funsearch Output and Results

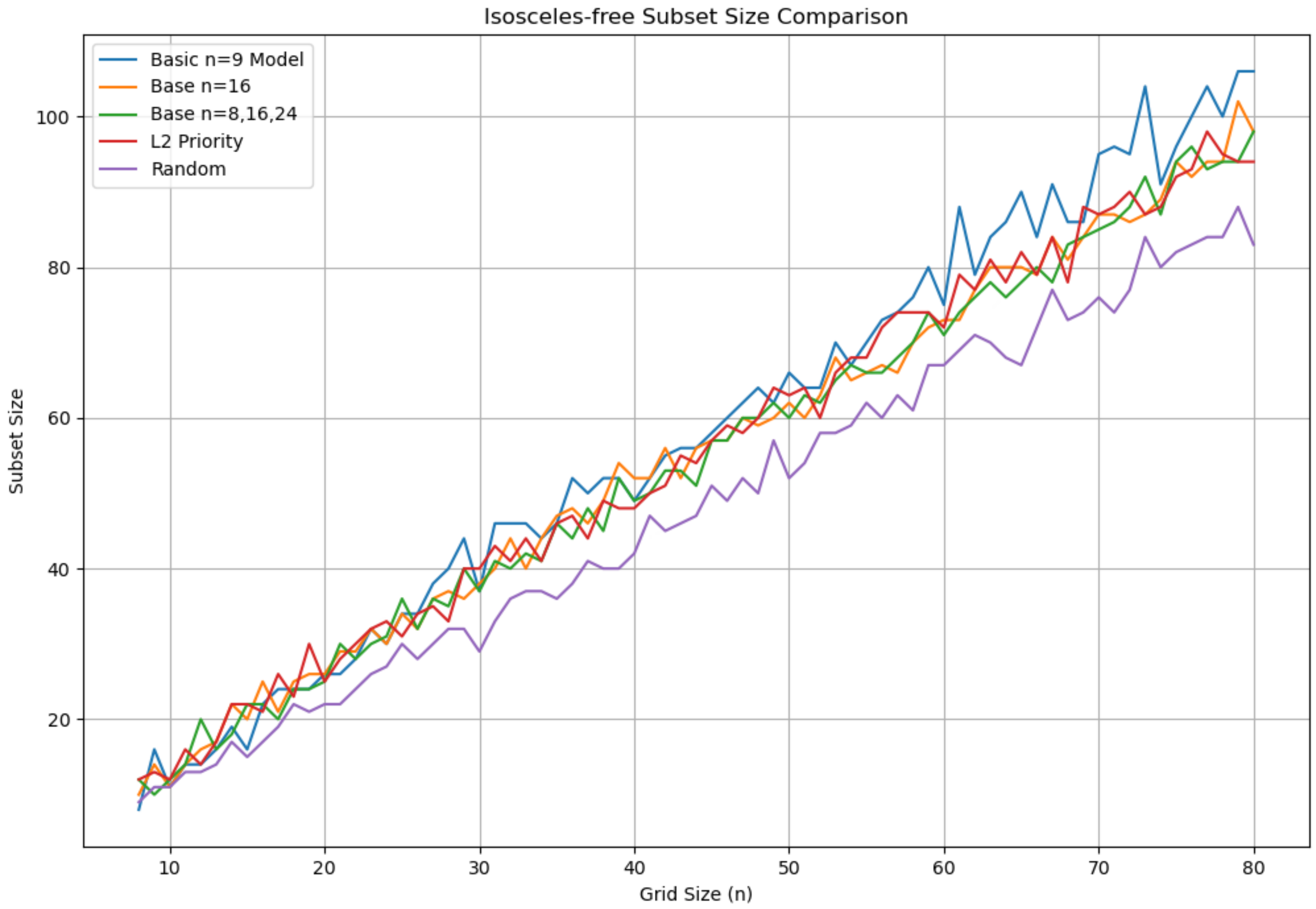
Advantage 1: We can see what the priority function looks like.



This model was trained to construct isosceles-free sets on a 9x9 grid. We can also visualize the priority function by plotting the priorities. There is, for example, a clear L2 preference (as expected).

Funsearch Output and Results

Advantage 2: We can test the generation mechanism on values it was not trained on.



The model trained on n=9 generalizes better than the other models. It even generalizes better than models trained on multiple different input values during training including a model that just prioritizes L2 distance from the center. So we can infer that the model is learning more than just that.

Funsearch Output and Results

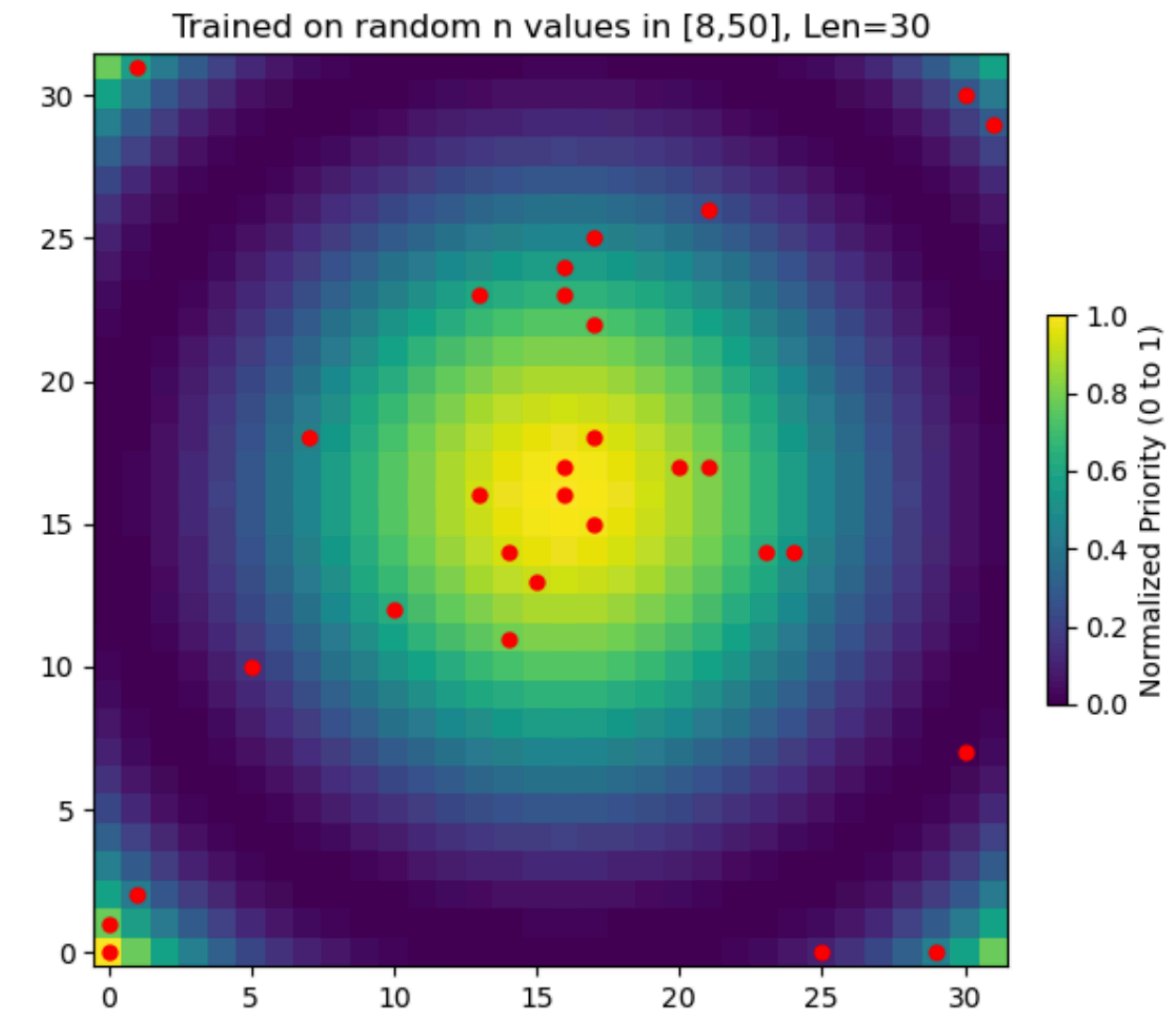
Advantage 3: It's relatively easy to change the code to create new experiments. If we want to 'mod out' the effect of the L2 norm, we could change the distance function to embed the problem on a torus. Everything else stays the same!

```
def distance(p1: Tuple[int, int], p2: Tuple[int, int]) -> float:
    return (p1[0] - p2[0])**2 + (p1[1] - p2[1])**2

def torus_distance(p1: tuple[int, int], p2: tuple[int,int], rows, cols):
    i1 = p1[0]
    j1 = p1[1]
    i2 = p2[0]
    j2 = p2[1]

    if j2 > j1:
        d1 = j2 - j1
    else:
        d1 = j1 - j2
    if d1 > cols - d1:
        d1 = cols - d1

    if i2 > i1:
        d2 = i2 - i1
    else:
        d2 = i1 - i2
    if d2 > rows - d2:
        d2 = rows - d2

    return d1**2 + d2**2
```

