Séminaire Lotharingien de Combinatoire **XX** (2026)
Article #95, 12 pp.

Proceedings of the 38th Conference on Formal Power Series and Algebraic Combinatorics (Seattle)

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

Xiaoyu Huang*[1] and Blake Jackson†[2] and Kyu-Hwan Lee‡[2,3]

[1]*Department of Mathematics, Temple University, Philadelphia, PA 19122, U.S.A.*
[2]*Department of Mathematics, University of Connecticut, Storrs, CT 06269, U.S.A.*
[3]*Korea Institute for Advanced Study, Seoul 02455, Republic of Korea*

Abstract. There is a large class of problems in algebraic combinatorics which can be distilled into the same challenge: construct an explicit combinatorial bijection. Traditionally, researchers have solved challenges like these by visually inspecting the data for patterns, formulating conjectures, and then proving them. But what is to be done if patterns fail to emerge until the data grows beyond human scale? In this paper, we propose a new workflow for discovering combinatorial bijections via machine learning. As a proof of concept, we train a transformer on paired Dyck paths and use its learned attention patterns to derive a new algorithmic description of the zeta map, which we call the *scaffolding map*.

Keywords: q,t-Catalan numbers, zeta map, machine learning, transformer

1 Motivation

The q,t-Catalan numbers are a family of polynomials in the variables q and t which refine the classical Catalan numbers [14]. The origin of these polynomials dates back to 1996 and is rooted in symmetric function theory, specifically the theory of Macdonald polynomials and the study of modules over the space of diagonal harmonics [13]. More precisely, the q,t-Catalan number $C_n(q,t) \in \mathbb{Z}_{\geq 0}[q,t]$ is the bigraded Hilbert series of the alternating component of the space of diagonal harmonics in $2n$ variables. Here are the first three q,t-Catalan numbers:

n	$C_n(q,t)$
1	1
2	$q+t$
3	$q^3+q^2t+qt+qt^2+t^3$

*xiaoyu.huang@temple.edu.
†blake.jackson@uconn.edu
‡khlee@math.uconn.edu

Since $C_n(q,t)$ are a refinement of the Catalan numbers, the search began for a combinatorial interpretation for these polynomials: a pair of *statistics* on some collection of objects enumerated by the Catalan numbers that would realize the coefficients of these polynomials. Ultimately, two formulas (discovered simultaneously) emerged, leveraging statistics on Dyck paths.

Theorem 1 (Haglund 2003, Haiman 2000)**.** *There exist nonnegative statistics area, bounce, and dinv on Dyck paths such that*

$$\begin{aligned} C_n(q,t) &= \sum_{w \in Dyck(n)} q^{area(w)} t^{bounce(w)} \\ &= \sum_{w \in Dyck(n)} q^{dinv(w)} t^{area(w)}. \end{aligned}$$

In the same paper which produced the bounce statistic, Haglund [15] also described a bijection on Dyck paths (called the *zeta map* ζ) with the property that

$$(area(w), dinv(w)) = (bounce(\zeta(w)), area(\zeta(w))).$$

Moreover, there have been various combinatorial descriptions of the zeta map (and/or its inverse)[1], which come in two main flavors: maps on the area vector of a Dyck path [3, 15] and sweep maps [4, 23].

The other side of the motivation for this paper comes from machine learning (ML). Its power and versatility have made it a valuable instrument for mathematical exploration. In recent years, a growing body of work under the theme of "machine learning for mathematics" has investigated whether ML models can recognize and exploit hidden mathematical patterns. When trained on datasets generated from mathematical objects, ML algorithms have sometimes achieved remarkably high classification accuracies in distinguishing objects according to their invariants or other defining features. Representative examples include algebraic curves [18, 19, 20, 5] and number fields [2, 17]. More intriguingly, interpreting what the model learns has occasionally led to new insights into the underlying mathematics. The recently uncovered *murmuration* phenomenon in arithmetic provides a striking instance of such progress [10, 20]. This line of inquiry suggests a novel paradigm for mathematical research: one begins by constructing datasets, applies machine learning techniques, analyzes the learned representations, and, from these interpretations, formulates and ultimately proves conjectures.

Combining these two compelling aspects, our main contribution is a new combinatorial algorithm for the zeta map, derived from a trained *transformer* model.

