From Black Box to Bijection: Interpreting Machine Learning to Build a Zeta Map Algorithm

Xiaoyu (Coco) Huang, **Blake Jackson**, Kyu-Hwan Lee

FPSAC 2026, Seattle

July 13, 2026

The goal of this line of research

Many questions in (algebraic) combinatorics can be distilled down to

Prototypical Question

Find a combinatorial interpretation, algorithm, or bijection (or a counterexample).

How to do math research in the words of George Pólya:

Pólya's method

First guess, then prove.

Works well when data is small (human scale), much harder when data is big

Proposed path forward

Use machine learning to augment Pólya's method: first guess (with the machine), then prove

AI in combinatorics: what it can and cannot do (yet)

Good at

- Searching for constructions and counterexamples (Wagner 2021; PatternBoost 2024; AlphaEvolve 2025)
- Classifying invariants such as Sato–Tate groups and number fields (He–Lee–Oliver 2022, 2023)
- Finding patterns in data too big for humans, e.g. murmurations (He–Lee–Oliver–Pozdnyakov 2025)

Not good at

- Producing verifiable objects: a prediction is not a proof
- Sparse-reward search spaces and discontinuous objectives
- Explaining *why* it works: high accuracy alone gives no insight

The horizon

Mechanistic interpretability bridges a gap: predictive model $\longrightarrow$ provable algorithm.

q,t-Catalan numbers

$C_n(q,t) \in \mathbb{Z}_{\geq 0}[q,t]$: bigraded Hilbert series of the alternating part of diagonal harmonics (Garsia–Haiman 1996); by construction $C_n(q,t) = C_n(t,q)$.

Theorem 1 (Haglund 2003, Haiman 2000)

There exist nonnegative statistics area, bounce, and dinv on Dyck paths such that

$$C_n(q,t) = \sum_{w \in \mathcal{D}(n)} q^{area(w)}\, t^{bounce(w)} = \sum_{w \in \mathcal{D}(n)} q^{dinv(w)}\, t^{area(w)}$$

Question 1 (Answered: Haglund 2003, ...)

Is there a bijection on Dyck paths with $(dinv, area) \mapsto (area, bounce)$? Yes: the zeta map ζ.

Question 2 (Open: some partial answers and conjectures)

Is there a bijection on Dyck paths with $(dinv, area) \mapsto (area, dinv)$?

▸ how to compute the statistics

Warning: which "zeta map"?

The literature contains several maps all answering to the name ζ:

- **Area sequence descriptions** (Andrews–Krattenthaler–Orsina–Papi 2002; Haglund 2003) $(\mathit{dinv}, \mathit{area}) \mapsto (\mathit{area}, \mathit{bounce})$
- **Sweep map descriptions** (Armstrong–Loehr–Warrington 2015; Thomas–Williams 2018) $(\mathit{area}, \mathit{bounce}) \mapsto (\mathit{dinv}, \mathit{area})$
- **SageMath**: `area_dinv_to_bounce_area_map()` $= \mathrm{rev} \circ \zeta_{\text{Haglund}}$, because its bounce path runs in the reverse direction from Haglund's
  (rev = read the word backward with $0 \leftrightarrow 1$ swapped, i.e., flip across $y = -x$).

Which "zeta"?

We generated training data with SageMath $\Rightarrow$ our model learned $\mathrm{rev} \circ \zeta$.

The idea: use ζ as a playground

I (and others) really want to answer the open question.

ζ answers a similar question and is well understood with multiple descriptions

$\Rightarrow$ use it as a controlled playground for discovering bijections with ML:

- data is exactly generable,
- correctness is exhaustively checkable at high n,
- the desired output is a *discrete algorithm*, not a numerical approximation.