Funsearch Output and Results

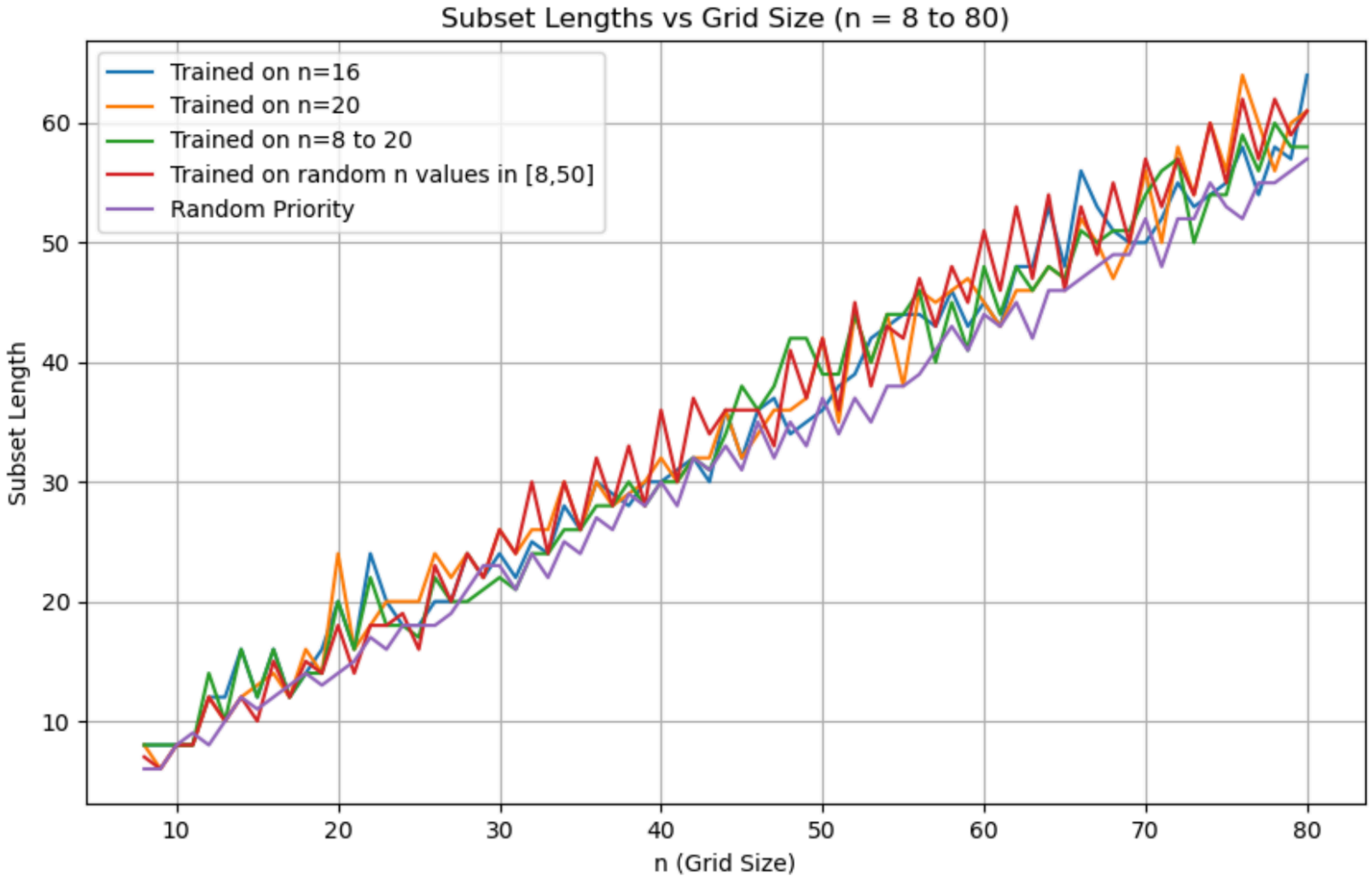
Advantage 3: It's relatively easy to change the code to create new experiments. If we want to 'mod out' the effect of the L2 norm, we could change the distance function to embed the problem on a torus. Everything else stays the same!

```
def tor_rand_points_8_50_v1(v: tuple[int, ...], n: int) -> float:
    """Returns the priority, as a floating number, of the vector v denoting the coordinates of a point in the n by n integer lattice.
        The priority function will be used to construct an isosceles-free subset of the lattice.
    """
    row, col = v
    distance = torus_distance((row, col), (n//2, n//2), n, n)
    return (1.0 - (distance / (n//2)) ** 2) ** 2 + 0.002 * np.abs(np.sin(0.01 * col) + 0.1 * np.sin(0.02 * row)) + 0.001 * np.cos(0.001 * (col + row))
```



Funsearch Output and Results

Advantage 3: It's relatively easy to change the code to create new experiments. If we want to 'mod out' the effect of the L2 norm, we could change the distance function to embed the problem on a torus. Everything else stays the same!



Funsearch Output and Results

Advantage 3: It's relatively easy to change the code to create new experiments.

A small change in the evaluation during training can result in a different but interesting problem.

```
def evaluate(n: int) -> int:
    """Returns the size of an isosceles-free subset of an n by n integer lattice."""
    subset = solve(n)
    return len(subset)
```

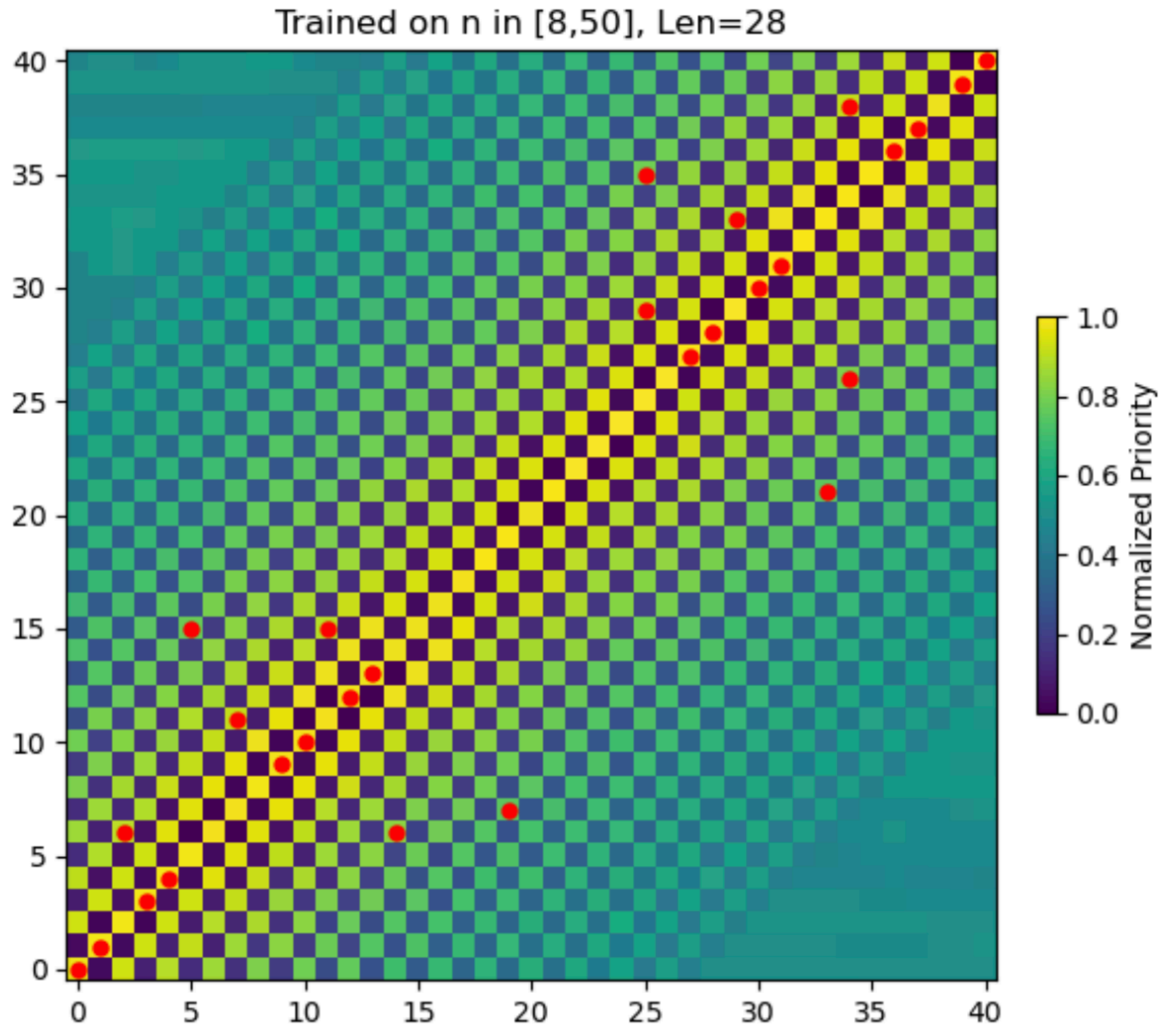
```
def evaluate(n: int) -> int:
    """Returns the size of an isosceles-free subset of an n by n integer lattice."""
    subset = solve(n)
    return -len(subset)
```

Here, we're finding small, maximal isosceles-free subsets of the lattice instead of large ones!