[1]The precise direction of the "zeta map" (and therefore the direction of the "inverse zeta map") seems to differ across the literature. We prefer to call the map that sends (area, dinv) to (bounce, area) the "(forward) zeta map."

Theorem 2 (Huang, Jackson, Lee 2025+)**.** *The zeta map is equivalent to a scaffolding map.*

The scaffolding map is reminiscent of the sweep map implementations of the zeta map; the main difference is its use of the implicit symmetry in the peaks of a Dyck path. We emphasize that this is a novel description of the zeta map that has not appeared in the literature before.

The proof of our main theorem consists of showing that the inverse zeta map (more precisely, a sweep style description of the inverse zeta map) composed with our scaffolding map is the identity on the set of all Dyck paths.

A broader goal of this paper is to use the zeta map as a proving ground for machine learning (ML) in combinatorics research, namely to demonstrate the ability to extract *combinatorial bijections* and *algorithms* from trained models. While the precise details may differ from one application of ML in math to another, the concepts articulated in this paper are widely applicable within combinatorics research.

2 Background on Machine Learning

As mentioned above, we adopt a *transformer* architecture as our principal learning model and apply it to the datasets of Dyck paths obtained from the zeta map. The transformer model, first proposed in the landmark paper "Attention is All You Need" [24], has demonstrated extraordinary effectiveness in sequence-to-sequence learning tasks, particularly in natural language processing (NLP). It now forms the fundamental framework for large language models (LLMs). More recently, transformers have proven successful in addressing various mathematical problems, ranging from basic computational tasks such as arithmetic [8] and eigenvalue estimation for symmetric matrices [7], to more challenging or open problems such as constructing Lyapunov functions for dynamical stability [1], determining certain Euler factors of elliptic curves from the other ones [6], and even identifying optimal solutions to longstanding combinatorial problems [9]. Given that our setting naturally involves mapping input sequences to output sequences, the transformer is an especially suitable choice of model.

2.1 Model: Transformer Architecture

For a comprehensive account of the transformer architecture, we refer the reader to [12, 24]. Here we provide a brief overview of the essential components relevant to our implementation.

The first stage of the model involves converting numerical data into discrete symbols in a manner analogous to word tokenization in natural language processing. In our setting, the tokenizer constructs a vocabulary consisting of $\{0, 1, bos, eos\}$ that appear in the dataset, each treated as an individual token. The model thus operates on symbolic

sequences rather than directly on numerical values. Each token is then mapped to a d-dimensional vector through a learnable embedding matrix of size $v \times d$, where v denotes the vocabulary size. In our case, $v = 4$ and $d = 256$ or 128. This process, achieved by multiplying the token's one-hot representation by the embedding matrix, embeds tokens into a continuous vector space that the model can manipulate effectively.

Because the transformer architecture lacks an inherent sense of sequential order, positional information must be explicitly supplied. To this end, positional encodings are added to the token embeddings, providing the model with knowledge of each token's location within the sequence. These encodings may be fixed or trainable parameters optimized jointly with the model.

The transformer can follow an encoder–decoder architecture, originally designed for sequence-to-sequence tasks such as translation. The encoder receives an input sequence, embeds and enriches it with positional information, and processes it through a stack of n transformer layers. Each encoder layer consists of two main components: a multi-head self-attention mechanism and a position-wise feedforward network. The self-attention module enables every token to attend to all others in the same sequence, thereby capturing dependencies of various lengths and types. From each token embedding, the model derives three vectors—a query, a key, and a value—which collectively form the matrices Q, K, and V for the entire sequence. The attention mechanism compares each query with all keys to compute attention weights that quantify how strongly one token should focus on another. This allows information to flow adaptively across the sequence, updating each token's representation according to its learned relationships with others.

Rather than relying on a single attention map, the transformer employs multiple attention heads, each designed to capture distinct types of correlations within the data. Each head performs an independent attention computation, and the resulting outputs are concatenated and linearly projected to produce the final representation. Each head can thus be viewed as an independent channel through which the model exchanges and processes information. Following the attention modules, each sequence position is independently transformed by a one-hidden-layer feedforward network: a linear projection, a nonlinear activation, and a second linear projection.