Dataset

$\left[\, w \,,\ \mathrm{rev} \circ \zeta(w) \,\right]_{w \in \mathcal{D}(13)}$; $\quad |\mathcal{D}(13)| = 742{,}900$; $\quad w =$ binary word of length 26

Transformers in one slide

Jay Alammar's "The Illustrated Transformer"

- Sequence-to-sequence translation: here, Dyck word $\to$ Dyck word.
- The key mechanism for us is **cross-attention**: at each output step, the decoder places weights on the input positions. *Which inputs does it consult, and when?*
  "Attention patterns."

Experiments: transformers learn the zeta map easily

- Standard architectures (4 layers, 8 heads, $d = 256$): $> 99\%$ accuracy for $n = 11, \ldots, 16$. Robust across encoder-only / decoder-only / encoder–decoder variants.
- High accuracy alone teaches us nothing about combinatorics.

Compression hypothesis

If a tiny model still solves the task perfectly, it must implement a simple algorithm, and that simple algorithm might be visible to the naked eye.

- **Minimal Dyck Transformer**: 1 encoder layer, 1 decoder layer, 1 attention head, $d = 128$; about 3.4×10^5 parameters.
- Trained on all of $\mathcal{D}(13)$: 100% accuracy, consistently across random seeds.

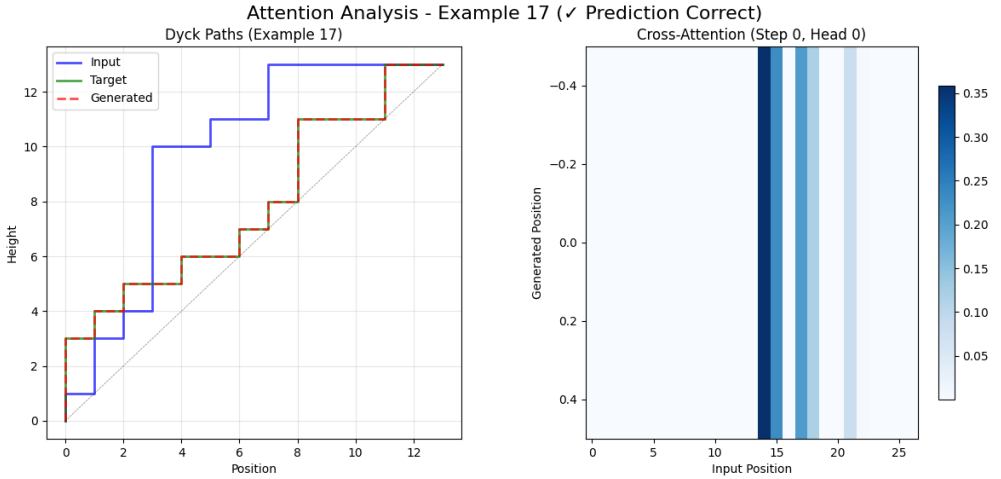
▸ architecture details

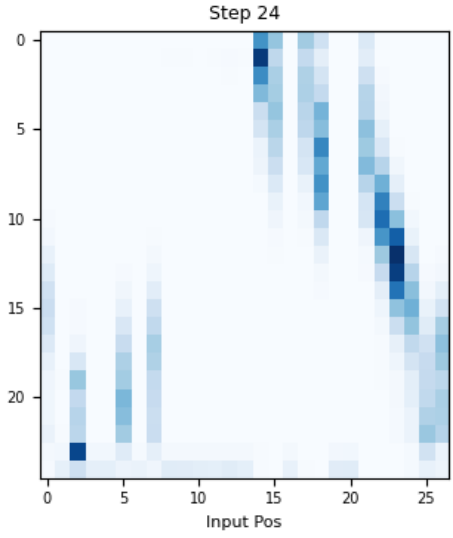
Three cross-attention patterns

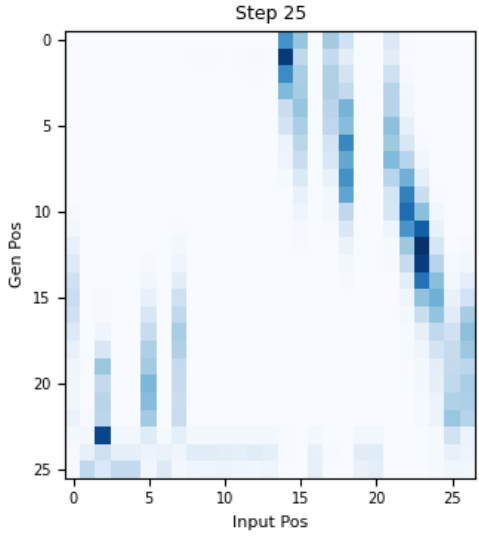
1. **Highest-level selection.** At the first output steps, attention concentrates on input positions of maximal level ℓ_i.
2. **North steps are ignored.** Positions with $w_i = 1$ receive essentially no cross-attention at any generation step.
3. **Triangular sequential shift.** After selecting a high-level East step i, attention slides to the next East step $i + 1$, whose level is $\ell_i - 1$. The model walks down the levels.

Together, these suggest the decoder *selects by level and then traverses*. Not a lookup table!

Step 0 cross-attention




Step 24
0
5
10
15
20
0
5
10
15
20
25
Input Pos


Step 25
Gen Pos
0
5
10
15
20
25
0
5
10
15
20
25
Input Pos

Are these patterns real? Ablation and probing