Funsearch Output and Results

Advantage 3: It’s relatively easy to change the code to create new experiments.

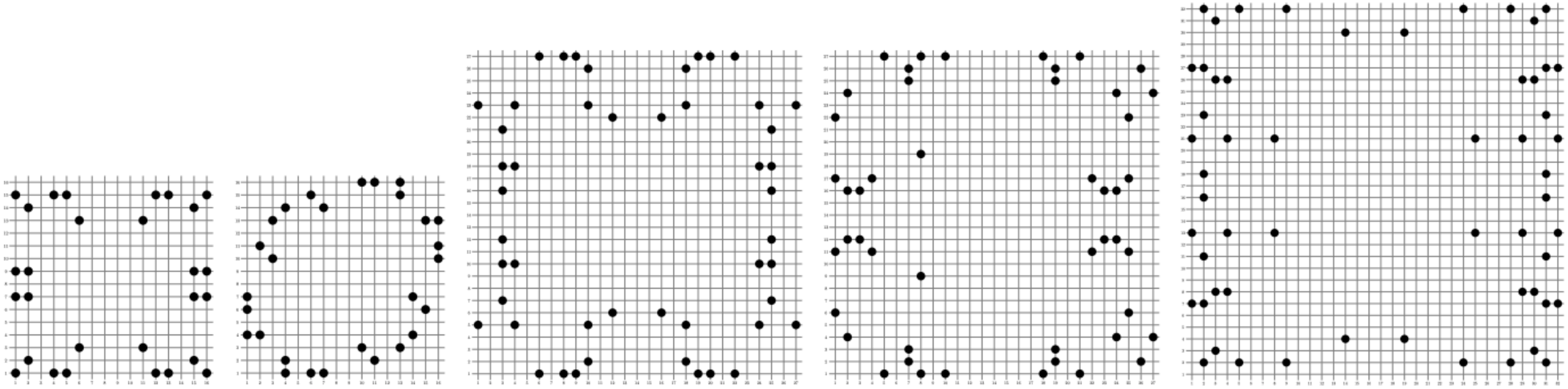
```python
###### Reverse Priorities (Removing points) ######
def rev_9(v: tuple[int,...], n: int) -> float:
    x, y = v
    vector_magnitude = np.linalg.norm([x, y])
    diff = abs(x - y)
    return 0.5 * (1.0 / (diff + 1) + 0.1 * (vector_magnitude ** 0.5)) + 0.05 * (n - vector_magnitude) - 0.001 * (x + y) + 0.0001 * (x ** 2 + y ** 2)
```



Funsearch Output and Results

Advantage 4: It is easy to build in prior knowledge. Just code it into the solve function.

We saw earlier that some of the best functions were found by enforcing some symmetry.