The decoder also consists of a stack of transformer layers, but introduces an additional *cross-attention* mechanism between the encoder and decoder. Within each decoder layer, the first submodule performs masked self-attention so that each position can only attend to earlier positions in the output sequence, ensuring autoregressive generation. The second submodule, called cross-attention, allows the decoder to attend to the encoder's output representations. This step integrates information from the encoded input sequence, effectively aligning the target tokens with relevant parts of the input sequence. From an interpretive viewpoint, cross-attention provides an explicit mapping between input and output structures. For example, in our setting, given an input $w = (w_i)$, the importance of each token w_i $(1 \leq i \leq 2n)$ to each token $\zeta(w)_j$ $(1 \leq j \leq k-1)$ when pro-

ducing $\zeta(w)_k$ is represented explicitly by the cross-attention matrices of the transformer model. By examining these matrices, we can determine which input tokens w_i the model attends to most strongly when generating each output token.

In summary, the transformer encodes an input sequence by embedding its tokens, enriching them with positional information, and successively processing them through stacked attention and feedforward layers in the encoder. The decoder then generates an output sequence, guided both by its own past outputs (through self-attention) and by the encoder's contextualized representations (through cross-attention). The final decoder outputs are projected onto the vocabulary to produce a probability distribution over possible tokens at each position, from which the most likely sequence is chosen as the model's prediction.

3 Existing algorithms

There are (roughly) two existing combinatorial algorithms for the zeta or inverse zeta map: area sequence maps and sweep maps. The sweep maps have a similar flavor to our scaffolding map algorithm, with the difference being that our map implicitly relies on the symmetry surrounding the peaks of the Dyck path. The first type of zeta map is as an algorithm on the area sequence/bounce path [3, 15], which can be described by the following algorithm:

1. Compute the area sequence $a_1, a_2, \ldots, a_n$ of w (row lengths from bottom).
2. Set $b = \max\{a_i\} + 1$ to be one more than the length of the longest row.
3. Scan the area sequence from left to right, looking for 0's.
   (a) For each 0 encountered, write a 1.
4. Scan the area sequence from left to right, looking for 0's and 1's.
   (a) For each 0 encountered, write a 0.
   (b) For each 1 encountered, write a 1.
5. Repeat step 3 using the integers k and $k+1$ for $k = 1, 2, \ldots, b$, where encountering the smaller number appends a 0 to the output, and the larger appends a 1 to the output. The resulting binary sequence is $\zeta(w)$.

The second version of the zeta map (or inverse zeta map) is a sweep map composed with a reversal [4, 23], which can be described by the following algorithm:

1. Start with the binary sequence representation $w = (w_i)_{i=1,\ldots,2n}$ of length $2n$ for the Dyck path over the alphabet $\{0, 1\}$.

2. Reverse the word to obtain $rev(w) = (w_{n-i+1})_{i=1,\ldots,2n}$.

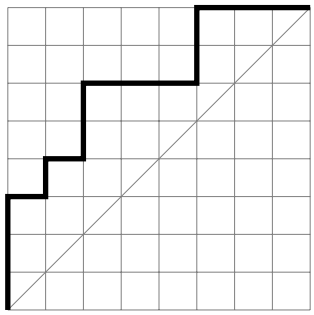
3. Assign levels to each position of $rev(w)$ by the following recursive process:

$$\ell_0 = 0, \qquad \ell_{i+1} = \begin{cases} \ell_i + 1 & \text{if } rev(w)_i = 1, \\ \ell_i - 1 & \text{if } rev(w)_i = 0. \end{cases}$$

4. "Sweep" the reversed word together with its level sequence from left to right repeatedly, recording the entry of the reversed word that corresponds to level $\ell = 0, -1, -2, \ldots, 4, 3, 2, 1$. The resulting binary sequence is $\zeta(w)$.

Below is an example of a sweep-style inverse zeta map acting on a Dyck vector of semilength 8.

