Attention Is Only as Good as Its Witnesses:

A Tutorial

A companion to the paper by Mostafa 'Neo' Mohsenvand

MIT Media Lab

8 November 2026

This tutorial explains the paper "Attention Is Only as Good as Its Witnesses: When Higher-Order Layers Help, and When They Do Not" for a reader who knows roughly what a neural network, a loss, gradient descent, embeddings and a Transformer's attention are. Every other idea is taught where it is first needed, with examples from the paper's own tasks. Every result and number quoted is the paper's; worked examples with small made-up numbers are labelled as such.

Contents

0 The puzzle **3**

1 Prerequisites **4**

Attention as a soft lookup . . . 4
Softmax and temperature . . . 4
Graphs, walks and matrix powers . . . 4
The card game SET . . . 5

2 Witnesses: what a layer sums over **7**

Lookup, join and pair layer . . . 7
Families of witnesses . . . 7
The partition function . . . 8
Gates . . . 8
Read-out links and temperature . . . 8
The paper's Figure 1 . . . 8
The layer zoo, first rows . . . 8
Three principles . . . 10

3 Learning to search **11**

The benchmark . . . 11
Transformers do not learn this search as graphs grow . . . 11
Why a lookup must precompute reach sets . . . 11
A pair state learns . . . 11
Three more ways to sum over the middle vertex . . . 12
The depth law . . . 12
One layer for walks of every length . . . 14

4 Learning SET **19**

Transformers memorise before they learn . . . 19
No lookup can rank SET's pairs . . . 19

The join learns, and computes with clocks . . . 19
Counting with characters . . . 20
Newton's identities . . . 20
Newton attention as a layer . . . 21
Exponential Newton: the join's softmax at Newton's cost . . . 21

5 Temperature and size: the size law **25**
One voice against a room of whispers . . . 25
What the law says for each family . . . 25
Exponential gates against cancellation . . . 26
Read-outs divided by the number of candidates . . . 26
The law, measured . . . 26

6 What the principles predict elsewhere **29**
CLRS-30 in brief . . . 29
One small looped block against a large model . . . 29
Parallel and sequential: NC and P . . . 29
Errors sit where the theory puts them . . . 30
Near ties, and one exception . . . 31
Where joins add little: language and chess . . . 31

7 What it means **34**
The account in one paragraph . . . 34
The whole zoo . . . 34
What was known, and what is new . . . 34
Limits . . . 34
Open questions you could work on . . . 34

A Glossary **36**

B Answers to the questions **39**

C Reading list **40**

References **41**

0 The puzzle

A Transformer's attention lets every token look at every other token. Each look is a single comparison: the token that asks (the query) scores one other token (a key) at a time, and the layer takes a weighted average of what the high-scoring tokens carry. Researchers have also built layers that score *pairs* of tokens, or chains of tokens, and their record is strangely uneven.

In some places these higher-order layers are the heart of the system. AlphaFold, which predicts the shapes of proteins, keeps a state for every pair of amino acids and updates each pair through a third one[1]. The same pattern is the core of the Edge Transformer, a model of systematic compositional parsing[2]. Elsewhere it adds little: in language modelling, attention over pairs of tokens gives small gains in average loss[3,4], with larger ones on mathematics, coding, reasoning and logic[3].

Meanwhile, ordinary Transformers fail at tasks that look easy. Trained to find a path through a graph, they stop converging as the graphs grow[5], although theory shows that attention can represent connectivity with only logarithmically many layers[6]. Trained on the card game SET, a Transformer memorises its training deals long before it learns the rule, and learns the rule only after seeing many distinct questions.

So there are two puzzles. Why do higher-order layers help so much in some places and so little in others? And why does a Transformer fail to learn tasks that it can, in principle, represent?

The paper's answer is to look at *what a layer adds up*. Every attention-like layer computes, for each query, a weighted sum over some collection of objects: single tokens, pairs of tokens, routes through a graph, groups of tokens. The paper calls the objects that a task's answer depends on its **witnesses**, and the collection a layer sums over its **family**. When the layer's family matches the task's witnesses, the layer learns easily. When it does not, each token must first build the missing structure into its own state, and that is where training fails. The title borrows a courtroom image: a verdict can be no better than the witnesses who are called.

Here is where the tutorial goes. Chapter 2 writes every layer in one form, a weighted sum over a family read out at a "temperature" that sets how sharply the best candidates win. From that form come three principles and two theorems: a **depth law** for how far stacked pair layers can see along a graph (Chapter 3) and a **size law** for how sharp a read-out must be as the number of candidates grows (Chapter 5). New layers built on them learn where Transformers fail:

- one layer that adds up every route through a graph at once, trained on graphs of 49 vertices, answers every question at every search depth on graphs of 250 vertices, in each of three runs (Chapter 3);
- a layer that adds up all pairs of cards in time proportional to the number of cards, when told that SET's values wrap around after three, learns SET from as few examples as a pair-attention layer and, trained on tables of 8 cards, works on tables of 20 (Chapter 4);
- on twelve graph-algorithm tasks of a standard benchmark, one small block of 186,000 parameters, run repeatedly, matches (within half a percentage point) or beats a model of at least 73 million parameters on 9 of 12 tasks of the official test set, and on 8 of 12 with freshly drawn test graphs (Chapter 6).

The same account explains where pair layers did *not* help (language and chess, at the scales tested) and where a parallel layer must loop many times (Chapter 6). The positive results are on synthetic and algorithmic tasks, with small models; Chapter 7 states the limits.

How to read this tutorial

Indented notes like this one hold worked examples, analogies (each with a note on where it breaks) and asides. A note marked **Background** gives general knowledge that is not a claim of the paper. "Figure 3" means a figure of this tutorial; "paper Fig. 3" means a figure of the paper, and the paper's figures are reprinted here with their own panel letters. Chapters 2 to 6 end with questions, answered in Appendix B; Appendix A defines every term, and Appendix C is a reading list.

1 Prerequisites

This chapter teaches four things: attention as a soft lookup, softmax and temperature, graphs and why matrix powers count walks, and the card game SET.

Attention as a soft lookup

Think of a dictionary lookup. You hold a query word, the dictionary holds keys, and you read the value stored under the matching key. Attention does this softly. Token q (the query) has a query vector Q_q; every token j has a key vector K_j and a value vector V_j. The layer scores each key by a dot product $Q_q \cdot K_j$, turns the scores into positive weights that sum to one, and returns the weighted average of the values:

$$Y_q = \sum_j w_{qj}\, V_j, \qquad w_{qj} = \frac{\exp(\beta\, Q_q \cdot K_j)}{\sum_{j'} \exp(\beta\, Q_q \cdot K_{j'})}. \tag{1}$$

The fraction is the **softmax**. In the standard Transformer[7], $\beta = 1/\sqrt{r}$ for query and key vectors of length r.

One detail matters for everything that follows. Token j computes its key K_j from its own state *before* the query that will read it is known, like an index card filed before anyone asks a question. So if the answer depends on something about token j that involves other tokens, such as "which towns can be reached from town j", then j's state must already contain it when the question arrives. We will call this the **index-card problem**; it comes back in Chapters 3 and 4.

Softmax and temperature

The number β in Equation (1) is the **inverse temperature**; $1/\beta$ is the temperature. The name comes from physics, where a state of energy E has weight $\exp(-E/T)$ at temperature T, so low energy is likely. A softmax writes the same thing with a score $s = -E$ in place of minus the energy: high scores are likely, and $\beta = 1/T$. At high temperature (small β) every option gets nearly equal weight; at low temperature (large β) the highest score takes nearly all of it.

Worked example: the lookup of paper Fig. 1a

In paper Fig. 1a (reprinted as Figure 5), a town on a map looks up the other 48 towns. Its eight neighbours (its *children*, one road away) score 1, the 40 other towns score 0, and the hand-set layer uses $\beta = 3$. Each child gets weight $e^3/(8e^3 + 40) = 0.100$ and each other town $1/(8e^3 + 40) = 0.005$, as printed in the paper's figure; together the children hold 80%. At $\beta = 0$ every town gets $1/48$ and the children hold 17%; at $\beta = 1$, 35%; at $\beta = 6$, 99% (Figure 1a).

Now keep β fixed and let the crowd grow. With $\beta = 3$ and 40 other towns the children hold 80% of the weight; with 4,000 other towns they would hold under 4% (our arithmetic), because each of the many low scorers still gets a little weight (Figure 1b). To keep the true answers on top as the crowd grows, β must grow too. Chapter 5 turns this into a theorem with an exact rate, the logarithm of the size of the crowd.

Two more words. The scores before the softmax are called **logits**. Quantities measured in natural logarithms are in **nats**: a margin of 3 nats between two logits means one option is $e^3 \approx 20$ times more heavily weighted than the other at $\beta = 1$.

Graphs, walks and matrix powers

A **directed graph** is a set of vertices joined by one-way edges; the paper draws vertices as towns and edges as one-way roads. A **walk** is a sequence of roads, each starting where the last one ended, and may visit a town more than once. A **path** in the strict sense (a *simple* path) never revisits a town. A graph with no walk that returns to its start is **acyclic**; a directed acyclic graph is a **DAG**. Town v is **reachable** from town u if some walk leads from u to v; breadth-first search (BFS) finds every reachable town by exploring one road at a time.

The **adjacency matrix** A of a graph with n towns is the $n \times n$ table with $A_{uv} = 1$ if there is a road $u \to v$ and 0 otherwise. The entry (u, v) of the product $A^2 = AA$ is

$$(A^2)_{uv} = \sum_k A_{uk}\, A_{kv},$$

and each term is 1 exactly when there is a road $u \to k$ *and* a road $k \to v$. So $(A^2)_{uv}$ counts the walks of two roads from u to v, one for each middle town k, and $(A^\ell)_{uv}$ counts the walks of ℓ roads (Figure 2). The

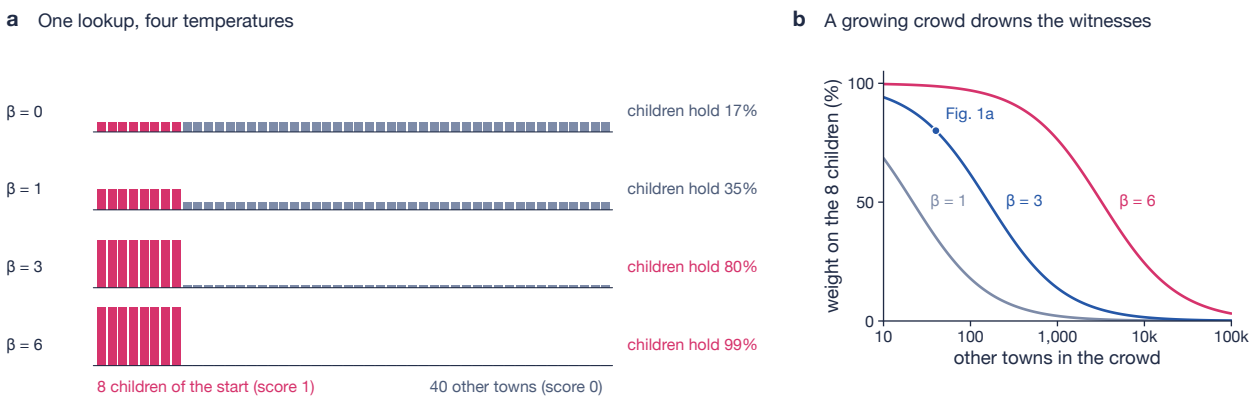


Figure 1 | Temperature decides how strongly the best candidates win. a, The lookup of paper Fig. 1a: eight children of the start score 1 (rose), 40 other towns score 0 (grey); each bar is one town's softmax weight. As β rises from 0 to 6, the weight moves from the crowd to the eight children. **b**, Keep the eight children and grow the crowd. Each curve is one β; the dot is paper Fig. 1a's case. At any fixed β the children's share collapses once the crowd is large enough; a larger β only delays the collapse. (Toy computation.)

identity matrix I counts the walk of zero roads from each town to itself, so $I + A + A^2 + \dots$ counts walks of every length; on a DAG the sum stops, because no walk is longer than the number of towns.