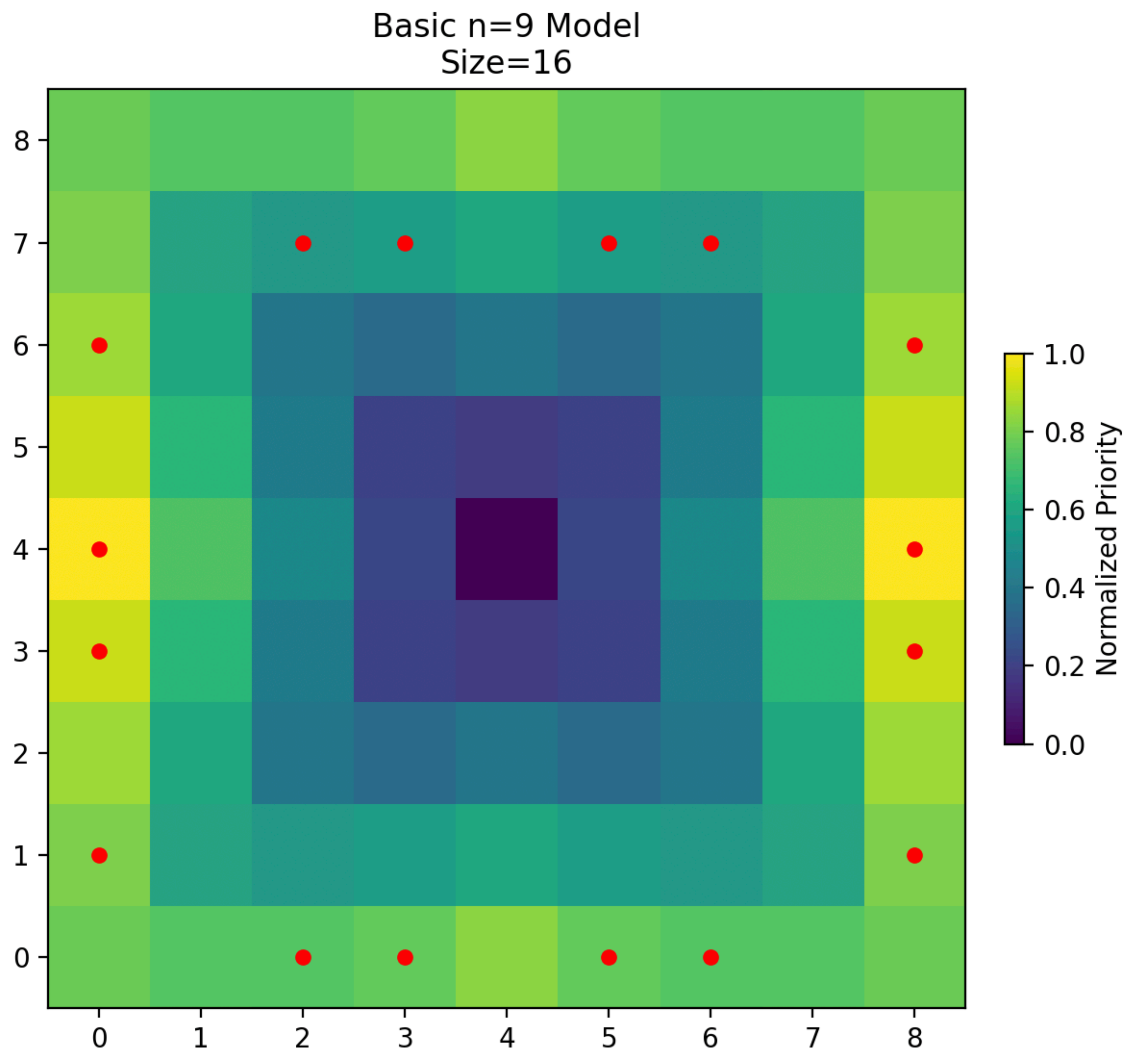


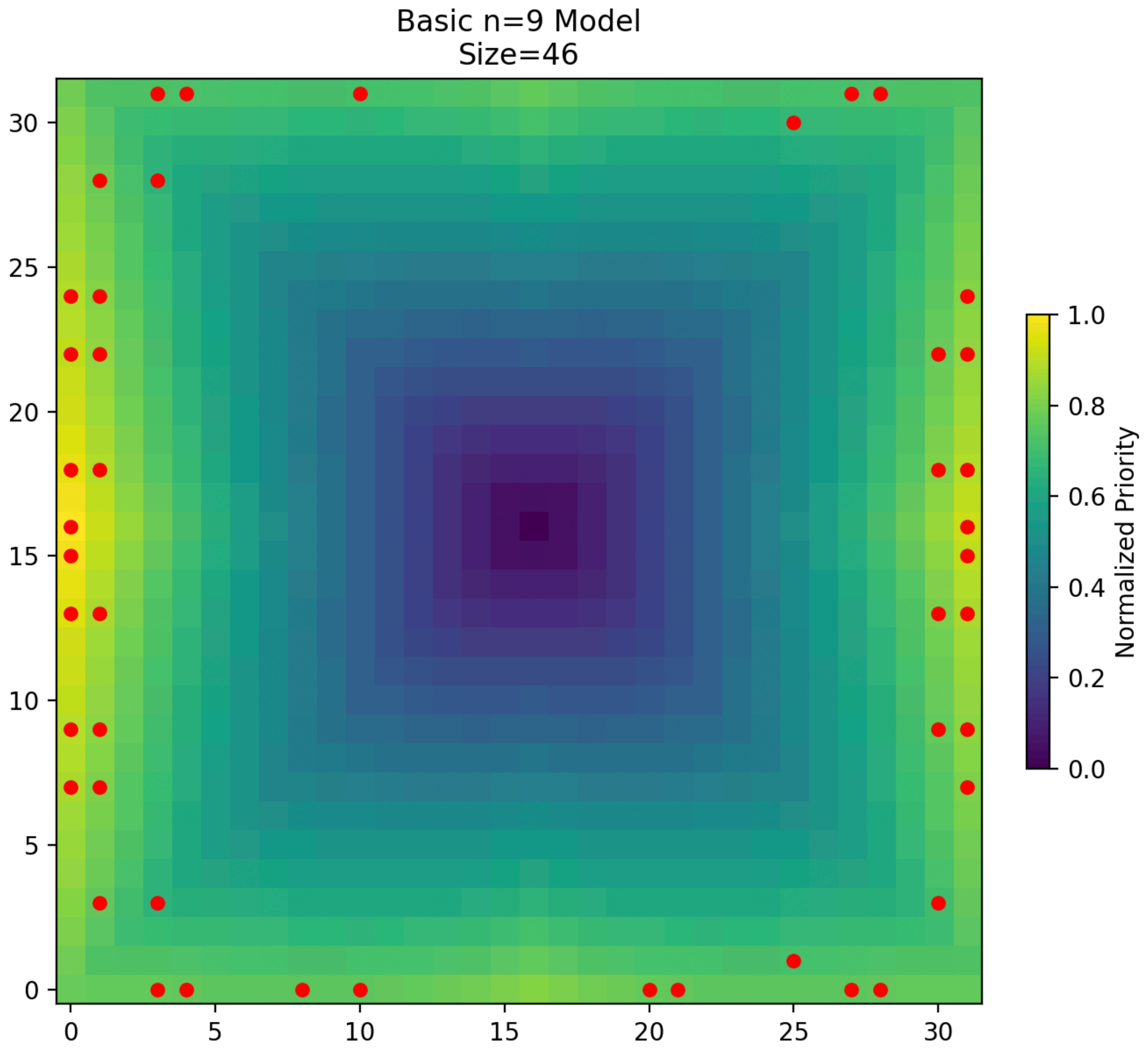
$f(16) = 28$ $f(16) = 28$ $f(27) = 48$ $f(27) = 48$ $f(32) = 56$

Funsearch Output and Results

Advantage 4: It is easy to build in prior knowledge. Just code it into the solve function.

We saw earlier that some of the best functions were found by enforcing some symmetry. Even if we didn’t see that beforehand, we see that the model trained on n=9 learned a symmetric solution. Almost true outside training distribution.





Funsearch Output and Results

Advantage 4: It is easy to build in prior knowledge. Just code it into the solve function.

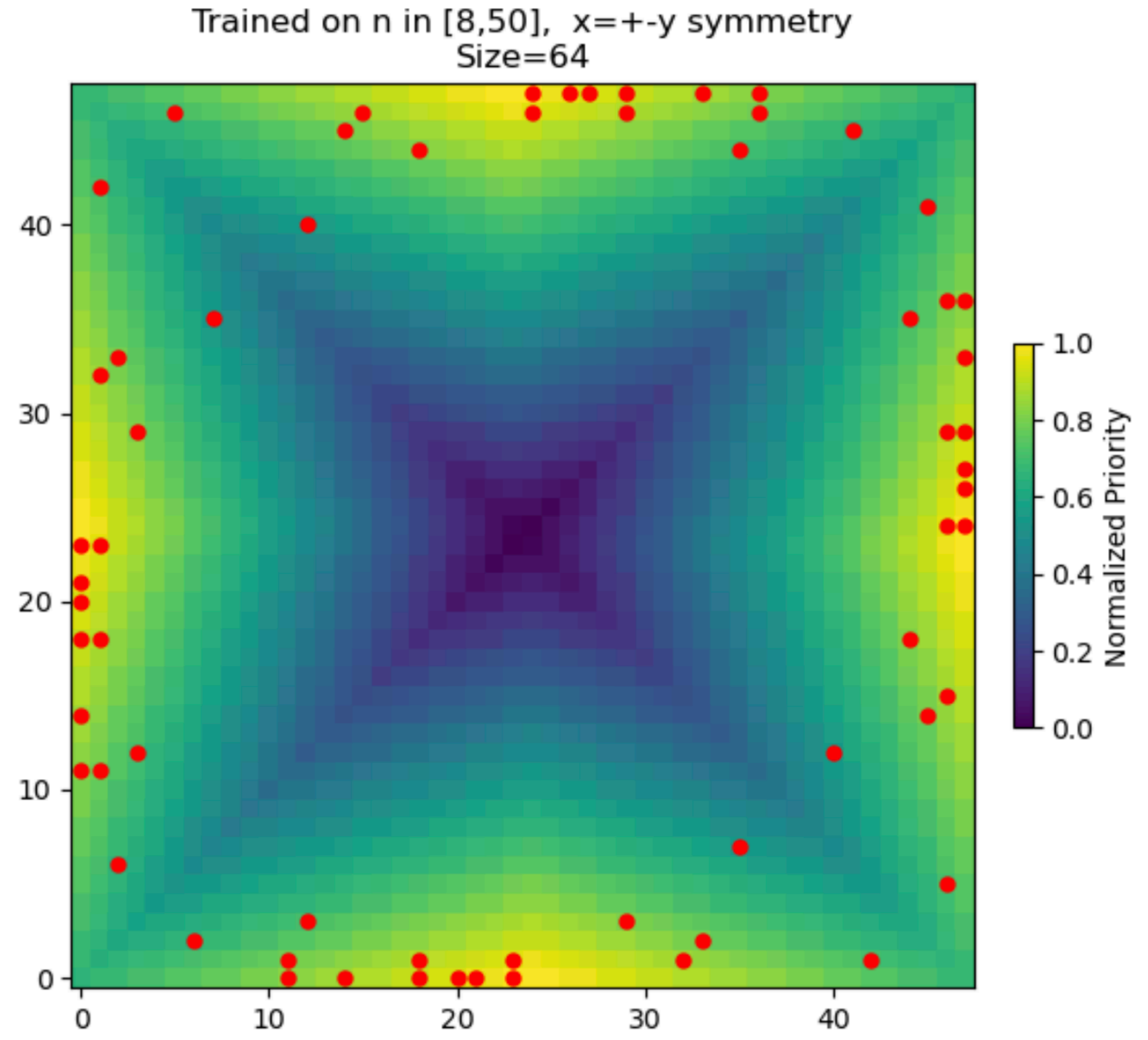
We saw earlier that some of the best functions were found by enforcing some symmetry. Even if we didn't see that beforehand, we see that the model trained on n=9 learned a symmetric solution. Almost true outside training distribution. So we can change the solve function to enforce symmetry.

```python
def solve_symmetry(n: int, priority_func: Callable, sym: str, **kwargs) -> List[Tuple[int, int]]:

    """Finds a large isosceles-free subset with enforced symmetry."""
    all_points = get_valid_points(n, sym)
    priorities = np.array([priority_func(p, n) for p in all_points], dtype=float)
    subset = []

    while np.any(priorities != -np.inf):
        max_index = np.argmax(priorities)
        point = all_points[max_index]
        new_points = generate_symmetric_points(point, n, sym)

        copy = subset.copy()
        copy.extend(new_points)
        if is_isosceles_free(copy):
            subset.extend(new_points)

        priorities[max_index] = -np.inf
    return subset
```