Example 1. *Consider the Dyck path w below.*



To implement the inverse zeta map in the spirit of [4], there are four steps we need to take: write down the binary sequence for w, reverse it, calculate levels, and then sweep. To any binary sequence, a level vector ℓ_i can be calculated recursively:

$$\ell_0 = 0 \qquad \ell_{i+1} = \begin{cases} \ell_i + 1 & \textit{if } w_i = 1 \\ \ell_i - 1 & \textit{if } w_i = 0. \end{cases}$$

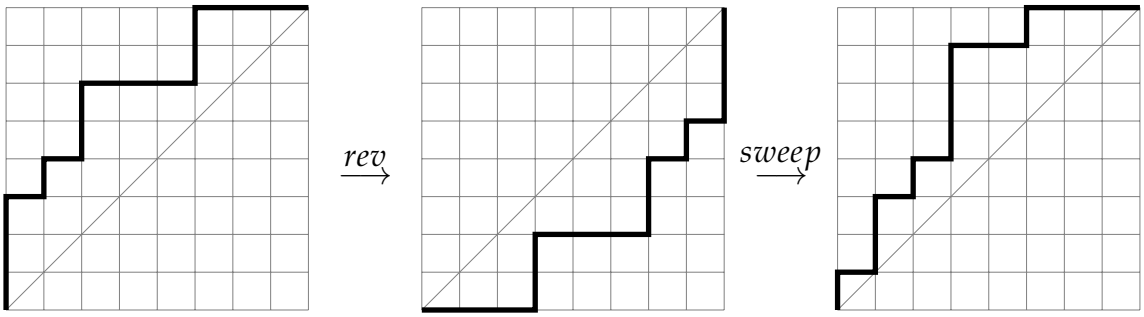
Following steps 1-3, we have

w	=	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*
rev(w)	=	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*
ℓ(rev(w))	=	*-1*	*-2*	*-3*	*-2*	*-1*	*-2*	*-3*	*-4*	*-3*	*-2*	*-3*	*-2*	*-3*	*-2*	*-1*	*0*

We then "sweep" the binary sequence and the corresponding level vector from left to right repeatedly, recording the entry of the sequence for levels $\ell = 0, -1, -2, -3, \ldots$. Doing this results in the image of the Dyck vector under the inverse zeta map:

$\zeta(w)$	=	*1*	*0*	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*0*
corresponding levels	=	*0*	*-1*	*-1*	*-1*	*-2*	*-2*	*-2*	*-2*	*-2*	*-2*	*-3*	*-3*	*-3*	*-3*	*-3*	*-4*

On the lattice path, $\zeta^{-1}(w)$ looks like a reversal followed by "sweeping" along the hyperplanes $y - x = k$ as k ranges from 0 to $-\infty$, collecting the steps crossed in level order:



4 Results

4.1 Experiment results

We began by experimenting with transformer models of varying complexities. Using a fixed architecture with an embedding dimension of 256, four encoder layers, and four decoder layers (each with eight attention heads), the model achieved 99% accuracy in learning the zeta map, that is, predicting the image of a Dyck path under the zeta map, for semi-lengths $n = 11, 12, 13, 14, 15, 16$[2].

Having established that a transformer architecture can successfully learn the zeta map, we next reduced the model's complexity and worked with one of the simplest transformer configurations: an embedding dimension of 128, a single encoder layer, a single decoder layer, and one attention head. The dataset consisted of all Dyck paths of semi-length $n = 13$, totaling 742,900 samples, and the resulting model contained 339,716 parameters. The hypothesis is that there is an underlying algorithm that the transformer model is implementing, and that one can uncover it when the model is sufficiently simple.

Again, this minimal transformer, which we name *Minimal Dyck Transformer*, consistently learned the zeta map with 100% accuracy across multiple random configurations.

4.2 Interpretation of *Minimal Dyck Transformer*

While there are various clues that the *Minimal Dyck Transformer* is implementing an interpretable algorithm, the most significant ones are the attention matrices throughout the generation of output tokens. An example is shown in Figure 1.

[2]The dataset sizes are 58 786, 208 012, 742 900, 2 674 440, 9 694 845, 35 357 670, corresponding to Catalan numbers $C(n)$.