Two variations will matter. First, if you only care *whether* a walk exists, replace every positive count by 1. Write $[\cdot]$ for 1 if the statement inside holds and 0 otherwise: $R_{uv} = [(I+A)_{uv} > 0]$ says "v is within one road of u", and the thresholded product $[\sum_k R_{uk}R_{kv} \geq 1]$ says "within two roads". Second, **repeated squaring**: squaring the relation "within D roads" gives "within $2D$ roads", so L squarings reach 2^L roads, where adding one road at a time would take 2^L steps. This doubling is the heart of the depth law in Chapter 3.

The card game SET

SET is played with 81 cards. Each card has four attributes: a **number** of symbols (one, two or three), a **shape** (oval, diamond or squiggle), a **fill** (solid, striped or open) and a **colour** (three colours; this tutorial's own diagrams draw them rose, blue and violet). Every combination appears once, so there are $3^4 = 81$ cards. Three cards form a **set** when, for each attribute, the three cards are all the same or all different (Figure 3a).

There is a neat way to say this with arithmetic. Write each attribute's values as 0, 1 and 2 and add **modulo 3**, on a clock with three hours where 3 is back to 0. Three equal values add to $3x$, which is 0 modulo 3. Three different values are 0, 1 and 2, which add to 3, also 0. Two alike and one different add to $2x + y$ with $y \neq x$, never 0 modulo 3. So three cards form a set exactly when they add to zero in every attribute (Figure 3b). The cards with this addition are written $(\mathbb{Z}_3)^4$, "four copies of the integers modulo 3".

Two facts follow. Any two cards a and b have exactly one third card that completes a set with them, $-(a+b)$. And a card q in your hand splits the other 80 cards into 40 *partner pairs* $\{a, -q-a\}$, each of which completes a set with q.

The paper's SET task works like this. A **deal** is one card in hand and a table of 8 other cards, all distinct, and the question is whether two table cards complete a set *with the card in hand*. Half of the deals have such a pair. With 8 table cards there are $\binom{8}{2} = 28$ pairs to consider. Larger tests use tables of 16, 20 and up to 40 cards: 40 is the largest table with no pair that completes a set with the card in hand, since such a table holds at most one card of each partner pair.[1] Two kinds of generalisation are tested: to *new cards in hand*, never held during training, and to *larger tables*.

[1] **Background.** In the ordinary game, which asks for any set among the table cards, 20 cards can avoid every set. The paper's question is narrower: only sets that include the card in hand count.

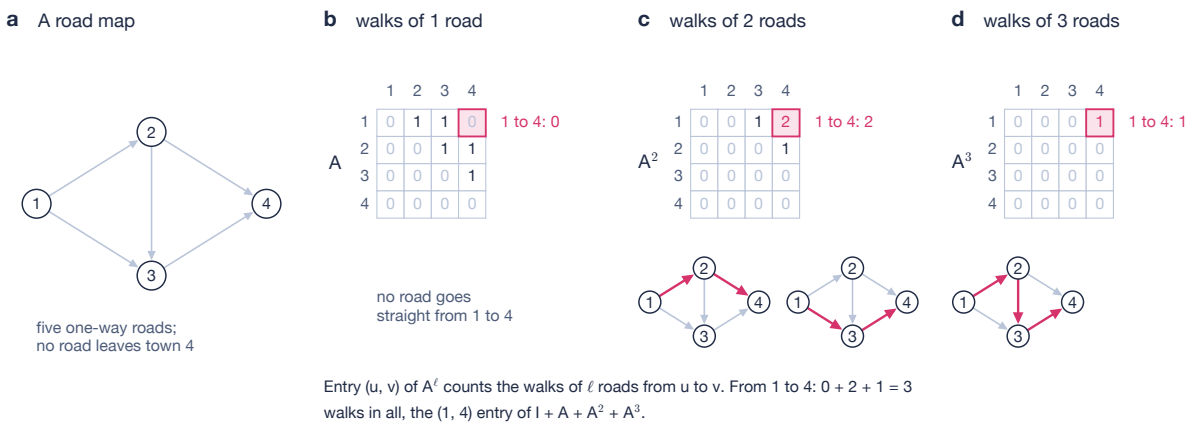


Figure 2 | **Matrix powers count walks.** **a**, Four towns and five one-way roads. **b**, The adjacency matrix A: row 1 has ones under 2 and 3, the towns one road from town 1; there is no road from 1 straight to 4. **c**, A^2: the entry for 1 to 4 is 2, and the two walks it counts are drawn below (through town 2 and through town 3). **d**, A^3: one walk of three roads, 1 to 2 to 3 to 4. The highlighted entries add up to all three walks from 1 to 4.

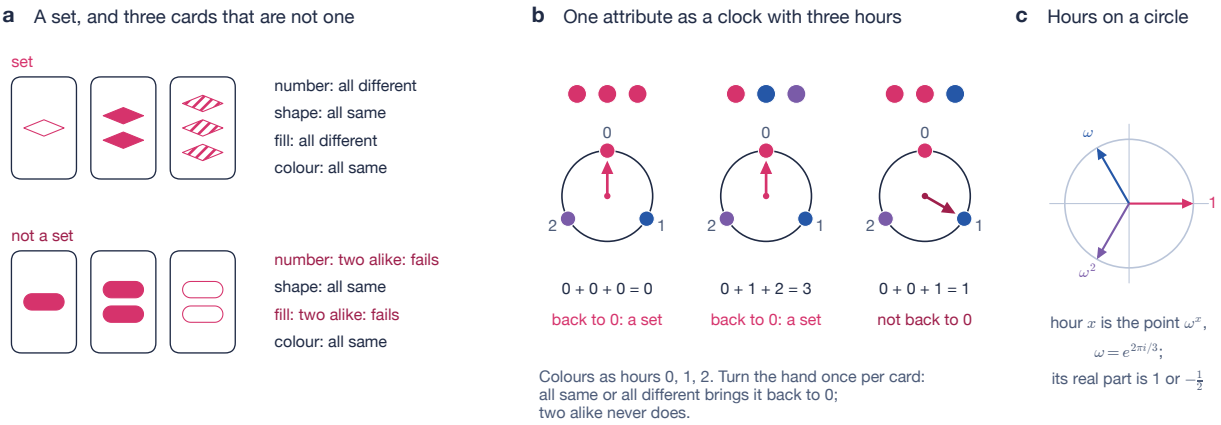


Figure 3 | **SET as arithmetic on clocks.** **a**, Top: one, two and three diamonds, all rose, with three different fills; every attribute is all the same or all different, so the three cards form a set. Bottom: the numbers are one, two and two and the fills solid, solid and open, so these three are not a set. **b**, One attribute as a clock with three hours. Each card turns the hand by its value; three equal or three different values bring it back to 0, two alike never do. **c**, The same hours as three points on a circle in the complex plane, 1, ω and ω^2 with $\omega = e^{2\pi i/3}$. Chapter 4 counts sets with this picture.

2 Witnesses: what a layer sums over

This chapter introduces the paper's central idea: every layer it studies adds up weights over a family of witnesses and reads the total at a temperature. We start with three layers, then define families, the partition function, gates and read-outs, and end with the first rows of the paper's table of layers.

Lookup, join and pair layer

A **lookup** is ordinary attention: the query scores each token on its own, $s_{qj} = Q_q \cdot K_j$, and averages. A **join** scores *combinations*. The simplest, **2-simplicial attention**[3,8], scores every ordered pair of tokens (j, k) with one number that depends on all three tokens, a *trilinear* score, and averages a value built from both partners (such as $V_j \odot V'_k$, their entrywise product) over all pairs:

$$s_{qjk} = \sum_i Q_{qi}\, K_{ji}\, K'_{ki}, \qquad Y_q = \sum_{j,k} \frac{\exp(\beta s_{qjk})}{\sum_{j',k'} \exp(\beta s_{qj'k'})}\, V_j \odot V'_k. \tag{2}$$

Compare a pair score that *splits* into two lookups, $f(q, j) + g(q, k)$. Its exponential factors, $\exp(\beta f)\exp(\beta g)$, so the sum over all pairs is a product of two sums over single tokens: nothing a lookup could not do. The trilinear score does not split this way, so the layer must list all n^2 pairs for each query. The number of tokens one score combines is the layer's **arity**: one for a lookup, two for this join.

A third kind of layer keeps a **pair state**: a vector z_{uv} for every ordered pair of tokens (u, v), read as "what the model believes about the relation from u to v". A **pair layer** updates z_{uv} from z_{uk} and z_{kv} for every middle token k, so its basic witness is a **two-step walk** $u \to k \to v$ (Figure 4). Chapter 3 writes its equation.

Look at SET. A lookup asks each table card, alone, how promising it is given the card in hand; by the index-card problem, that card would have to know in advance which other cards it pairs with. A join asks every pair at once, and a pair that completes a set can score high as a pair. In search, a lookup asks each town alone how relevant it is to the start; a pair layer can ask about each two-step route start $\to k \to$ goal.

Analogy: witnesses in a courtroom

A question is a case to be decided. A lookup calls each witness to the stand alone; a join calls witnesses in pairs, so that testimony that only makes sense together can be heard together; a walk is a chain of testimony, each witness vouching for the next. A *false* witness here is not a liar: it is a candidate that does not qualify, such as a pair of cards that does not complete a set. *Where it breaks:* a court calls a few chosen witnesses, while a layer hears every candidate in its family at once and weighs them all by a number, and the architecture, not the layer, decides the family.