Funsearch Output and Results

Advantage 4: It is easy to build in prior knowledge. Just code it into the solve function.

```
def sym_xpmy_random_n_8_50(v: tuple[int, ...], n: int) -> float:
  """Returns the priority, as a floating number, of the vector v denoting the coordinates of a point in the n by n lattice…
  x, y = v
  diff = abs(x - y)
  midpoint = n // 2
  diagonal_distance = abs(x + y - n)
  odd_x_or_y = (x % 2 == 1) or (y % 2 == 1)
  return (diff + diagonal_distance + abs(np.abs(x - midpoint) - np.abs(y - midpoint)) + (1 if odd_x_or_y else 0)) / 2
```

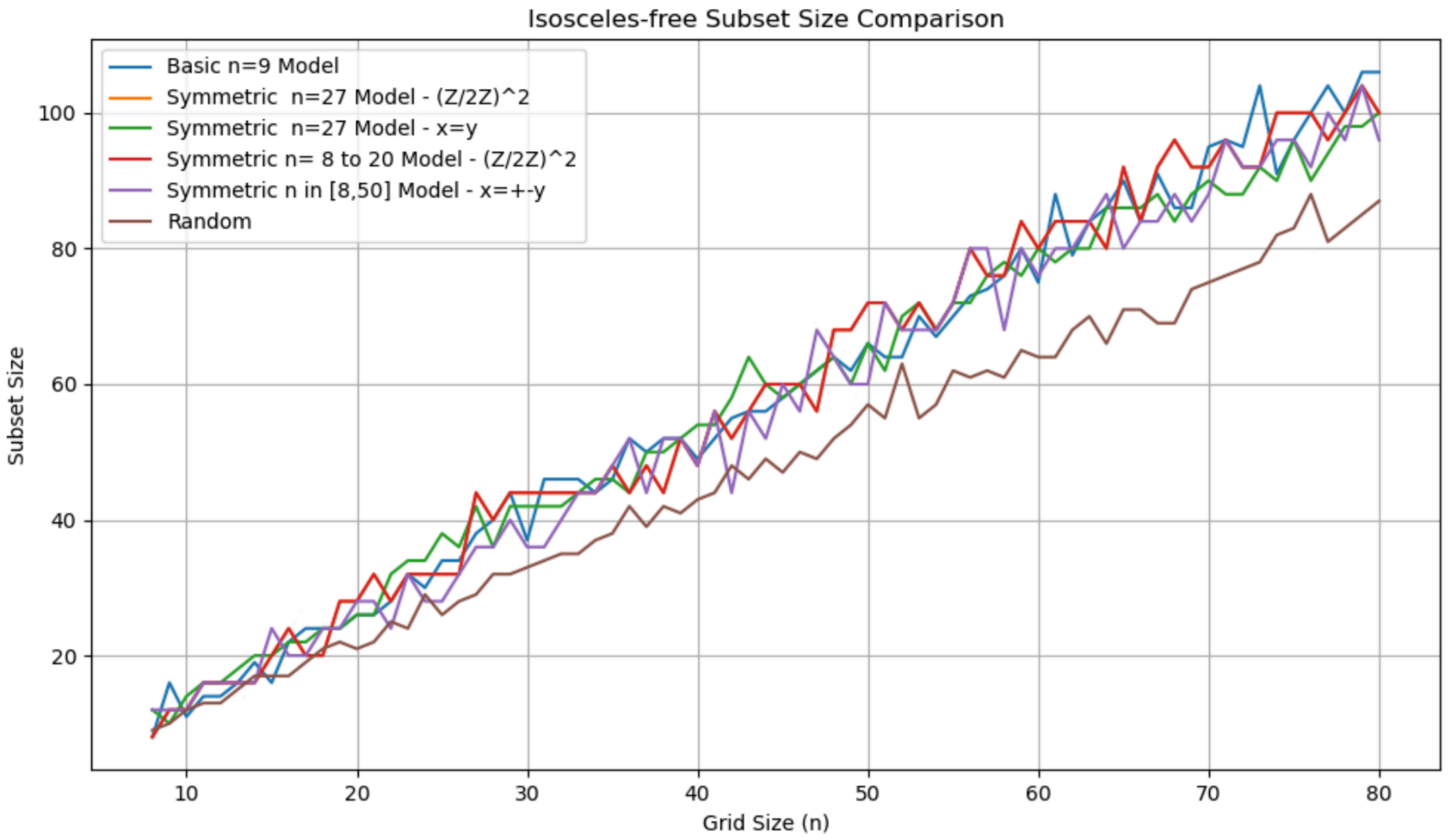


Funsearch Output and Results

Advantage 4: It is easy to build in prior knowledge. Just code it into the solve function.

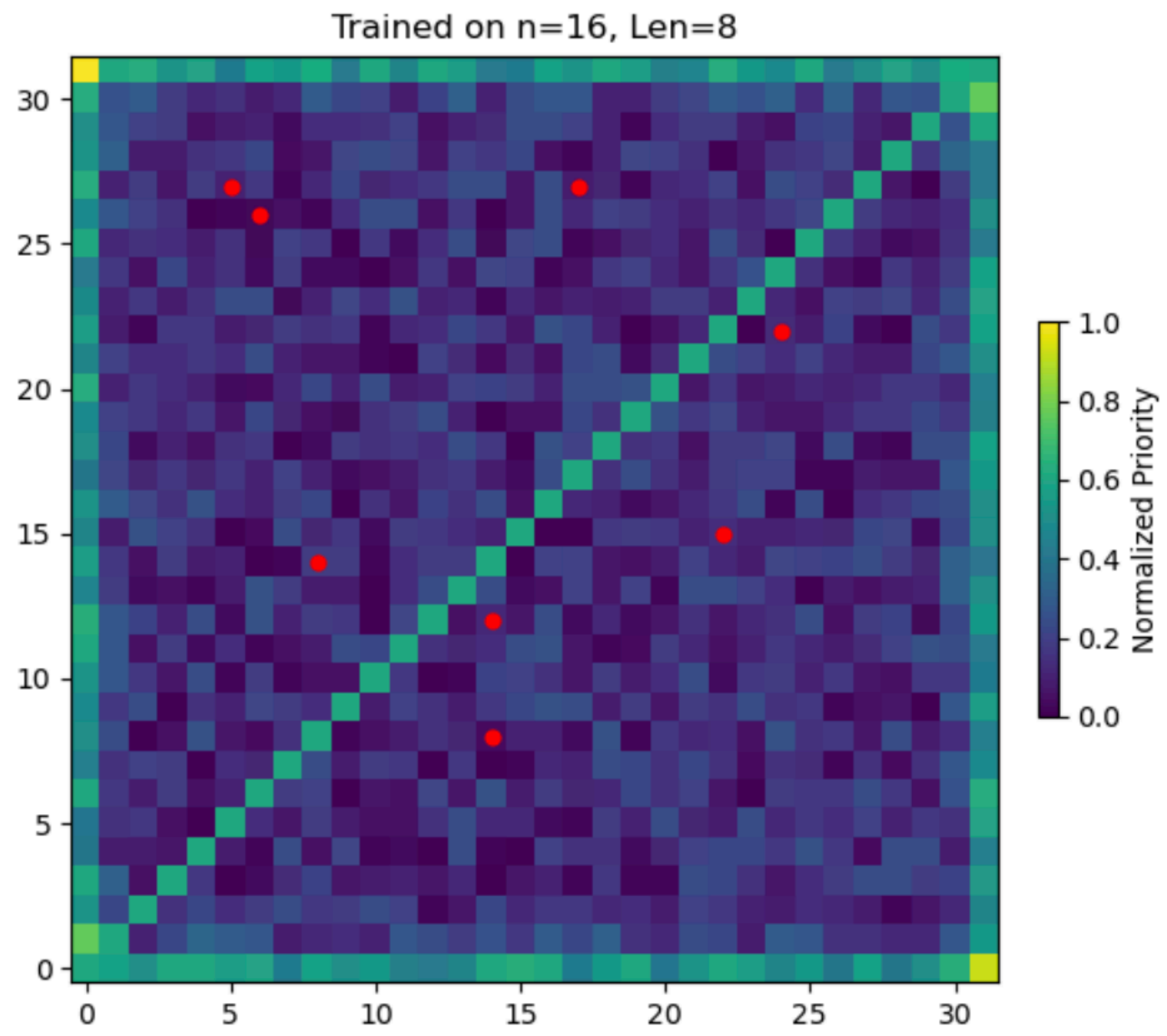
This gives us both better results and better generalization.

problem setup	n								
	12	**13**	**16**	**21**	**23**	**25**	**27**	**32**	**64**
maximum known	20	22	28	36	40	44	48	56	110
basic models	20	20	26	34	36	40	40	46	86*
symmetric models	20	22	28	36	40	40	44	52	96