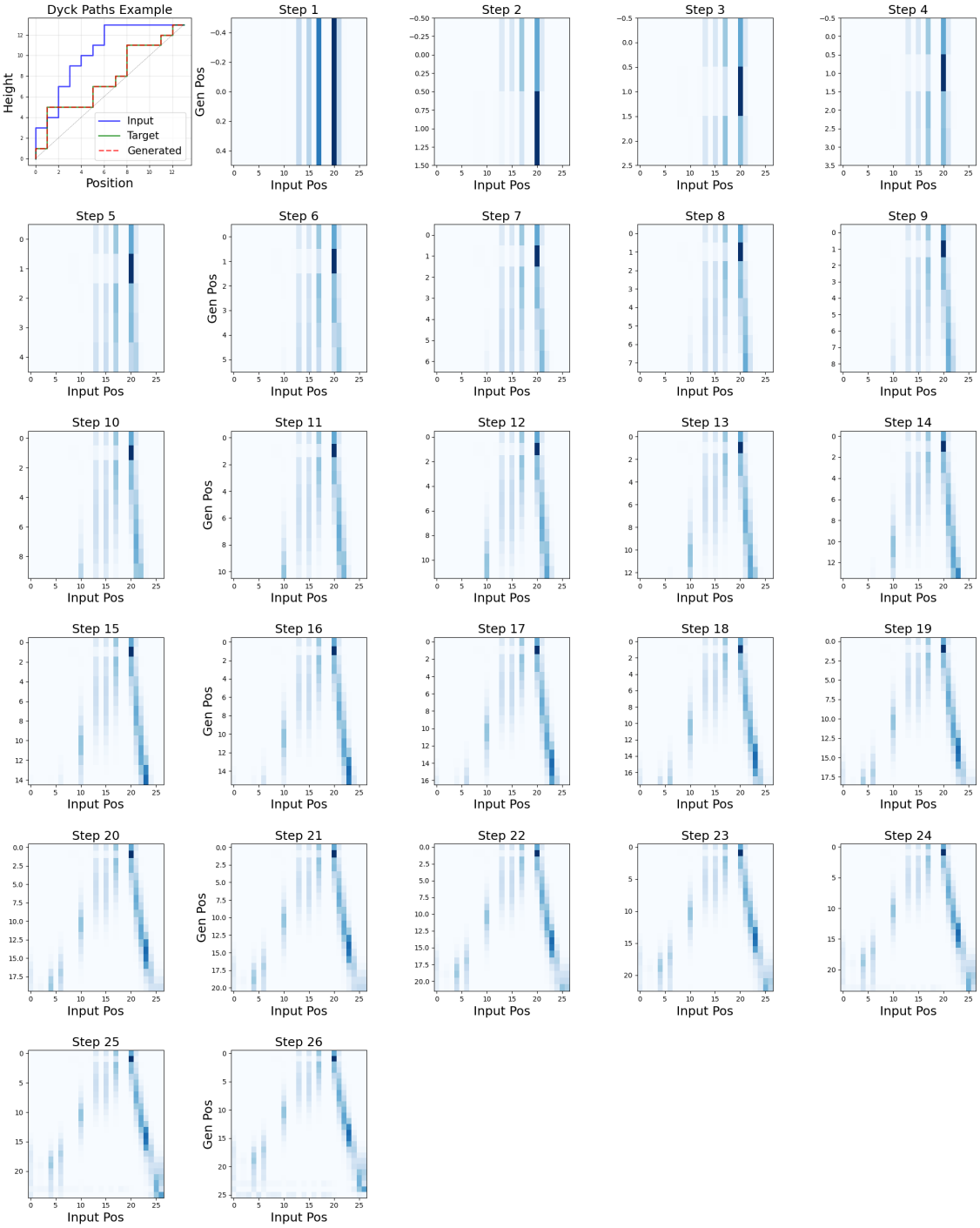


Figure 1: The visualizations show the cross-attention matrices across all generation steps for a single example, where the zeta map input is $1,1,1,0,1,0,1,1,1,0,1,1,0,1,0,1,0,1,1,0,0,0,0,0,0,0$. In the visualization for Step k, the entry at position $(i+1, j+1)$ represents the attention weight from $(\zeta(w))_i$ to w_j when generating $(\zeta(w))_k$. The $+1$ shift arises from the default *bos* prefix added to every input and output sequence. Darker colors correspond to stronger attention.

As mentioned in Section 2.1, we can obtain an explicit description of the importance of each w_i to each $\zeta(w)_j$ by examining cross-attention. As the transformer model generates the zeta map outputs step by step, one can dynamically track which input positions receive attention at each generation step.