Families of witnesses

A **witness** is a small object made of parts: a single token; a pair of tokens; a walk, whose parts are its roads; a subset of tokens; a **spanning tree** of a graph (a set of edges that connects all vertices without forming a cycle), whose parts are its edges. For each query q, a layer has a **family** F_q of candidate witnesses: single tokens for softmax attention, ordered pairs for the 2-simplicial join, two-step walks for a pair layer. Later chapters add walks of every length and subsets.

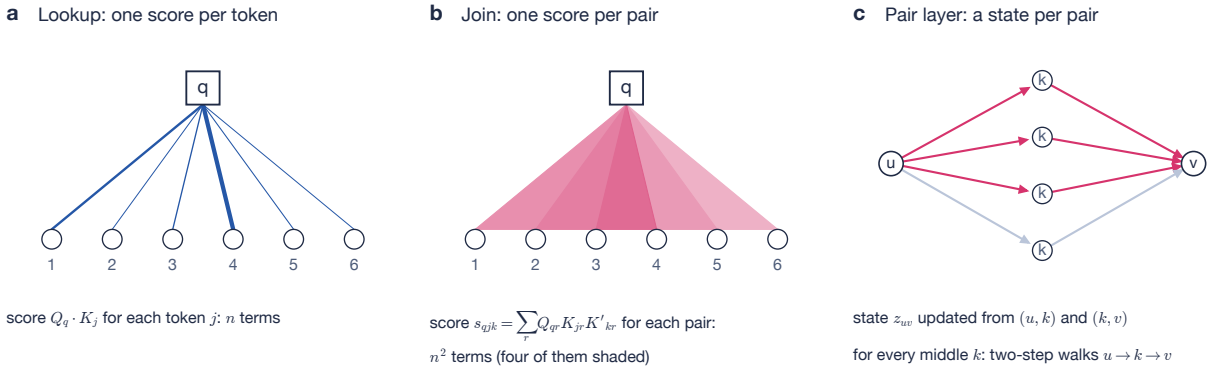


Figure 4 | What one score covers. a, A lookup: the query q scores each token on its own (n scores). **b**, A join: q scores each pair of tokens with one trilinear number (n^2 scores; four pairs shaded). **c**, A pair layer: the state of the pair (u, v) is updated from the pairs (u, k) and (k, v), for every middle token k: a sum over two-step walks.

The partition function

Every layer in the paper follows one recipe. Give each part τ of each witness a positive weight $\psi_q(\tau)$, its **potential**. Multiply the potentials of a witness's parts to get the witness's weight. Add the weights of all witnesses in the family: the total is the family's **partition function** Z_q, a name from statistical physics, where it is the sum of the weights of all states of a system. If each witness w also carries a value $v_q(w)$, add weight times value as well, giving Y_q:

$$Z_q = \sum_{w \in F_q} \prod_{\tau \in w} \psi_q(\tau), \qquad Y_q = \sum_{w \in F_q} \Big(\prod_{\tau \in w} \psi_q(\tau) \Big) \, v_q(w). \tag{3}$$

Because a weight is a product over parts, its logarithm is a sum: a witness's score is the sum of its parts' scores, and potentials are usually exponentials, $\psi = \exp(\beta s)$. Softmax attention is the simplest case: each witness is one token j with potential $\exp(\beta\, Q_q \cdot K_j)$ and value V_j, and its output, Equation (1), is Y_q/Z_q.

Worked example: two-step walks

Let the witnesses be the two-step walks $u \to k \to v$ and give each road potential 1 if it exists and 0 if not. A walk weighs 1 if both its roads exist and 0 otherwise, so $Z = \sum_k A_{uk} A_{kv} = (A^2)_{uv}$, the number of two-road walks. With soft potentials between 0 and 1, Z becomes a soft count.

Gates

A **gate** is a positive weight that multiplies what passes through a layer. Two kinds appear in this tutorial. A **sigmoid gate** $\sigma(x) = 1/(1+e^{-x})$ lies between 0 and 1, like the gates of an LSTM. An **exponential gate** $\exp(\beta(s-a))$ compares a score s with a level a: it is at least 1 when $s \geq a$ and shrinks by a factor e^{β} for every unit the score falls below the level. The two are related: when x is very negative, $\sigma(x) \approx e^{x}$, so in its low tail, where candidates that do not qualify sit, a sigmoid gate behaves like an exponential one.[2] The paper's third principle, below, is about exponential gates.

Read-out links and temperature

Having computed Z (and Y), a layer turns them into its output through one of four **links**:

$$Y_q/Z_q \;\text{ (average)}, \qquad \kappa_n\, Z_q \;\text{ (count, fixed scale } \kappa_n), \qquad 1 - e^{-Z_q} \;\text{ (OR)}, \qquad \beta^{-1} \log Z_q \;\text{ (free energy)}. \tag{4}$$

- The **average** is what softmax attention does.
- The **count** multiplies Z by a fixed number κ_n that may depend on the input size n (for example $1/\sqrt{n}$).
- The **OR** asks whether *at least one* witness is present: $1 - e^{-Z}$ is near 0 when Z is near 0 and above 0.95 once $Z \geq 3$. It is the "noisy-OR" of probability: if witnesses are present independently with small probabilities ρ_k, the chance that at least one is present is $1 - \prod_k (1-\rho_k)$, close to $1 - e^{-\sum_k \rho_k}$ [9].
- The **free energy** $\beta^{-1} \log Z$ is a soft maximum of the scores: with $Z = \sum_w \exp(\beta s_w)$ it tends to $\max_w s_w$ as β grows (**Background**; Mensch and Blondel[10] use this smoothing for dynamic programming). If the scores are minus costs, the same quantity with signs flipped is a soft *minimum* of the costs: for costs 3, 5 and 6 it is 2.83 at $\beta = 1$ and 3.00 at $\beta = 10$ (our arithmetic). This is how sums of exponentials become the minima of shortest-path algorithms.

The **temperature** $1/\beta$ is the scale of the log-potentials. As Chapter 1 showed, it decides whether the best witnesses dominate the total or drown in the crowd.

The paper's Figure 1

Paper Fig. 1 (Figure 5) shows the three kinds of sum on one search question and one SET deal, with hand-set layers and nothing trained.

The layer zoo, first rows

The paper collects its layers in one table. Here are the three rows we can read now; Chapters 3 and 4 add rows, and Chapter 7 prints the whole table. A **closed form** is a formula that computes Z without listing the family's members. **Cost** counts operations for all n queries of a sequence (or, for a pair state, all n^2 pairs).

[2] **Background:** $\sigma(x) = e^{x}/(1+e^{x})$, and $1 + e^{x} \approx 1$ when x is very negative.

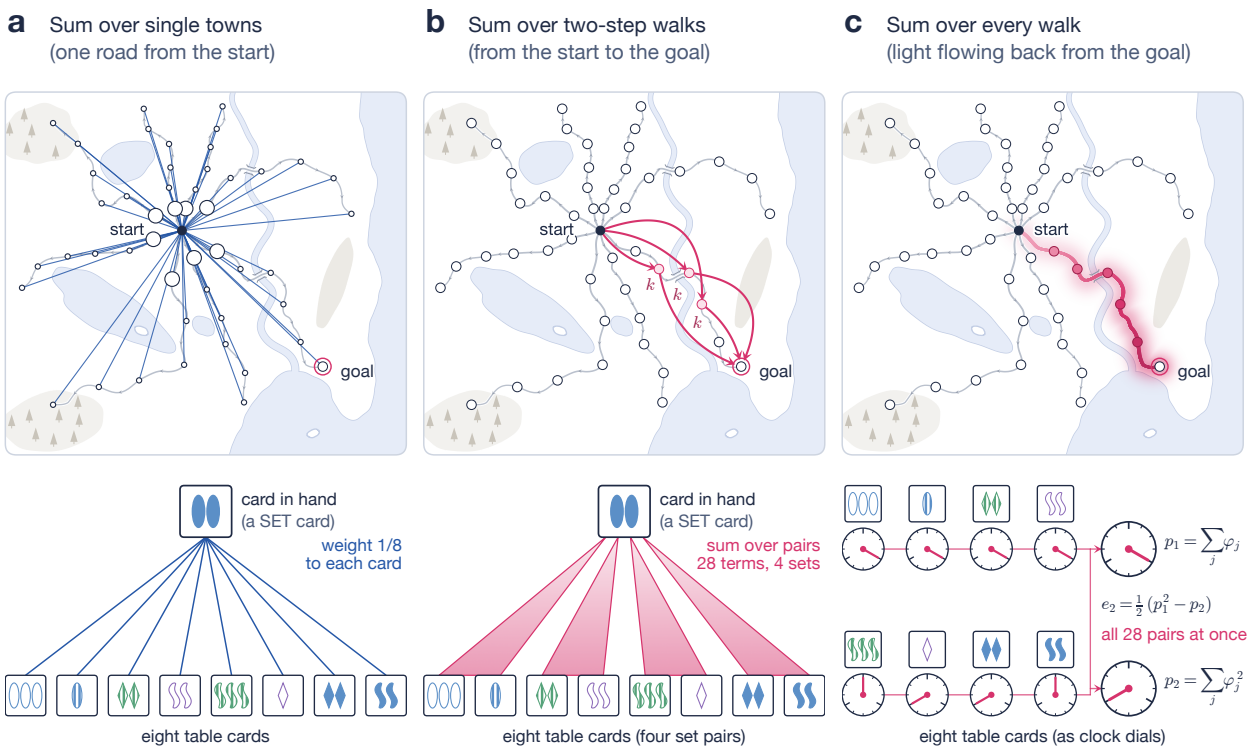


Figure 5 | Paper Fig. 1: what a layer sums over. Hand-set layers, nothing trained. Top: one search question on a map of 49 towns joined by one-way roads. Bottom: one SET deal with eight table cards (the paper's figure draws SET's colours as blue, green and violet). **a**, *A lookup sums over single witnesses.* The start weighs every town by its own score (circle area: 0.100 for each of its eight children, 0.005 for every other town); nothing in this sum knows where the goal is. The card in hand weighs each table card 1/8. **b**, *A join sums over pairs.* Top: two-step walks start $\to k \to$ goal in the composed relation "within four roads": each step is itself a route of up to four roads, and three middle towns k are within four roads of both ends. Bottom: the 28 pairs of table cards, the four set pairs shaded. **c**, *A closed form sums over every witness at once.* Top: the propagator (Chapter 3) sends light back from the goal, weight 0.8^{ℓ} at ℓ roads, so only the route that reaches the goal is lit. Bottom: Newton attention (Chapter 4) turns each card into a dial, its value under one character that combines all four attributes (Chapter 4), adds the dials (p_1) and their squares (p_2), and gets the sum of all 28 pair terms from $e_2 = (p_1^2 - p_2)/2$ without listing a pair (e_2 is a sum over pairs, defined in Chapter 4, not Euler's number e).

Table 1 | The layer zoo, first rows (from the paper's Table 1).