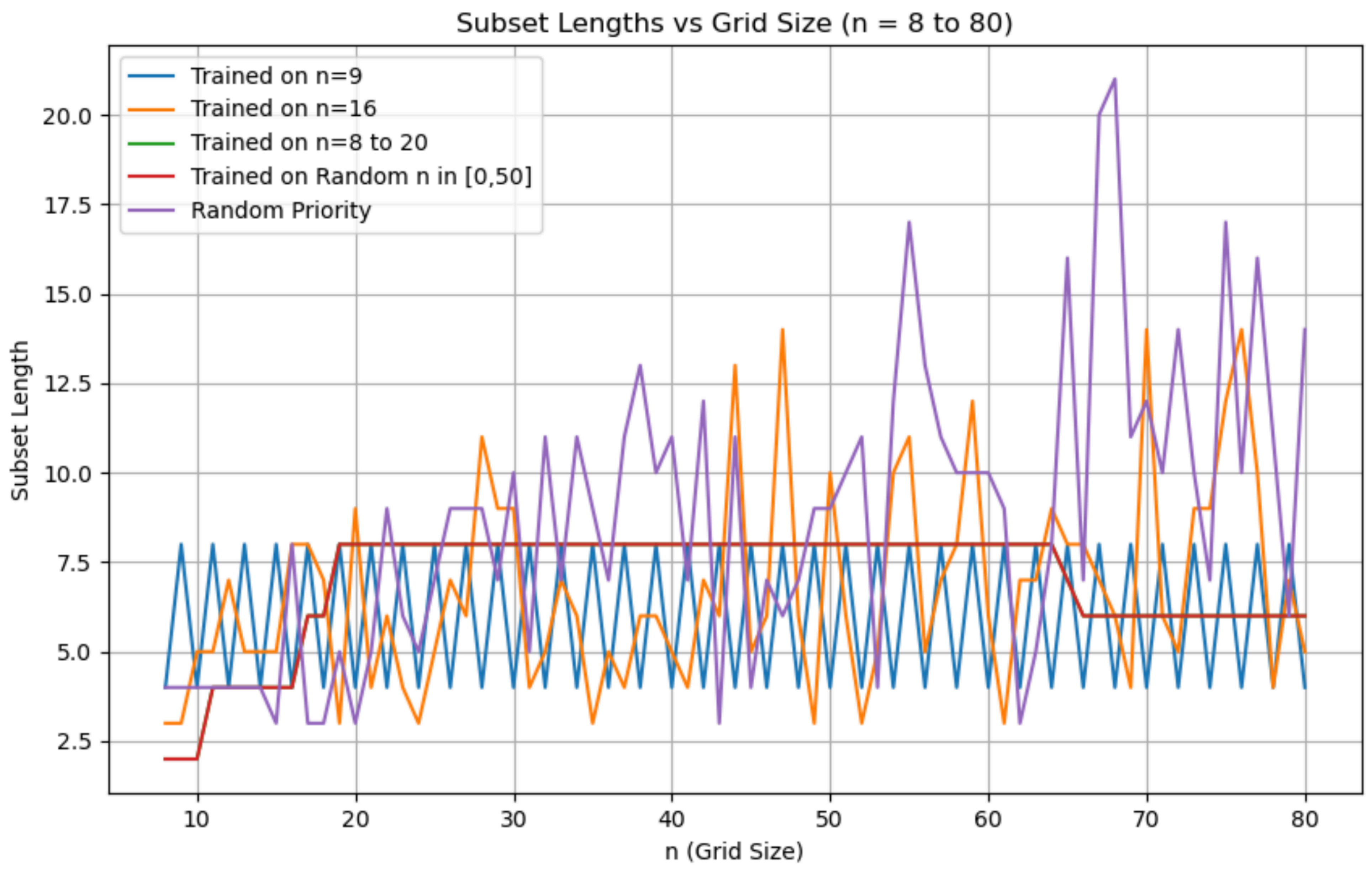


Funsearch Output and Results

Caution: Changing the approach, even in mathematically equivalent ways, can change the performance.

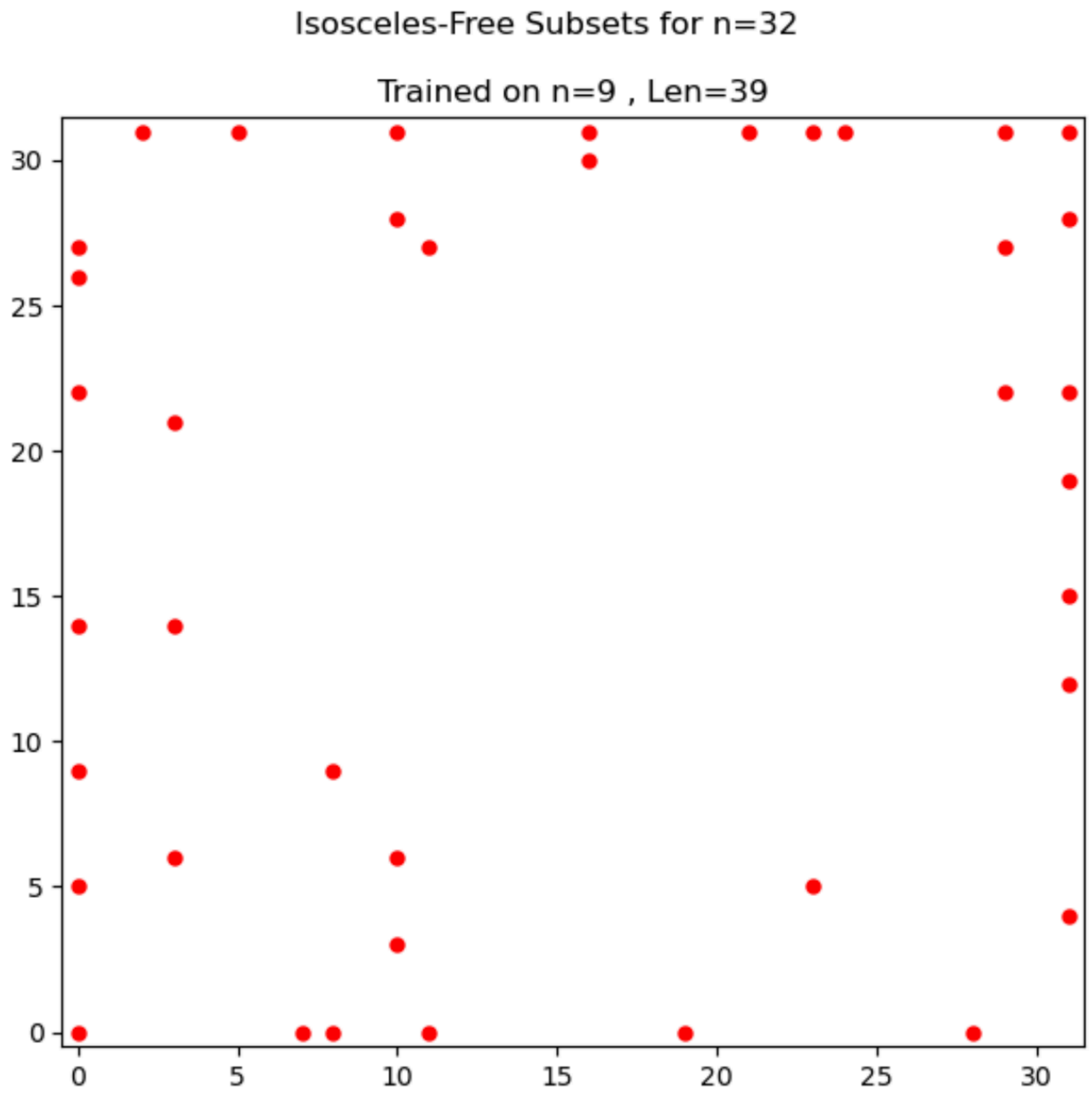


Instead of starting with an empty set and learning how to add points, we started with a dense set and learned how to remove points.