Across all examples, three prevalent attention patterns emerge. First, most prominently observed in the Step 1 attention visualization, *Minimal Dyck Transformer* tends to focus on w_i with the highest level ℓ_i. Second, there are consistent column gaps in the attention matrices, indicating that the model ignores the i-th input position whenever $w_i = 1$ (i.e., a vertical step) throughout generation. Third, as more output steps are generated, a triangular shape appears in the attention visualizations for many examples, suggesting a gradual shift of focus toward consecutive input positions during generation.

To validate these observations, we conducted the following analyses on a trained *Minimal Dyck Transformer* with 100% accuracy[3].

1. Ablation studies: we manually removed all attention to w_i if $w_i = 1$ in the model. The resulting accuracy decreased by only 1.5%, confirming that the model indeed does not attend to such positions during generation.

2. Probing studies: we extracted encoder hidden states and verified that the *Minimal Dyck Transformer* implicitly computes the level sequence ℓ.

The model's tendency to attend to w_i with the highest level ℓ_i and to shift attention consecutively resembles the behavior of the sweep map description of the inverse zeta map. However, its consistent disregard for positions where $w_i = 1$ suggests that it is learning a different algorithm.

4.3 Scaffolding map

From the attention patterns and subsequent analysis, we derived the following procedure, which appears to be implemented implicitly by the transformer model; we refer to it as the *scaffolding map*.

1. Start with the binary sequence representation $w = (w_i)_{i=1,\dots,2n}$ of length $2n$ for the Dyck path over the alphabet $\{0, 1\}$.

2. Compute levels ℓ: start with $\ell_0 = 0$, and for each $i = 1, \dots, 2n$,

$$\ell_{i+1} = \begin{cases} \ell_i + 1, & \text{if } w_i = 1, \\ \ell_i - 1, & \text{if } w_i = 0. \end{cases}$$

[3]These analyses are considered standard methods in mechanistic interpretability [22].

3. Collect the right-step positions:

$$R = \{\, i \in \{1, \ldots, 2n\} : w_i = 0 \,\}.$$

4. Collect *peaks* by level: for every peak position i with $(w_i, w_{i+1}) = (1, 0)$, add i to the list corresponding to level ℓ_i.

5. Initialize:

$$\textit{out} = [\,], \qquad \textit{agents} = [\,], \qquad \textit{current level} = \max\{\ell_i : i \text{ is a peak}\}.$$

6. While $|\textit{out}| < 2n$:

   (a) Let *queue* be the union of
   - all positions i currently in *agents*, and
   - all peak positions j in *peaks* with $\ell_j = \textit{current level}$.

   (b) Sort *queue* in decreasing order and, for each $i \in \textit{queue}$ in this order, append w_i to *out*.

   (c) Update all agents simultaneously as follows. For each i in *agents*:
   - If $i \in R$, replace i by $i + 1$ if $i + 1 \in R$ and $i + 1 \leq 2n$; otherwise remove i from *agents*.
   - If $i \notin R$, replace i by $i - 1$ if $i - 1 \notin R$ and $i - 1 \geq 1$; otherwise remove i from *agents*.

   (d) For each peak j with $\ell_j = \textit{current level}$, attempt to append to *agents* the positions $j + 1$ if $j + 1 \in R$ and $j + 1 \leq 2n$, and $j - 1$ if $j - 1 \notin R$ and $j - 1 \geq 1$ (avoiding duplicates if desired).

   (e) Decrease *current level* by 1.

7. Return *out*.

We name it the scaffolding map because the 1-moves (the w_i's such that $w_i = 1$) act only to build up the levels ℓ_i, whereas the full set of levels and Dyck word entries can be reconstructed from the 0-steps, analogous to how a physical scaffolding defines the shape of a structure. The elements of *agents* are then imagined as positions of workers moving along this scaffolding, gathering the corresponding level and Dyck word data.

Although prior work has reverse-engineered the underlying algorithms that deep learning models implement for modular arithmetic and group operations [21, 11, 25], to the best of our knowledge, the scaffolding map is the first new sequential-output algorithm discovered through interpreting a neural network for a mathematical problem. Echoing the observations of [25], we also find a surprising diversity of solutions, suggesting that additional algorithmic variants remain to be uncovered.

References

[1] A. Alfarano, F. Charton, and A. Hayat. "Global Lyapunov functions: a long-standing open problem in mathematics, with symbolic transformers". *Advances in Neural Information Processing Systems* **37** (2024), pp. 93643–93670. DOI.