Attention weights are suggestive, not proof of mechanism. So we experiment:

Causal ablation

Force the decoder's attention on every North step to zero, at every generation step.

Result: accuracy drops by $\sim 1.5\%$. The North steps really are unnecessary at decoding time.

Linear probing

Train a linear map from encoder hidden states h_i to levels ℓ_i.

Result: levels are linearly recoverable, so the encoder computes ℓ without ever being told to.

Two-phase picture: **encoder** computes the level structure; **decoder** selects and traverses.

▸ probing figures ▸ ablation curve

The scaffolding map φ

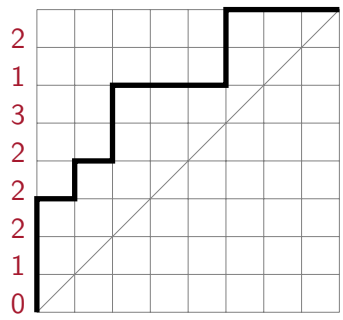
Setup: levels ℓ_i; East steps $R = \{i : w_i = 0\}$; *peaks* i with $(w_i, w_{i+1}) = (1, 0)$.

1. Initialize $out = [\,]$, $agents = [\,]$, $cur\_level = \max\{\ell_i : i \text{ a peak}\}$.
2. While $|out| < 2n$:
   - 2.1 Queue = (peaks at *cur_level*) $\cup$ (agents); sort decreasingly; append each w_i to *out*.
   - 2.2 Move each agent: right if in R (staying in R), left if not in R (staying out of R); remove agents with no valid move.
   - 2.3 Each peak j at *cur_level* spawns agents at $j+1$ (if in R) and $j-1$ (if not in R).
   - 2.4 Decrease *cur_level* by 1.
3. Return *out*.

Remark 1 (Mnemonic)

A **shuttle** arrives at each peak; out come a **worker**, who moves right along East steps, and a **ninja**, who moves left along North steps. Everyone reports their w_i's, level by level.

Worked example ($n = 8$)



$w = 1\,1\,1\,0\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0$

Queues (positions, sorted decreasingly per level):

$$
\begin{aligned}
Q_4 &= (8) && \to\ 1 \\
Q_3 &= (13, 9, 7, 5, 3) && \to\ 1\,0\,1\,1\,1 \\
Q_2 &= (14, 12, 10, 6, 4, 2) && \to\ 0\,1\,0\,0\,0\,1 \\
Q_1 &= (15, 11, 1) && \to\ 0\,0\,1 \\
Q_0 &= (16) && \to\ 0
\end{aligned}
$$

$\varphi(w) = 1\,|\,1\,0\,1\,1\,1\,|\,0\,1\,0\,0\,0\,1\,|\,0\,0\,1\,|\,0$

Main theorem

Theorem 2 (Huang–J.–Lee 2025+)

The scaffolding map equals the zeta map up to a reversal convention: $\varphi = \mathrm{rev} \circ \zeta$.