Layer	Family	Closed form of Z	Cost	Read-out
Softmax attention	single tokens	a sum	n^2	average
2-simplicial join	ordered pairs of tokens	none (listed one by one)	n^3	average
Pair layer, sum over k	two-step walks	matrix product	n^3	count, scale $1/\sqrt{n}$

Softmax attention sums n terms for each of n queries. The join's trilinear score must be listed pair by pair, n^2 pairs for each of n queries. The pair layer's sum over k, for all n^2 pairs at once, is a matrix product, n^3 operations. AlphaFold's triangle update[1] belongs to this family of two-step walks.

Three principles

From this view the paper draws three principles, two of them hypotheses and one a theorem.

1. **Match the family** (a hypothesis). In one step, a layer computes a read-out of its family's partition function and nothing else; any other structure the answer needs must already be in the partners' states. So, the paper hypothesises, a layer learns a task easily when the answer is a read-out of its family's partition function.
2. **Use a closed form** (a hypothesis). A product sums over subsets, a matrix inverse over walks, a determinant over spanning trees. A layer with such a closed form, the paper hypothesises, learns what stacked layers learn by composition, at less cost. Closed forms are parallel computations, so they also mark where a layer must instead loop (Chapter 6).
3. **Read the sum at low temperature through exponential gates, with a margin that grows with the logarithm of the family's size** (a theorem: the size law of Chapter 5).

The paper checks the size law in Lean, together with the construction half of the depth law. **Lean** is a programming language in which a proof is written so that a computer checks every step down to the axioms[11]; **Mathlib** is its library of formalised mathematics[12]. The experiments test the first two principles.

Check your understanding

1. What is the family of witnesses of softmax attention, and what is its read-out link?
2. In the two-step-walk example, why is Z equal to $(A^2)_{uv}$?
3. Why does a pair score $f(q, j)+g(q, k)$ give nothing a lookup could not, while the trilinear score of Equation (2) forces the join to list all n^2 pairs?
4. Which link asks "is there at least one witness?", and why does it saturate?
5. A sigmoid gate never exceeds 1. In what sense can it still act as an exponential gate?

3 Learning to search

Search is where the gap between lookups and pair layers is widest, and where the paper's new propagator layer does its most striking work. The chapter follows the paper's argument: Transformers fail, a pair state learns, the depth law says how far it sees, and one propagator layer sees all the way.

The benchmark

The paper uses the search benchmark of Saparov et al.[5], with its authors' generator, test sets and Transformer code. A **question** is a DAG written as a list of its edges in random order, followed by a start vertex, a goal vertex and the path taken so far; the **answer** is the next vertex on a path to the goal. Think of standing at a crossroads with a list of one-way roads in your hand.

A question's **lookahead** is the number of steps needed to tell the correct branch from the others. On the 250-vertex question of paper Fig. 4d, for example, each town on the answer route, taken as the current town, has as its lookahead its number of roads to the goal. The question of paper Fig. 1 has lookahead 6. Graphs of n vertices have lookaheads up to $\lfloor (n-1)/2 \rfloor$, which is 124 at 250 vertices. Training questions come from a *balanced* generator, the test sets hold equal numbers of questions at every lookahead, and the benchmark also has *random* graphs. A run **converges** when its training accuracy is at least 99.5% on two consecutive checks.[3]

Transformers do not learn this search as graphs grow

The benchmark's Transformer, run from its authors' code with every step of the path asked about, converged in none of ten runs on graphs of 31 to 100 vertices, each trained on at least 5.2 million examples.[4] Its authors trained it on 236 million examples per run on a variant that asks only about the first step: 5 of 14 seeds converged at 31 vertices, 2 at 41 and 1 at 49.

Do the paper's models win only because they read the graph better? To check, the paper trained a **fair control**: a Transformer given the same vertex representations as its own pair models, with the starting features of each pair of vertices added into its attention and read-out. It converged in all three runs at 49 vertices and in none of three at 125, within 10 million examples (Figure 11d).

Why a lookup must precompute reach sets

Stand at the current vertex c and consider a child v: the right move is to the child from which the goal can be reached. In one attention step c sees only v's key, filed before the question was asked: the index-card problem. So v's state must already say whether the goal is reachable from v, for whatever goal the question names, and the only way to have that ready is for each vertex to carry *the set of vertices it reaches*. That is what trained Transformers do: Saparov et al.[5] found that each vertex holds its reach set and successive layers merge these sets. In witness terms, the answer is a sum over walks, and a layer over single tokens can answer only if each token has already computed those walks. The sets grow with the graph, and fewer and fewer training runs converge.

A pair state learns

The paper's pair models first turn the tokens that mention a vertex into one vector for that vertex, its **name**, and start each pair's state z_{uv} (16 numbers) from learned features of the two names. Then each **summed pair layer** applies

$$z_{uv} \leftarrow z_{uv} + \sigma(\Theta_g t_{uv}) \odot \frac{1}{\sqrt{n}} \sum_k (\Theta_a t_{uk}) \odot (\Theta_b t_{kv}), \qquad z_{uv} \leftarrow z_{uv} + \mathrm{MLP}(t'_{uv}). \tag{5}$$

Here t_{uv} is the state rescaled to unit size over its 16 channels, t'_{uv} the same after the first update, Θ_a, Θ_b and Θ_g learned 16×16 matrices (Θ always marks learned weights), σ a sigmoid gate, $\odot$ entrywise multiplication, and the MLP a small network applied to each pair separately. The heart is the sum over the middle vertex k, something about (u, k) times something about (k, v): a matrix product over two-step walks.

[3] Training accuracy counts any valid next vertex; test accuracy counts the single next vertex on the shortest useful path, which is unique in every test question, and no test question is trained on.

[4] Three more runs were **registered** for 15 million examples, that is, written down with their predictions, budget and stopping rule before they ran, and were stopped for cost at 2.1 to 3.1 million, still at chance. Every design in the paper was registered this way; the paper lists each prediction with its outcome.

Eight such layers, with 39,000 to 49,000 parameters, converged in all twelve runs on graphs of 49 to 125 vertices, each within 1.3 million examples, and each answers at least 98.6% of test questions at every lookahead (Figure 11d).[5]

Three more ways to sum over the middle vertex

The paper compares the summed layer with two relatives that change only the sum over k (Figure 6).

- **Triangle attention**[2] replaces the equal weights $1/\sqrt{n}$ by a softmax over k of a score that couples the two legs:

$$\sum_k \mathrm{softmax}_k\big(\beta\,(\Theta_q t_{uk})\cdot(\Theta_k t_{kv})\big)\,(\Theta_a t_{uk})\odot(\Theta_b t_{kv}).$$

Because each score involves both (u,k) and (k,v), it must form and store all n^3 scores.

- The **OR join** gives each leg its own sigmoid gate and reads the sum through the OR link, one matrix product per channel:

$$O_{uv} = 1-\exp\Big(-\sum_k \sigma(\Theta_a t_{uk})\,\sigma(\Theta_b t_{kv})\Big), \qquad z_{uv} \leftarrow z_{uv} + \sigma(\Theta_g t_{uv})\odot\Theta_o O_{uv}.$$

A layer is **looped** when one block, with one set of weights, is applied K times in a row; the paper's looped models draw K from 5 to 8 during training and are tested after $\lceil \log_2 n \rceil + 2$ iterations.

The depth law

How far along a graph can L pair layers see? Repeated squaring suggests an answer, and the paper proves it in both directions.

The construction. Give the pair state a channel R that starts as "within one road" and a channel H that starts as "this vertex is the goal". Each layer squares R through the sum over k, so after layer i, R_{uv} says "v is within 2^i roads of u". Each layer also updates the goal channel: v is marked if it is within 2^{i-1} roads of a vertex marked one layer earlier. Since $2^i - 1 = 2^{i-1} + (2^{i-1} - 1)$, after layer i the goal channel marks exactly the vertices within $2^i - 1$ roads of the goal (Figure 7). A small MLP with thresholds keeps every channel a clean 0 or 1, and a last unit checks that v is a child of the current vertex c.

The lower bound. Could a cleverer network of the same kind see further? A **message-passing network** is a graph network in which each vertex repeatedly updates its state from its own state and the collection of its neighbours' states. The classical tool for its limits is the **Weisfeiler–Leman test**: give every vertex a colour, then repeatedly recolour each vertex by its own colour together with the collection of its neighbours' colours. Two vertices that still share a colour after L rounds cannot be told apart by any message-passing network of depth L that starts from the same colours.[6] The **folklore 2-dimensional** version colours *pairs* of vertices: the new colour of (u,v) is its old colour together with the collection, over all middle vertices k, of the colour pairs $(\mathrm{colour}(u,k), \mathrm{colour}(k,v))$. That is the shape of the pair update, so after L layers a pair's state can depend only on its colour after L rounds[13]. If two questions give the pair (c,v) the same colour, no L-layer network of this kind can answer them differently.

The paper builds two directed paths. In one the goal lies 2^L roads beyond v, so it is reachable; in the other it lies $2^L + 1$ roads behind c and v, upstream, so it is never reached (Figure 8). It proves that (c,v) gets the same colour in both, for every L.

Why, for every L: a window and a census. The proof pins down exactly what L rounds can know. After ℓ rounds of folklore 2-WL on a path, a pair (x,y) knows three things and nothing more: how far apart x and y are, and in which direction, if they are at most 2^ℓ roads apart; every label (the goal, the current vertex, an end of the path) within $2^\ell - 1$ roads of x or of y, a **window** of radius $2^\ell - 1$ around each end of the pair; and one count over the whole path, the **census**: how many vertices see each possible small window of labels around themselves. For the pair (c,v) after L rounds, the window runs from $2^L - 1$ roads before c to 2^L roads after it. A goal 2^L roads beyond v sits just past its right edge, and a goal $2^L + 1$ roads behind v just past its left edge. If the path is long enough and c is far enough from both ends, the two paths also have the same census, so the two questions look identical from (c,v). Move either goal one road closer and it enters the window, and the colours differ: the bound is exact.[7]

Here is the theorem, in the paper's terms with lightly simplified wording.