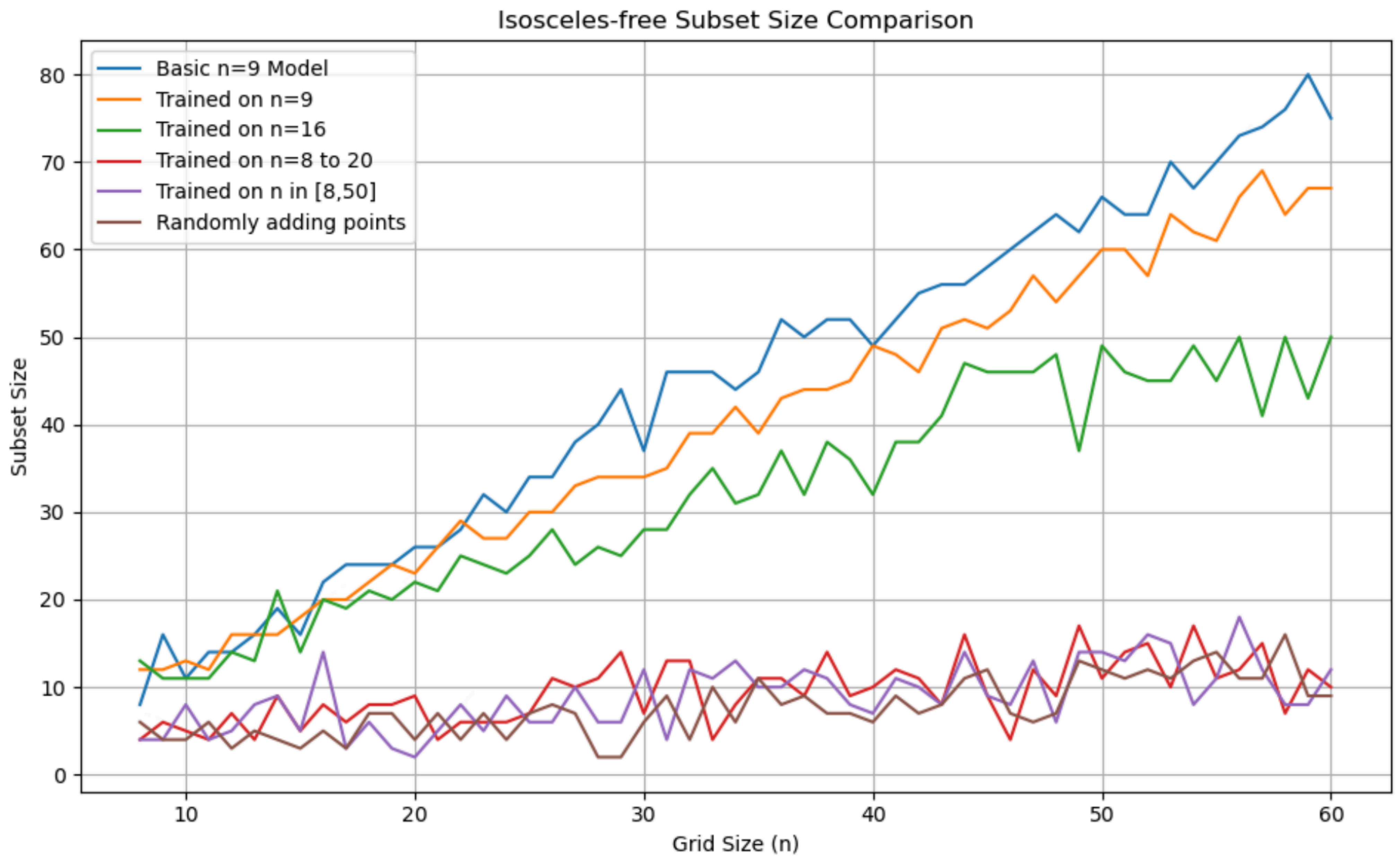


Funsearch Output and Results

Caution: Changing the approach, even in mathematically equivalent ways, can change the performance.

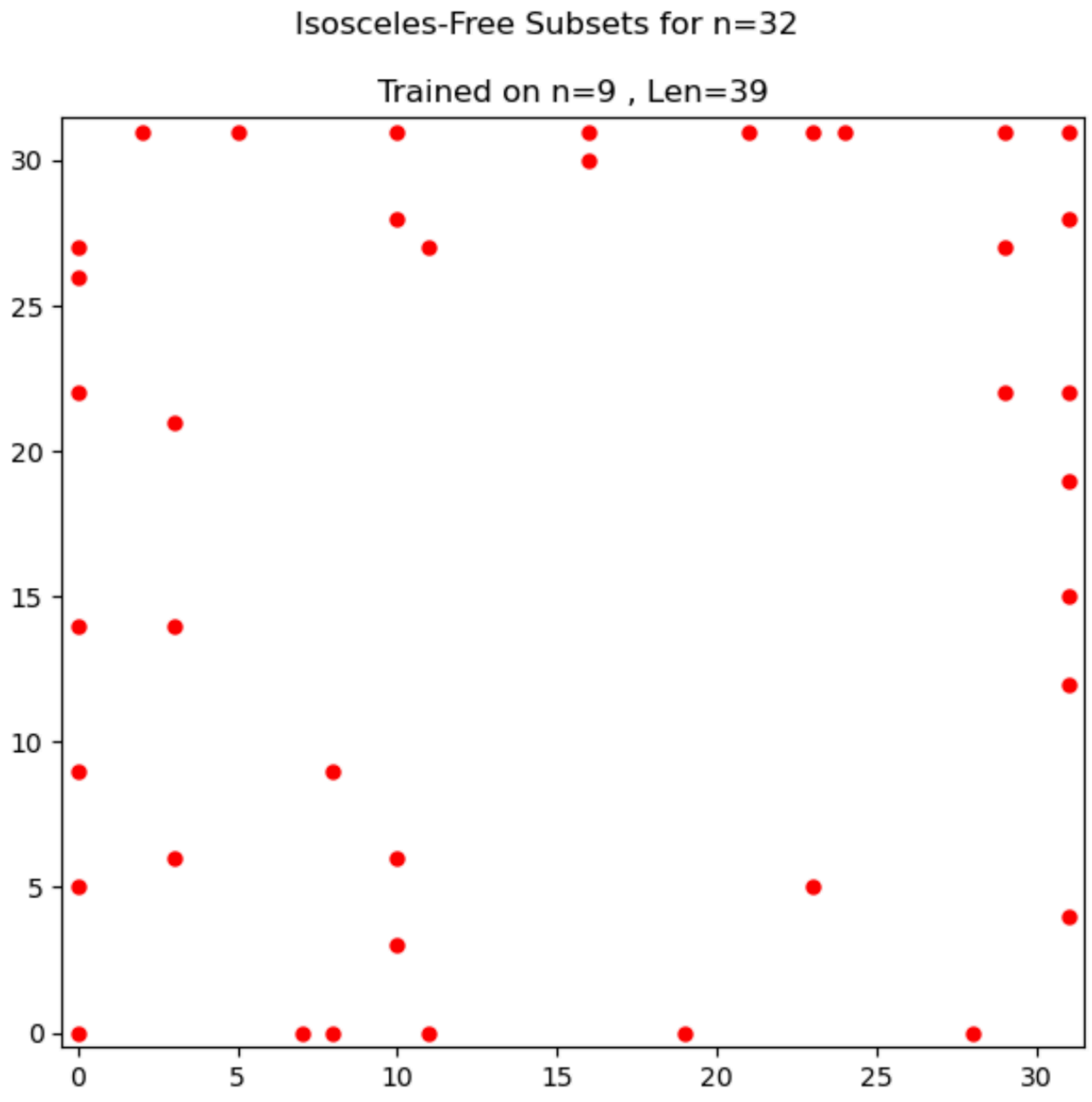


Instead of learning the weights on a greedy algorithm, we learn a priority function that takes in a subset and picks the next point to add.