Proof strategy: both maps are determined by the same level-by-level data. We match the two descriptions for each level k, on the next slide.

Remark 2

Reminiscent of sweep maps, but organized around the symmetry at the *peaks*. To our knowledge, this is the first new sequential-output algorithm found by interpreting a neural network.

Proof idea, on the running example

Write $\tilde{a}(w) = (a_n, \ldots, a_1)$ for the reversed area sequence; here $\tilde{a}(w) = (2, 1, 3, 2, 2, 2, 1, 0)$.

- P_k: the subsequence of $\tilde{a}(w)$ with entries in $\{k, k-1\}$. These determine $\mathrm{rev} \circ \zeta$.
- Q'_k: replace each position i in the scaffolding queue Q_k by b_i, its column length in the path. These determine φ.

k	P_k from $\mathrm{rev} \circ \zeta$	queue Q_k (positions)	Q'_k from φ
4	(3)	(8)	(3)
3	$(2, \mathbf{3}, 2, 2, 2)$	$(13, \mathbf{9}, 7, 5, 3)$	$(2, \mathbf{3}, 2, 2, 2)$
2	$(\mathbf{2}, 1, \mathbf{2}, \mathbf{2}, \mathbf{2}, 1)$	$(\mathbf{14}, 12, \mathbf{10}, \mathbf{6}, \mathbf{4}, 2)$	$(\mathbf{2}, 1, \mathbf{2}, \mathbf{2}, \mathbf{2}, 1)$
1	$(\mathbf{1}, \mathbf{1}, 0)$	$(\mathbf{15}, \mathbf{11}, 1)$	$(\mathbf{1}, \mathbf{1}, 0)$
0	$(\mathbf{0})$	$(\mathbf{16})$	$(\mathbf{0})$

Bold orange: East-step positions ($w_i = 0$, the workers) and their entries, which equal k. Blue: North-step positions ($w_i = 1$, the shuttles and ninjas) and their entries, which equal $k - 1$.

Every row matches: $P_k = Q'_k$ for all k, hence $\varphi = \mathrm{rev} \circ \zeta$. □

The general workflow

1. Train on exact combinatorial data;
2. **compress** until the model is barely big enough;
3. examine attention patterns / representations for recurring structural signals;
4. test the signals **causally** (ablation, probing);
5. abstract the mechanism into a combinatorial procedure;
6. **prove it**

Nothing here is specific to ζ: the procedure applies wherever the target is a bijection or algorithm on exactly generable data.

Future directions

Question 2

Find a combinatorial bijection on Dyck paths interchanging *area* and *dinv* (or *area* and *bounce*).

Here we knew the input/output pairs. What if we don't?

- "Piece of straw in the haystack": greedily construct *some* statistic-swapping bijection, train, interpret. Issue: for $n = 8$ the number of area–dinv swapping bijections has 435 digits (OEIS A382909).
- Genetic/evolutionary search (FunSearch, AlphaEvolve). Issue: highly discontinuous valuation.
- Reinforcement learning. Issue: good at swapping statistics, bad at staying bijective.

Thank you!

Open-source code:

`https://github.com/cocoxhuang/DyckTransformer`

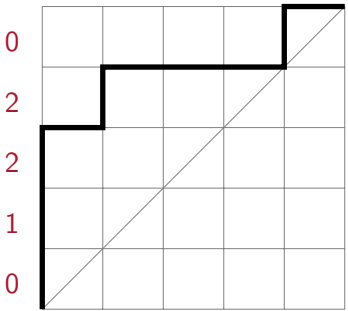
`https://github.com/AxiomMath/axplorer`

Visit me at ICARM!