[5] Stopped at convergence they answer only 65 to 91% of the benchmark's random graphs, because balanced training rarely asks about later steps of a path or paths that merge; trained on to 3 million examples, as the propagator below is, they answer

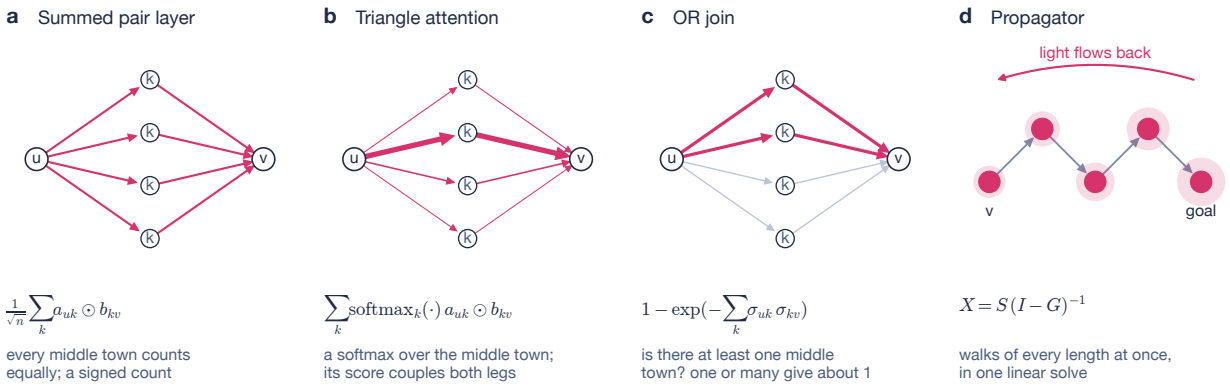


Figure 6 | Four ways a pair state sums over walks. **a**, The summed pair layer weighs every middle town equally and scales by $1/\sqrt{n}$; its factors are linear maps of the state and can be negative. **b**, Triangle attention: a softmax over the middle town, with a score that couples both legs (arrow width: weight). **c**, The OR join: separate sigmoid gates on the two legs, read through $1 - \exp(-\Sigma)$, so one middle town or many both give about 1. **d**, The propagator (below): walks of every length at once, as light flowing back from the goal along the roads.

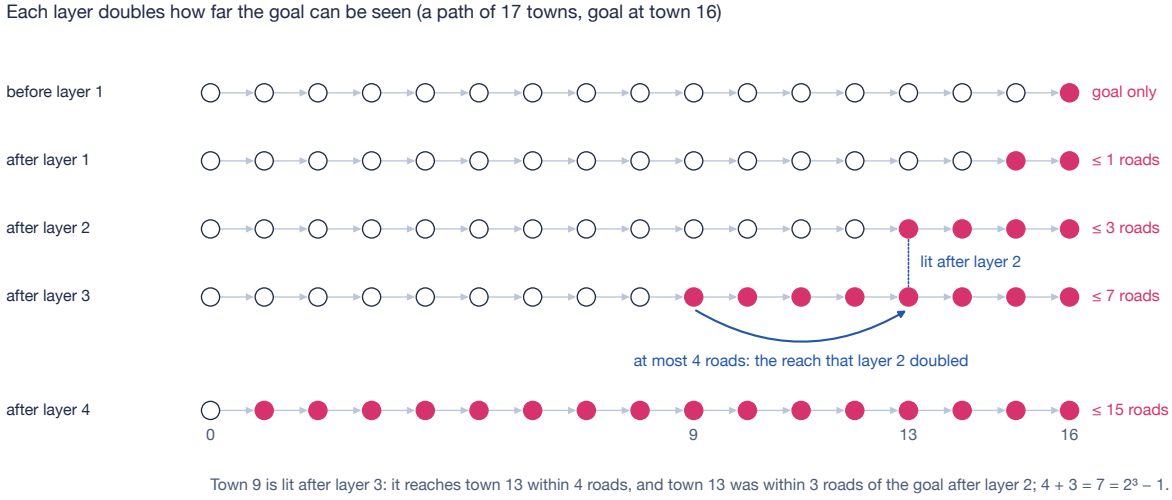


Figure 7 | Each layer doubles how far the goal can be seen. A path of 17 towns with the goal at town 16; a rose town is one the construction's goal channel marks. Before the first layer only the goal is lit; after layers 1 to 4, every town within 1, 3, 7 and 15 roads. The blue arrow shows why town 9 lights up at layer 3: it reaches town 13 within 4 roads (the reach that layer 2 doubled), and town 13 was already lit. Four layers see 15 roads from a child, so they decide lookahead 16 from the current town.

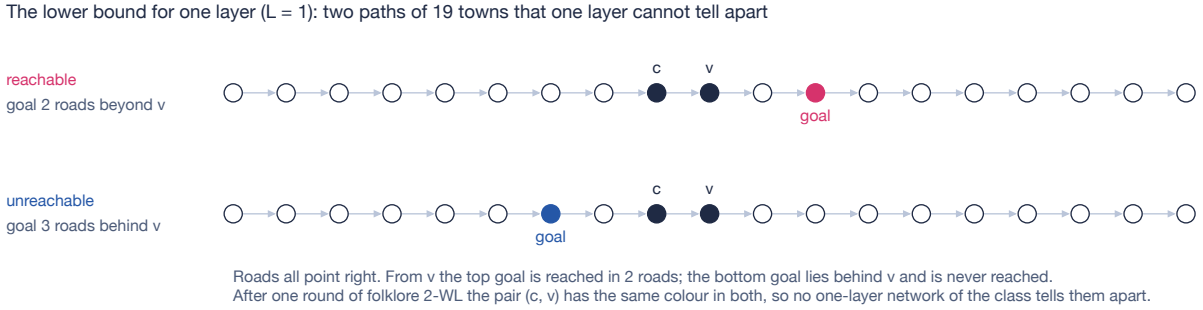


Figure 8 | The two paths behind the depth law's lower bound, for one layer. Both paths have 19 towns ($2^{L+2} + 11$ with $L = 1$), the current town c and its child v in the same place, and roads pointing right. Top: the goal is 2 roads beyond v, reachable. Bottom: the goal is 3 roads behind v, never reached. One round of folklore 2-WL gives the pair (c, v) the same colour in both, so no one-layer network of the class can tell "go to v" from "do not".

Theorem 1 (Depth law). *(i) For every number of vertices n and every $L \geq 1$ there are weights for L layers of Equation (5), applied to the state itself rather than its rescaled copy, whose read-out at (c, v) is positive exactly when v is a child of c from which the goal is reached in at most $2^L - 1$ steps. (ii) Let each of L layers update z_{uv} by any function of z_{uv} and the collection $\{(z_{uk}, z_{kv})\}_k$, and let every pair start from a state that depends only on its atomic type (whether $u = v$, which roads join u and v, and which of them is the goal or the current vertex). For every $L \geq 1$, no such network separates at (c, v) a goal 2^L steps beyond v from one $2^L + 1$ steps behind it, both goals on a directed path of $2^{L+2} + 11$ vertices.*[8] *So, from such anonymous starting states, the largest distance from v to the goal that L layers decide on every graph is exactly $2^L - 1$.*

LEAN: `PairSearch.pair_states_search_sharp`, `PairSearch.pair_states_search_looped`

Part (i) is repeated squaring, a classical idea[14], checked in Lean for the layer's own form.[9] Part (ii) is the window-and-census argument above, a written proof (not in Lean), and the paper's exact colour refinements for $L \leq 6$ agree with it. It does not cover starting states that identify vertices, such as the trained models' names: with unique names, two rounds of an update that keeps all its information already see the whole graph. The summed pair layer, triangle attention and the OR join all belong to the class of part (ii). The propagator, below, does not.

What trained models do. Lookahead counts from the current vertex, one more than from its child, so part (i) guarantees lookahead 2^L and part (ii) rules out $2^L + 1$ for anonymous states on paths. Trained pair states over names, with L from 2 to 8 layers, answer at least 95% at every lookahead up to 2^L in every run and fall soon after (Figure 9).[10]

Why do models that *do* have names still stop near 2^L? The paper proves no reason and calls the match empirical. Its Supplement reports what the trained states hold: a linear probe (a linear regression on the 16 channels) recovers "v reaches the goal within 32 steps", the construction's goal channel, with an AUC of 0.97 to 1.00 (1 is perfect, 0.5 chance) after the fifth iteration of a looped pair layer; and an eight-layer model's read-out, applied after each layer, answers at least 90% only up to lookahead 1 for five layers, then up to 5, 9 and 24 (the deepest tested) after layers 6, 7 and 8. The paper claims no more than that.

One layer for walks of every length

Eight pair layers spend their depth composing two-step walks. The second principle says: if the family has a closed form, use it. The family of walks of every length has one.

From echoes to matrices. Clap at one end of a long hall: you hear the clap, then an echo quieter by a factor g, then an echo of the echo, quieter by g^2, and so on. The total is the **geometric series**

$$1 + g + g^2 + g^3 + \cdots = \frac{1}{1 - g} \quad \text{when } |g| < 1,$$

5 when $g = 0.8$ (Figure 10a); if $g \geq 1$ the sum never settles. The same identity holds for matrices. Let G be an $n \times n$ matrix of gates, G_{uv} the fraction of weight that passes from u to v in one step. Then G^ℓ adds up all walks of ℓ steps, each weighted by the product of its gates, exactly as A^ℓ counted walks, and

$$I + G + G^2 + G^3 + \cdots = (I - G)^{-1}.$$

The matrix $(I - G)^{-1}$ is the **resolvent** of G. The series converges when repeated multiplication by G eventually shrinks every vector, that is, when the **spectral radius** of G (the largest size of its eigenvalues) is below one. Then one linear solve, about n^3 operations, replaces infinitely many walks.

The propagator. In each of four channels the paper's propagator forms gates G and a source S from the

96.6 to 97.5% at 49 vertices.

[6] **Background:** a network of this kind sees exactly what the colours see, round by round.

[7] The census is where the length of the path enters: on paths that are too short, or with c too close to an end, it can differ between the two questions.

[8] The paper's Proofs give the general condition: at least $2^{L+2} - 2$ vertices, with c at least $2^{L+1} - 2$ steps after the first vertex and at least $2^{L+1} - 1$ steps before the last.

[9] The construction uses five channels and residual ReLU MLPs of width 4; its first $L - 1$ layers are one repeated layer.

[10] Depths 2 to 8 at 49 vertices and 4 to 6 at 100. At 100 vertices four layers answer 28 to 36% at lookahead 17 and five layers 18 to 58% at 33, cut-offs registered in advance. A few models beat the worst case: the two-layer models answer lookahead 5, and two four-layer runs at 49 vertices that never converged answer 87 to 98% beyond lookahead 16.