[2] M. Amir, Y.-H. He, K.-H. Lee, T. Oliver, and E. Sultanow. "Machine learning class numbers of real quadratic fields". *International Journal of Data Science in the Mathematical Sciences* **1**.02 (2023), pp. 107–134. DOI.

[3] G. E. Andrews, C. Krattenthaler, L. Orsina, and P. Papi. "ad-Nilpotent b-Ideals in sl(n) Having a Fixed Class of Nilpotence: Combinatorics and Enumeration". *Transactions of the American Mathematical Society* **354**.10 (2002). Publisher: American Mathematical Society, pp. 3835–3853. DOI.

[4] D. Armstrong, N. Loehr, and G. Warrington. "Sweep maps: A continuous family of sorting algorithms". *Advances in Mathematics* **284** (2015), pp. 159–185. DOI.

[5] A. Babei, B. Banwait, A. Fong, X. Huang, and D. Singh. "Machine learning approaches to the Shafarevich–Tate group of elliptic curves". 2026. DOI.

[6] A. Babei, F. Charton, E. Costa, X. Huang, K.-H. Lee, D. Lowry-Duda, A. Narayanan, and A. Pozdnyakov. "Learning Euler factors of elliptic curves". 2026. DOI.

[7] F. Charton. "Linear algebra with transformers". 2021. arXiv:2112.01898.

[8] F. Charton. "Learning the greatest common divisor: explaining transformer predictions". 2023. arXiv:2308.15594.

[9] F. Charton, J. Ellenberg, A. Wagner, and G. Williamson. "Patternboost: Constructions in mathematics with a little help from ai". 2024. arXiv:2411.00566.

[10] L. Chiou. "Elliptic curve 'murmurations' found with AI take flight". *Qunata Magazine* (2024). Link.

[11] B. Chughtai, L. Chan, and N. Nanda. "A toy model of universality: Reverse engineering how networks learn group operations". *International Conference on Machine Learning*. PMLR. 2023, pp. 6243–6267.

[12] N. Elhage et al. "A mathematical framework for transformer circuits". *Transformer Circuits Thread* **1**.1 (2021), p. 12.

[13] A. M. Garsia and M. Haiman. "A Remarkable q, t-Catalan Sequence and q-Lagrange Inversion". *Journal of Algebraic Combinatorics* **5**.3 (1996), pp. 191–244. DOI.

[14] J. Haglund. "The q, t-Catalan Numbers and the Space of Diagonal Harmonics".

[15] J. Haglund. "Conjectured statistics for the q,t-Catalan numbers". *Advances in Mathematics* **175**.2 (2003), pp. 319–334. DOI.

[16] M. Haiman. "The q, t-Catalan Numbers and the Alternating Component of the Diagonal Harmonics". Preprint, University of California, Berkeley. 2000.

[17] Y.-H. He, K.-H. Lee, and T. Oliver. "Machine-Learning Number Fields". *Mathematics, Computation and Geometry of Data* **2**.1 (2022), pp. 49–66. DOI.

[18] Y.-H. He, K.-H. Lee, and T. Oliver. "Machine-learning the Sato–Tate conjecture". *Journal of Symbolic Computation* **111** (2022), pp. 61–72. DOI.

[19] Y.-H. He, K.-H. Lee, and T. Oliver. "Machine learning invariants of arithmetic curves". *Journal of Symbolic Computation* **115** (2023), pp. 478–491. DOI.

[20] Y.-H. He, K.-H. Lee, T. Oliver, and A. Pozdnyakov. "Murmurations of elliptic curves". *Experimental Mathematics* **34**.3 (2025), pp. 528–540. DOI.

[21] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt. "Progress measures for grokking via mechanistic interpretability". 2023. arXiv:2301.05217.

[22] L. Sharkey et al. "Open problems in mechanistic interpretability". 2025. arXiv:2501.16496.

[23] H. Thomas and N. Williams. "Sweeping up zeta". *Selecta Mathematica* **24**.3 (2018), pp. 2003–2034. DOI.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". *Advances in neural information processing systems* **30** (2017).

[25] Z. Zhong, Z. Liu, M. Tegmark, and J. Andreas. "The clock and the pizza: Two stories in mechanistic explanation of neural networks". *Advances in neural information processing systems* **36** (2023), pp. 27223–27250. DOI.