Workshops, summer schools, & research visits:

`https://icarm.io`

Back slide: statistics on Dyck paths



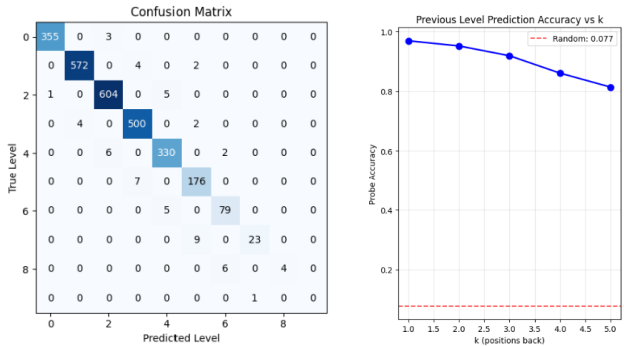
The red numbers are the area sequence, read bottom to top.

The *area sequence* $(a_1, \ldots, a_n)$: a_i counts the full boxes between the path and the diagonal in row i, rows numbered from the bottom. Here $(a_i) = (0, 1, 2, 2, 0)$.

- $area(w) = \sum_i a_i = 0 + 1 + 2 + 2 + 0 = 5$
- $dinv(w) = \#\{(i, j) \mid i < j,\ a_i - a_j \in \{0, 1\}\} = \#\{(1, 5), (2, 5), (3, 4)\} = 3$
- $bounce(w)$: shoot a billiard from $(0, 0)$ heading North; at the start of each East step turn East until the diagonal, then turn North; record the diagonal hits $(j_1, j_1), \ldots, (j_k, j_k)$. Then $bounce(w) = \sum_i (n - j_i)$.

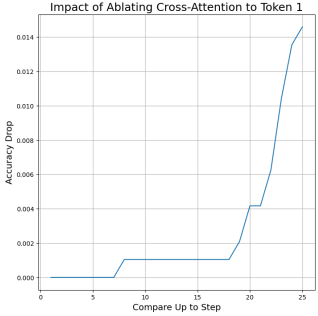
return to talk

Back slide: probing details



Left: confusion matrix of a linear map trained to predict the level ℓ_i from the encoder hidden state $h_i \in \mathbb{R}^{128}$ at position i. Errors concentrate near the diagonal, so the encoder knows the levels, but fuzzily. Right: accuracy of predicting the level ℓ_{i-k} of an *earlier* position from h_i, as a function of the offset k. The states also carry some memory of preceding positions.

Back slide: ablation curve



The intervention: in the decoder's cross-attention, the weight on every input position i with $w_i = 1$ is forced to zero (masked in the logits before the softmax), at every generation step. Horizontal axis: the generated word is compared with the correct target up to step k. Vertical axis: the drop in accuracy caused by the intervention. The drop stays below 1.5% even at the final step.

return to talk

Back slide: architecture of the Minimal Dyck Transformer

- **Tokens.** Four symbols: 0, 1, *bos*, and *eos*, where *bos* and *eos* mark the beginning and end of every sequence.
- **Embeddings.** Each token is mapped to a vector in $\mathbb{R}^d$, where $d = 128$ is the *embedding dimension*. A learned vector depending only on the position i is added, so the model can tell positions apart.
- **Layers.** 1 encoder block and 1 decoder block, each with a single attention head, followed by a feedforward network: two linear maps $\mathbb{R}^{128} \to \mathbb{R}^{256} \to \mathbb{R}^{128}$ with a GELU in between.
- **GELU.** The activation $\mathrm{GELU}(x) = x \cdot \Phi(x)$, with Φ the standard normal CDF: a smooth version of $\max(0, x)$, applied coordinatewise.
- **Attention masks.** Output position k may only attend to output positions $1, \ldots, k-1$; cross-attention to the input is unrestricted.
- **Output.** A linear map $\mathbb{R}^{128} \to \mathbb{R}^4$ scores the 4 tokens at each step; the highest score is the prediction. In total: 339,716 trainable parameters.

return to talk