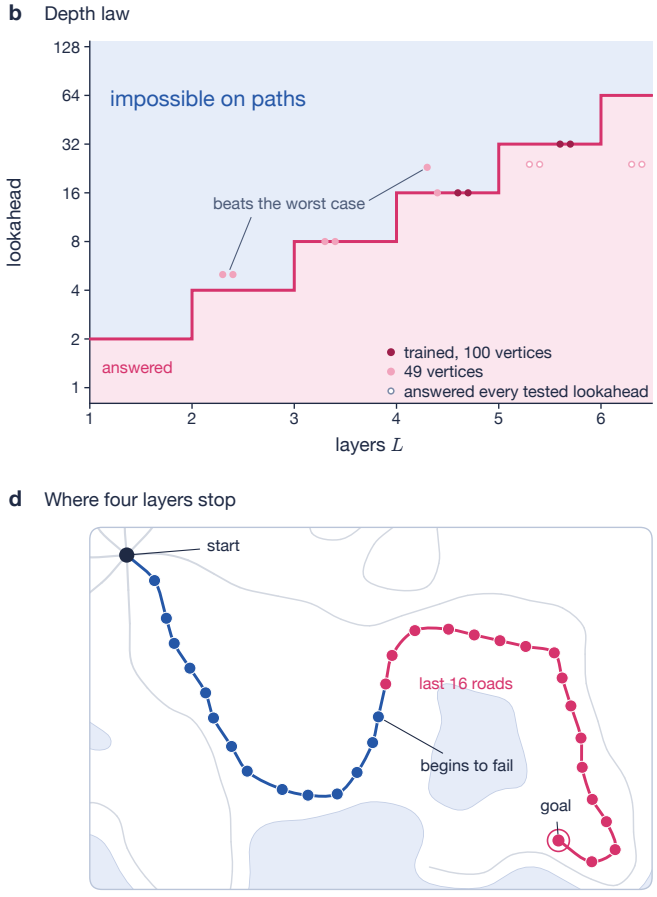


Figure 9 | Paper Fig. 4, panels b and d (the depth law; the whole figure is reprinted in Chapter 5). **b**, Layers L across, lookahead up (doubling at each grid line). Below the rose staircase, L layers answer every lookahead up to 2^L; from $2^L + 1$ (blue), no network of the class separates a reachable goal from an unreachable one on paths, from anonymous states. Dots: trained pair states over names at 49 vertices (light) and 100 (dark), at the deepest lookahead answered at 95% or more throughout (open: every tested lookahead). Most sit on the staircase; the three above it start from names on graphs that are not paths. **d**, The four-layer construction on a 250-vertex test question: each town on the answer route, taken as the current town, is answered when the goal is at most 16 roads away (rose), not beyond (blue).

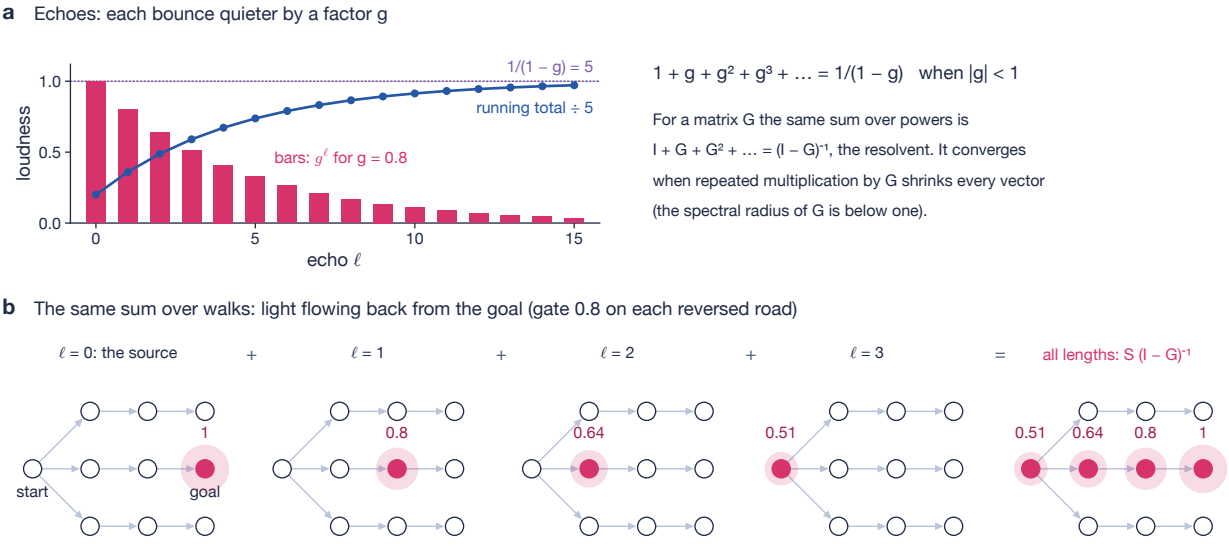


Figure 10 | The geometric series as echoes, and as walks. **a**, Echoes that each keep a fraction $g = 0.8$ of the last one's loudness (bars) add up to $1/(1-g) = 5$ (the blue running total, divided by 5, approaches the dotted line). **b**, The same sum over walks on a small map. The source puts weight 1 on the goal; each further power of G, with gate 0.8 on every road walked backwards, carries the light one more road back. Adding all powers gives $S(I-G)^{-1}$: the goal's route is lit with weights 1, 0.8, 0.64 and 0.51, and the start's other two children stay dark. (Toy example.)

rescaled pair state t, caps the gates, and solves for the walk sum:

$$\tilde{G}_{uv} = t_0\,\sigma(\theta \cdot t_{uv} + b), \qquad G_{uv} = \frac{t_0\,\tilde{G}_{uv}}{\max\left(\sum_{v'} \tilde{G}_{uv'},\ t_0\right)}, \qquad S_{uk} = \sigma(\theta' \cdot t_{uk} + b'),$$
$$X = S\,(I - G)^{-1} = \sum_{\ell \geq 0} S\,G^{\ell}, \tag{6}$$

with $t_0 = 0.99$ and θ, b, θ', b' learned. Every quantity is an $n \times n$ matrix indexed by pairs of vertices. Take one row: if the source is 1 at the pair (current vertex c, goal) and 0 elsewhere in row c, then $X_{cv} = \sum_{\ell}(G^{\ell})_{\text{goal},v}$, the total weight of all walks from the goal to v through the gates. The logarithm of X and $1 - e^{-X}$ then enter the pair state through a learned map and a gate, followed by the per-pair MLP. The first fraction in Equation (6) is the **row cap**: if the gates leaving a vertex add up to more than t_0, they are scaled down to add up to exactly t_0.

Why the cap works. If every row of a non-negative matrix sums to at most $t_0 < 1$, each step passes on at most a fraction t_0 of the weight, so all walks of ℓ steps from any vertex carry at most t_0^{ℓ}, and the series converges like $1 + t_0 + t_0^2 + \ldots$ on any graph, cycles included. The general statement is **Gershgorin's circle theorem**[15]: every eigenvalue lies in a disc around a diagonal entry whose radius is the sum of the sizes of the other entries in its row, so the spectral radius is at most the largest row sum of the entries' sizes.

How the layer searches. Set the gates by hand: open on every road *walked backwards* (a gate on (u, v) when there is a road $v \to u$), closed elsewhere, with the source on the goal. Weight starts at the goal and flows backwards along roads, reaching v exactly when a road path leads from v to the goal. So $X_{cv} > 0$ exactly when the goal can be reached from v, and a walk of ℓ roads keeps weight t_0^{ℓ} (paper Fig. 1c, Figure 10b).

> **Analogy: light flowing back from the goal**
>
> Pour light in at the goal and let it run backwards along every road, dimming at each step. Every town it reaches has a road path to the goal, and the brightness tells how short the path is; the resolvent adds it all up at once. *Where it breaks:* real light and real echoes can cancel; here weights only add, and one solve gives the total.

The propagator was the third registered design: the fixed t_0 and the row cap were added after two earlier designs without them lost accuracy at 250 vertices, the first (which learned t_0) in one of three runs and the second (every channel started near one) in two of three, where the spectral radius passed one. A ceiling well below one would cost accuracy on long routes, since a walk of ℓ steps keeps at most t_0^{ℓ} of its weight (Chapter 5).

Results. Trained on balanced graphs of up to 49 vertices, the propagator layer of 28,472 parameters converged in all three runs within about a million examples, each run training for 160 to 188 seconds on one H100 GPU. In every run it answers every test question at every lookahead at 75, 100, 125 and 250 vertices (at 250 vertices, 200 questions at each of 124 lookaheads), and 99.0 to 100% of the benchmark's random graphs of that size. Looped triangle attention, trained to the same sizes, answers at least 99% at every lookahead at 250 vertices in one run of three, and in four of fifteen runs across five training variants (Figure 11e).

What the trained layer learned. Each trained run recovers the hand-built layer plus one extra gate (Figure 11a–c). On a 250-vertex question, the answer channel's gates after the cap are 0.97 to 0.98 (median) on reversed roads and below 10^{-3} on forward roads and on pairs with no road, low enough: the hand-built capped layer still answers correctly with gates up to 4×10^{-3} on those pairs.[11] The source sits on the pair of the current vertex and the goal (0.999), so walks start at the goal. The extra gate, on the question's own start and goal (0.22 to 0.78 across runs), is closed in the construction.

Cost and precedents. One training step of the propagator (without the cap) processes 6.0 and 6.9 times as many examples per second as a looped OR join (in its simplest, one-term form) at 125 and 250 vertices, with 0.39 and 0.52 of its memory. Earlier models compute resolvents, fixed points or path sums over a *given* graph[16–18]; none that the paper found computes one from gates on a pair state, or applies one to search. Table 2 adds this chapter's four layers to the zoo.

Check your understanding

1. Why must a Transformer build reach sets in its token states, while a pair state need not?
2. By the construction, how many roads from a child v can three pair layers see, and what lookahead is that?

[11] Without the cap the limit would be $1/W = 10^{-3.6}$, where W is the total weight of all walks along the road gates; at that background the walk sum diverges.