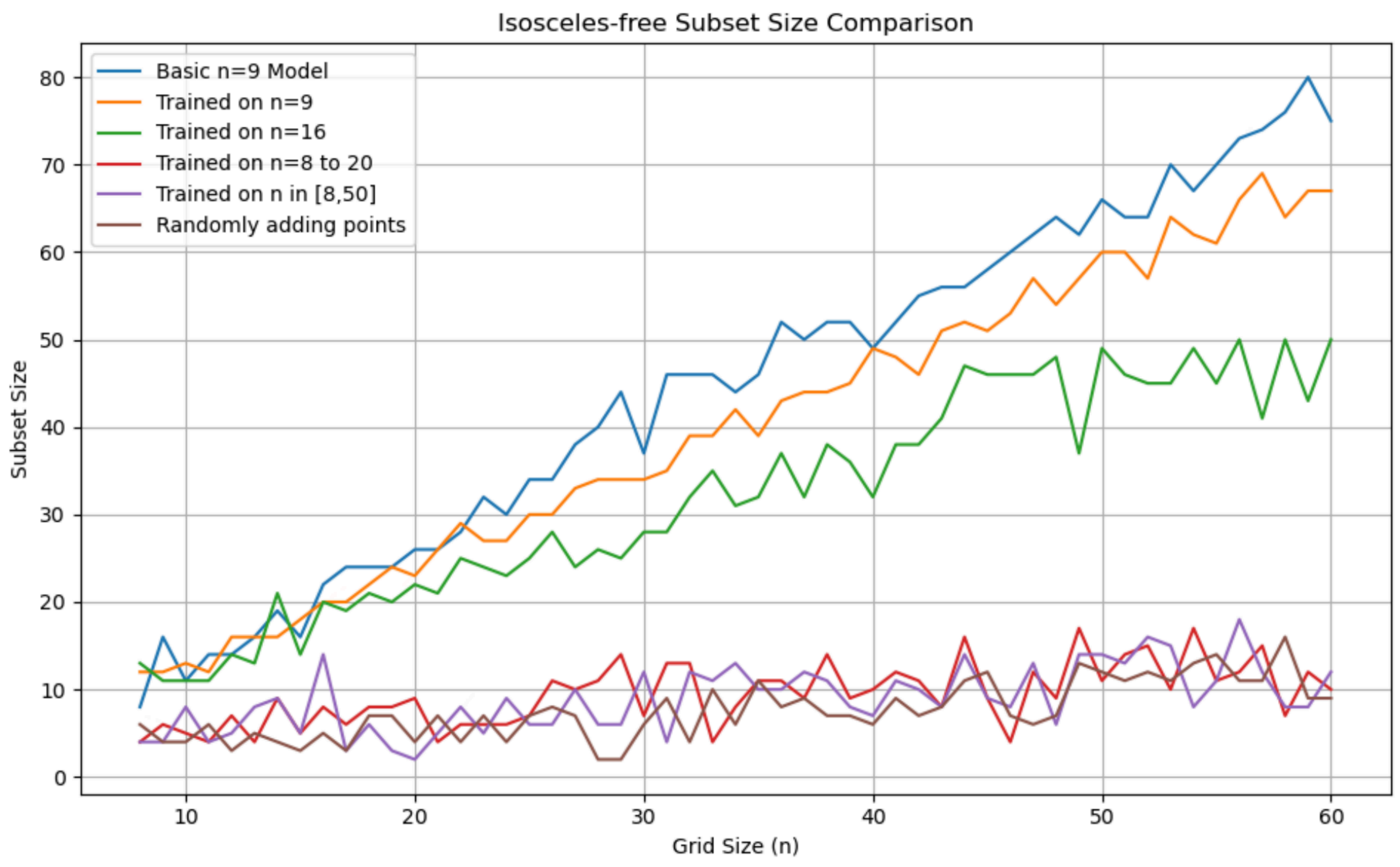


Funsearch Output and Results

Caution: Changing the approach, even in mathematically equivalent ways, can change the performance.



Instead of learning the weights on a greedy algorithm, we learn a priority function that takes in a subset and picks the next point to add.



Updates in this space

FunSearch has a successor, AlphaEvolve, DeepMind June 2025[11].

Google DeepMind

AlphaEvolve: A coding agent for scientific and algorithmic discovery

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*
Google DeepMind[1]

In this white paper, we present *AlphaEvolve*, an evolutionary coding agent that substantially enhances capabilities of state-of-the-art LLMs on highly challenging tasks such as tackling open scientific problems or optimizing critical pieces of computational infrastructure. *AlphaEvolve* orchestrates an autonomous pipeline of LLMs, whose task is to improve an algorithm by making direct changes to the code. Using an evolutionary approach, continuously receiving feedback from one or more evaluators, *AlphaEvolve* iteratively improves the algorithm, potentially leading to new scientific and practical discoveries. We demonstrate the broad applicability of this approach by applying it to a number of important computational problems. When applied to optimizing critical components of large-scale computational stacks at Google, *AlphaEvolve* developed a more efficient scheduling algorithm for data centers, found a functionally equivalent simplification in the circuit design of hardware accelerators, and accelerated the training of the LLM underpinning *AlphaEvolve* itself. Furthermore, *AlphaEvolve* discovered novel, provably correct algorithms that surpass state-of-the-art solutions on a spectrum of problems in mathematics and computer science, significantly expanding the scope of prior automated discovery methods (Romera-Paredes et al., 2023). Notably, *AlphaEvolve* developed a search algorithm that found a procedure to multiply two 4×4 complex-valued matrices using 48 scalar multiplications; offering the first improvement, after 56 years, over Strassen's algorithm in this setting. We believe *AlphaEvolve* and coding agents like it can have a significant impact in improving solutions of problems across many areas of science and computation.

Updates in this space

FunSearch has a successor, AlphaEvolve, DeepMind June 2025[11].

FunSearch [83]	*AlphaEvolve*
evolves single function	evolves entire code file
evolves up to 10-20 lines of code	evolves up to hundreds of lines of code
evolves code in Python	evolves any language
needs fast evaluation ($\leq$ 20min on 1 CPU)	can evaluate for hours, in parallel, on accelerators
millions of LLM samples used	thousands of LLM samples suffice
small LLMs used; no benefit from larger	benefits from SOTA LLMs
minimal context (only previous solutions)	rich context and feedback in prompts
optimizes single metric	can simultaneously optimize multiple metrics

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

So far in the course

What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

- So far in the course
- What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

- So far in the course
- What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

- So far in the course
- What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

- So far in the course
- What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

An advantage of generating algorithms over traditional representation is that we test generalization through runs.

So far in the course

What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

Data + Reasoning Integrated
Integrating both data and theory in the search process can improve search (AI Hilbert)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

An advantage of generating algorithms over traditional representation is that we test generalization through runs.

- So far in the course (blue)
- What we saw today + role in FunSearch and Algorithm Generation (green)

Relations to the workshop thus far

<u>Data-Driven Methods</u>
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

<u>Automated Reasoning</u>
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

<u>Data + Reasoning Iterated</u>
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

<u>Data + Reasoning Integrated</u>
Integrating both data and theory in the search process can improve search (AI Hilbert)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

An advantage of generating algorithms over traditional representation is that we test generalization through runs.

Prior information is used to build a skeleton of a program that we will use in the neural search process. Knowledge of the problem is essential to search.

- (blue) So far in the course
- (green) What we saw today + role in FunSearch and Algorithm Generation

Relations to the workshop thus far

Data-Driven Methods
Designing approaches that learn implicit functions from data (NNs, Transformers, SR)

Automated Reasoning
Automated theorem provers can help us build tools to auto formalize and verify knowledge.

Data + Reasoning Iterated
We can take functional forms learned on data and test on theory to study outputs. (AI Descartes)

Data + Reasoning Integrated
Integrating both data and theory in the search process can improve search (AI Hilbert)

When we don't have data, these methods can still work via self improvement by generating data and gradually training on best examples

For classes of problems where verification is easy but solving is hard, we can leverage fast inference at a large scale to find solutions

An advantage of generating algorithms over traditional representation is that we test generalization through runs.

Prior information is used to build a skeleton of a program that we will use in the neural search process. Knowledge of the problem is essential to search.

- Blue: So far in the course
- Green: What we saw today + role in FunSearch and Algorithm Generation

References

[1] Terry Tao Blog Post: Link here (Open question: best bounds for cap sets, Feb 2007)

[2] Ellenberg, Jordan S., and Lalit Jain. "Convergence rates for ordinal embedding." *arXiv preprint arXiv:1904.12994* (2019).

[3] Spot, Boston Dynamics Robot: https://bostondynamics.com/blog/starting-on-the-right-foot-with-reinforcement-learning/

[4] Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." *arXiv preprint arXiv:1712.01815* (2017).

[5] Wagner, Adam Zsolt. "Constructions in combinatorics via neural networks." *arXiv preprint arXiv:2104.14516* (2021).

[6] Charton, François, 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).

[7] Roziere, Baptiste, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi et al. "Code llama: Open foundation models for code." *arXiv preprint arXiv:2308.12950* (2023).

[8] Romera-Paredes, B., Barekatain, M., Novikov, A. *et al.* Mathematical discoveries from program search with large language models. *Nature* **625**, 468–475 (2024). https://doi.org/10.1038/s41586-023-06924-6

[9] Matej Balog CMSA Talk - https://www.youtube.com/watch?v=VJ_meFdGwE8&t=753s

[10] Ellenberg, Jordan S., Cristofero S. Fraser-Taliente, Thomas R. Harvey, Karan Srivastava, and Andrew V. Sutherland. "Generative modeling for mathematical discovery." *arXiv preprint arXiv:2503.11061* (2025).

[11] Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov et al. "AlphaEvolve: A coding agent for scientific and algorithmic discovery." *arXiv preprint arXiv:2506.13131* (2025).

Karan Srivastava
ksrivastava4@wisc.edu