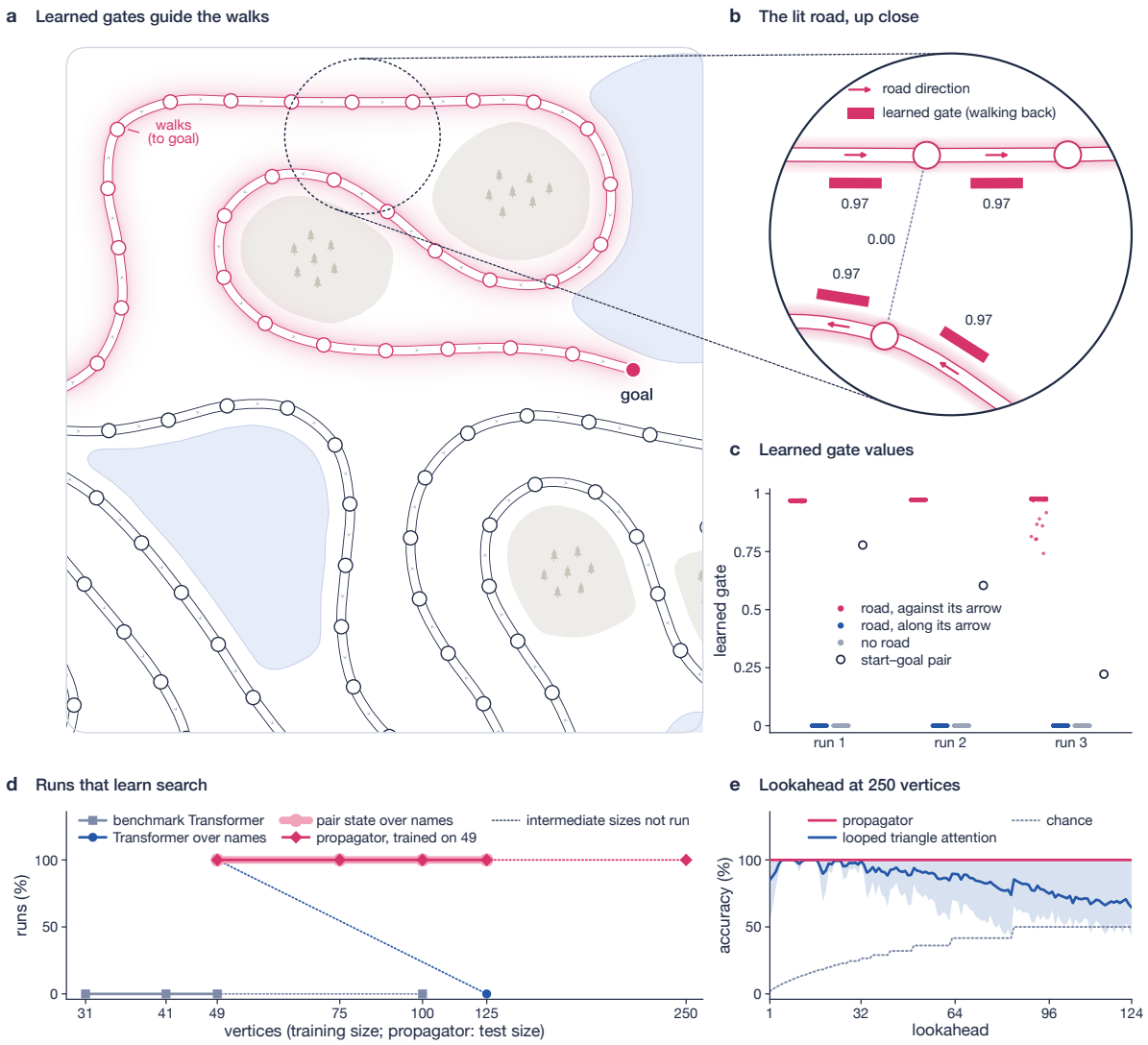


Figure 11 | Paper Fig. 2: one trained propagator layer searches graphs five times larger than any in training. a, Part of a 250-vertex test question with lookahead 31, lit by the walks to the goal of a layer trained on graphs of at most 49 vertices. Only the goal's branch is lit. **b**, The circled spot, enlarged: each road's arrow and its learned gate for walking *back* along it (rose bars, 0.97), and a dotted pair of towns with no road (gate 0.00). **c**, Every learned gate of the channel that carries the answer, in three runs: roads walked back (rose, near 1), the same roads walked forward (blue, near 0), sampled pairs with no road (grey, near 0) and the question's own start and goal (open, 0.22 to 0.78). **d**, Careful: two different criteria share this axis. For the other models, the share of runs that converge when *trained* at each size (three runs per point; one for the benchmark's Transformer at 100); for the propagator, trained only at 49 vertices, the share that answer at least 99% at every lookahead at each *test* size. **e**, Accuracy against lookahead at 250 vertices (200 questions each; mean and range of three runs): the propagator stays at 100%, looped triangle attention (at each run's best number of iterations) falls with depth; dotted, chance.

3. What does the Weisfeiler–Leman argument add, and why does it not apply directly to the trained models?
4. Why does the propagator need the spectral radius of G below one, and how does the row cap ensure it?
5. In a trained propagator, which gates are open and where is the source? Why does that answer the question?

Table 2 | Rows added in this chapter (from the paper's Table 1). Cost for all n^2 pairs.

Layer	Family	Closed form of Z	Cost	Read-out
Triangle attention	two-step walks	none (listed over k)	n^3, memory n^3	average
OR join	two-step walks	matrix product	n^3, memory n^2	OR
L pair layers, or one looped L times	walks of at most 2^L steps	repeated squaring	$L\,n^3$	OR, threshold
Propagator	walks of every length	$(I-G)^{-1}$	one solve, n^3	OR, free energy

4 Learning SET

SET is the paper's second test case. Its witnesses are pairs of table cards, so single tokens are the wrong family, and its rule is arithmetic modulo 3, so one can see exactly what each trained layer computes. This chapter builds three tools along the way: characters, Newton's identities and the Jacobi–Anger expansion.

Transformers memorise before they learn

A four-layer Transformer of 535,000 parameters trained on 32,768 deals answers 98% of its training deals and 54% of new ones, and first passes 90% on new deals at 524,288 deals (Figure 15a). Given enough distinct questions it does learn the rule: trained with 27 different cards in hand on 221,184 deals, a four-layer Transformer of 66,000 parameters answers 93% of deals whose card in hand it never held. With fewer cards in hand in training, every model the paper trained, the join included, is at chance on new cards in hand.

No lookup can rank SET's pairs

Suppose a layer scores each pair of table cards by adding two lookup scores, one per card: $f(q, a_j) + f(q, a_k)$, where f may be any function of the card in hand q and one table card. This is the index-card problem in its purest form: each card's score is fixed before its partner is known.

> **Proposition 1** (No single-card scores rank SET's pairs). *Let a layer score each pair of table cards $\{j, k\}$ by $f(q, a_j) + f(q, a_k)$, the sum of two lookup scores, for some function f of the card in hand q and one table card. For every q, every f and every table size $n \geq 4$ there is a deal on which a pair that does not complete a set with q scores at least as high as a pair that does.*

The proof is a small trap (Figure 12). Take any card a_j and its completion $a_k = -q - a_j$, then another card a_l and its completion $a_m = -q - a_l$. The pairs $\{j, k\}$ and $\{l, m\}$ complete sets; the cross pairs $\{k, l\}$ and $\{j, m\}$ do not, because each card has exactly one completion. If both set pairs scored above both cross pairs, the first comparison would give $f(a_j) > f(a_l)$ and the second $f(a_l) > f(a_j)$. Both cannot hold. Separations of this kind between pair-based and single-element set networks are known in general[19].

The join learns, and computes with clocks

A join's family is the pairs of table cards, so Proposition 1 does not bind it. One layer of 2-simplicial attention with 62,000 parameters passes 90% at 8,192 deals, 64 times fewer than the large Transformer needs, and trained on 55,296 deals it answers 99.6% of deals with new cards in hand.

What does it compute? In this tutorial a **clock** always means one thing: a rule that reads an hour (0, 1 or 2) and scores it by $\cos(2\pi \cdot \text{hour}/3)$, which is 1 at hour 0 and $-\frac{1}{2}$ at hours 1 and 2. (A trained clock may be turned by a fixed angle, a mix of that cosine and the matching sine; the counting box below shows that either way it is one conjugate pair of characters.) For SET, the natural hours are the four attributes' values of the sum $q + a_j + a_k$: hour 0 on an attribute means "all the same or all different" there.

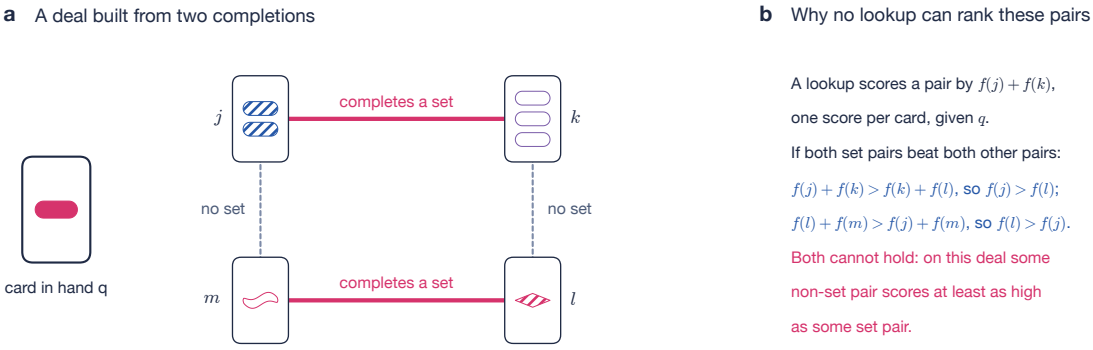


Figure 12 | Proposition 1's trap, on real cards. a, The card in hand q is one solid rose oval. Card j (two striped blue ovals) and k (three open violet ovals) complete a set with it: number, fill and colour all different, shape all the same. So do m (one open rose squiggle) and l (one striped rose diamond). The cross pairs $\{k, l\}$ and $\{j, m\}$ do not. **b**, A sum of one score per card would have to rank j above l and l above j at once.

The paper decomposes the trained join's score, over all triples of cards, into patterns, and finds that in joins trained with all 81 cards in hand, 82 to 94% of the score's variance lies on just four clocks, one per attribute (Figure 15b). "Variance lies on a pattern" means this. Any score over the three hours of one attribute splits into its average plus a varying part, and the varying part is a mix of a cosine and a sine of $2\pi \cdot \text{hour}/3$. Scores of 2, 0.5 and 0.5 at hours 0, 1 and 2, for example, are their average 1 plus $(1, -\frac{1}{2}, -\frac{1}{2})$, which is exactly the clock: all of their variation lies on it. The join's score for a pair is, to a good approximation, the *sum* of its four clocks.

Then the softmax turns the sum into a product. Writing $\gamma_1, \ldots, \gamma_4$ for the four clock values,

$$\exp\big(\beta(\gamma_1+\gamma_2+\gamma_3+\gamma_4)\big) = \exp(\beta\gamma_1)\,\exp(\beta\gamma_2)\,\exp(\beta\gamma_3)\,\exp(\beta\gamma_4).$$

Each factor is large only when its attribute is at hour 0, so the product is large only when *all four* are, that is, when the three cards form a set: the softmax turns four clocks into an AND. In the trained join of Figure 15b, the softmax weights over the 81 possible third cards peak on the set, and the next largest weight is 7×10^{-7}.

Counting with characters

To count sets exactly, we need a little algebra on the cards. A **group** is a set with an addition that has a zero and can be undone (every element has a negative); SET's cards with addition modulo 3 form one, and the order of addition does not matter, so it is an *abelian* group. Its **characters** are the group's version of the waves of a Fourier series (which writes a function as a sum of waves): functions that turn each card into a point on the unit circle and turn sums into products.

The counting box: characters of $(\mathbb{Z}_3)^4$

One hour. Let $\omega = e^{2\pi i/3}$, a third of the way round the unit circle, so $\omega^3 = 1$. The three points $1, \omega, \omega^2$ are equally spaced, so they balance: $1+\omega+\omega^2 = 0$ (algebraically, $\omega^3 - 1 = (\omega-1)(1+\omega+\omega^2) = 0$ and $\omega \neq 1$). Hence the average $\frac{1}{3}(1+\omega^z+\omega^{2z})$ is 1 if $z = 0$, and for $z = 1$ or 2 it is $\frac{1}{3}(1+\omega+\omega^2) = 0$ (for $z = 2$ use $\omega^4 = \omega$). The real part of ω and of ω^2 is $-\frac{1}{2}$.

Four attributes. Three cards form a set when $z = q + a_j + a_k$ is zero in all four attributes, so

$$[z=0] = \prod_{i=1}^{4} \tfrac{1}{3}\big(1+\omega^{z_i}+\omega^{2z_i}\big) = \frac{1}{81}\sum_{\mathbf{s}} \omega^{\mathbf{s}\cdot z}.$$

Multiplying out gives $3^4 = 81$ terms, one for each list $\mathbf{s} = (s_1, \ldots, s_4)$ with entries 0, 1 or 2. Each $\chi_{\mathbf{s}}(z) = \omega^{\mathbf{s}\cdot z}$ is a **character**, and it splits over the three cards: $\chi_{\mathbf{s}}(q + a_j + a_k) = \chi_{\mathbf{s}}(q)\,\chi_{\mathbf{s}}(a_j)\,\chi_{\mathbf{s}}(a_k)$.

Counting the 81. One is trivial ($\mathbf{s} = 0$, always 1). The other 80 come in 40 *conjugate pairs*, $\chi_{\mathbf{s}}$ and $\chi_{-\mathbf{s}}$, mirror images whose sum is twice the real part of either. A **clock is one conjugate pair on one attribute**: $\mathbf{s} = (1,0,0,0)$ with its mirror $(2,0,0,0)$ is the number clock, and $\chi + \bar{\chi} = 2\cos(2\pi z_1/3)$. Four attributes give 4 clocks, which are 8 characters; the remaining 36 pairs mix attributes, such as $(1,1,0,0)$, which turns with number and shape together.

Newton's identities

Counting with characters needs, for each character, a sum over all pairs of table cards of a product of per-card factors, and listing pairs costs $\binom{n}{2}$ terms. An identity that Isaac Newton found more than three centuries ago removes the listing. For numbers $\varphi_1, \ldots, \varphi_n$, the **elementary symmetric polynomial** e_m is the sum over all subsets of m of them of their product, and the **power sum** p_m is $\sum_j \varphi_j^m$. (Here e_m with a subscript is a polynomial, not Euler's number e.) **Newton's identities**[20] give the first from the second:

$$e_2 = \sum_{j<k} \varphi_j\varphi_k = \tfrac{1}{2}\big(p_1^2 - p_2\big), \qquad p_m = \sum_j \varphi_j^m, \qquad m\,e_m = \sum_{l=1}^{m} (-1)^{l-1} e_{m-l}\,p_l. \tag{7}$$

Worked example: Newton's identities with three numbers

Take $\varphi = (2,3,5)$: $p_1 = 10$, $p_2 = 38$, $p_3 = 160$. Then $e_2 = (10^2 - 38)/2 = 31$, and indeed $2\cdot3 + 2\cdot5 + 3\cdot5 = 31$. The recursion gives $3e_3 = e_2p_1 - e_1p_2 + e_0p_3 = 310 - 380 + 160 = 90$, so $e_3 = 30 = 2\cdot3\cdot5$. All appear at once in $(1+2\lambda)(1+3\lambda)(1+5\lambda) = 1 + 10\lambda + 31\lambda^2 + 30\lambda^3$, the coefficient of λ^m being e_m. Why e_2 works: p_1^2 adds $\varphi_j\varphi_k$ over all ordered pairs including $j = k$; subtracting p_2 removes $j = k$, and halving counts each unordered pair once.

Newton attention as a layer

Newton attention, one of the paper's three new layers, puts this to work (Figure 13a). It has 64 **channels**, copies of the same computation with their own learned weights. In each channel:

1. Each card j gets a complex feature φ_j, a dial: a sigmoid gate between 0 and 1 times a point on the unit circle at an angle learned from the card's vector.
2. For each card, the layer adds up its partners' features and their squares, $p_1 = \sum_j \varphi_j$ and $p_2 = \sum_j \varphi_j^2$ over the other cards, and forms $e_1 = p_1$ and $e_2 = \frac{1}{2}(p_1^2 - p_2)$.
3. The card in hand q, whose partners are the table cards, multiplies in its own feature: $\varphi_q\, e_2 = \sum_{j<k} \varphi_q \varphi_j \varphi_k$, one product per triple of card in hand and pair of table cards.
4. As trained, the features $\varphi_q\, e_m$ for $m = 0, 1, 2$ are divided by $\binom{n}{m}$, so that e_2 becomes an *average* over pairs, then by a root mean square over the 64 channels computed for each deal, and a learned linear map turns them into the answer.

If each dial is a character, step 3 gives $\chi(q)\chi(a_j)\chi(a_k) = \chi(q + a_j + a_k)$, and summing over characters counts sets:

> **Proposition 2** (Characters count set pairs at linear cost). *Let $q \in (\mathbb{Z}_3)^4$ be the card in hand and $a_1, \dots, a_n \in (\mathbb{Z}_3)^4$ the table cards. There are 40 characters $\chi_1, \dots, \chi_{40}$ of $(\mathbb{Z}_3)^4$, one from each conjugate pair of non-trivial characters, such that for every q and every table the number N of unordered pairs of table cards that complete a set with q is*
>
> $$81\,N = \binom{n}{2} + 2\,\mathrm{Re} \sum_{h=1}^{40} \chi_h(q)\, e_2\big(\chi_h(a_1), \dots, \chi_h(a_n)\big).$$

LEAN: `Newton.set_count`, `Newton.newton_identities`

The trivial character contributes $\binom{n}{2}$ and each conjugate pair twice the real part of one member, as in the counting box. Each e_2 costs two power sums, so the count costs time linear in n per channel. Measured on a GPU, Newton attention's time and memory per training step stay flat from 9 to 4,096 tokens per sequence, while the join runs out of memory at 1,024 (Figure 15d).

Worked example: one attribute

Shrink SET to one attribute: three characters and $3N = \binom{n}{2} + 2\,\mathrm{Re}\,[\chi(q)\, e_2]$ with $\chi(x) = \omega^x$. Let the card in hand have hour 0 and the table hours be $(1, 2, 2)$. The pair terms $\omega^{x_j + x_k}$ are $\omega^3 = 1$, $\omega^3 = 1$ and $\omega^4 = \omega$, so $e_2 = 2 + \omega$, whose real part is $2 - \frac{1}{2} = 1.5$. Then $3N = 3 + 3 = 6$ and $N = 2$: the two pairs that join hour 1 with an hour 2. A set pair always contributes $1 + 2 \times 1 = 3$ and any other pair $1 + 2 \times (-\frac{1}{2}) = 0$. Newton's identity gives the same e_2 from $p_1 = \omega + 2\omega^2$ and $p_2 = \omega^2 + 2\omega$, without listing a pair.

What Newton attention learned. Trained as the join is, a Newton layer of 55,714 parameters first passes 90% at 32,768 deals, four times as many as the join needs. In all 17 runs that reach 97%, its channels hold 38 to 40 of the 40 characters of Proposition 2. That explains the slower learning: the join's exponential turns four clocks into the AND over all 81 characters for free, while Newton's read-out is linear in each channel and must find the 40 characters one by one.

Why it failed on larger tables. Read out as trained, the layer answers only 73 to 76% on tables of 16 cards. The divisions of step 4 are the reason (Figure 14). Measure the read-out's decision threshold in *sets*: trained at 8 cards, it calls "set" once its output passes the value it gives a deal with 0.54 sets, between no set and one set, as it should. From 8 to 16 cards one set's term shrinks by $\binom{8}{2}/\binom{16}{2} = 28/120 = 0.23$ and the root mean square by 0.44, so one set's normalised share falls to about half, while the normalised baseline of the other pairs grows 2.3 times. A model of these two divisions, fitted on 8-card deals alone, predicts that the same threshold sits at 1.76 to 1.79 sets on 16 cards; the measured value is 1.76 to 1.77. A deal with exactly one set now falls below the threshold and is called negative.

The fix is to train at two sizes: trained on both 8- and 12-card tables, the same layer answers at least 95.5% at 16 cards and 92.8% at 20 (92.4% at 16 in a repeat with the same seeds). From two sizes it learns a threshold that depends on size. Read as a count, with no division, the layer is exact at every size, but that form did not train to the rule (85% on 8-card tables). Chapter 5 turns this into a general lesson.

Exponential Newton: the join's softmax at Newton's cost

Can one keep the join's sharp softmax, which gets the AND for free, and Newton's linear cost? The paper's third new layer, **exponential Newton**, does this exactly when the join's score is a sum of clocks on $\mathbb{Z}_3$ and

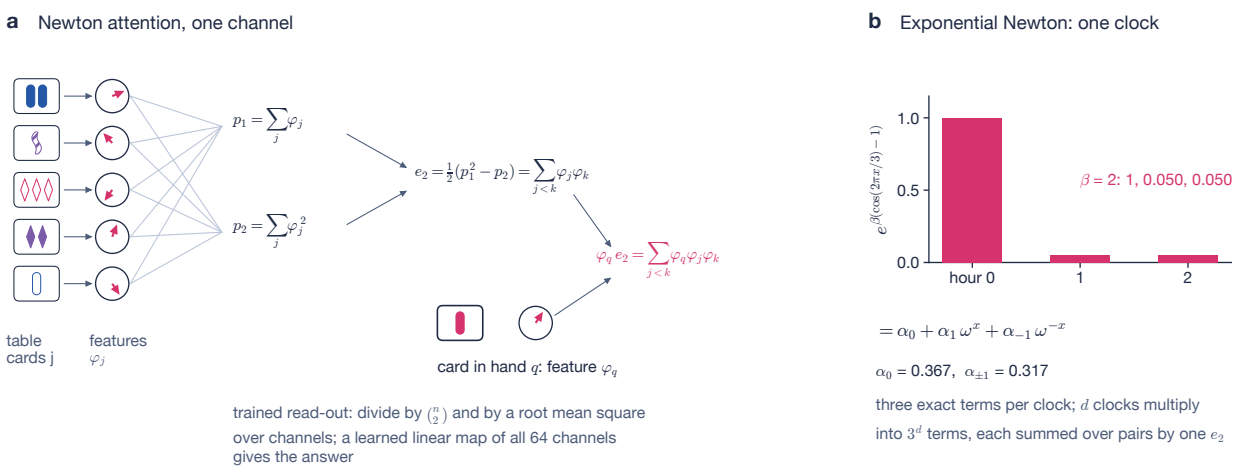


Figure 13 | Newton attention and exponential Newton, one channel. **a**, Each table card becomes a dial φ_j; the dials and their squares are added (p_1, p_2), Newton's identity gives e_2, the sum over all pairs, and the card in hand multiplies in its own dial, giving one term per triple. As trained, the result is divided by $\binom{n}{2}$ and a root mean square, and a linear map of all 64 channels gives the answer. **b**, Exponential Newton's clock: $\exp(\beta(\cos(2\pi x/3) - 1))$ takes only two values, 1 at hour 0 and $e^{-3\beta/2}$ (0.050 at $\beta = 2$) at hours 1 and 2, so it equals a three-term sum of characters with the coefficients shown.

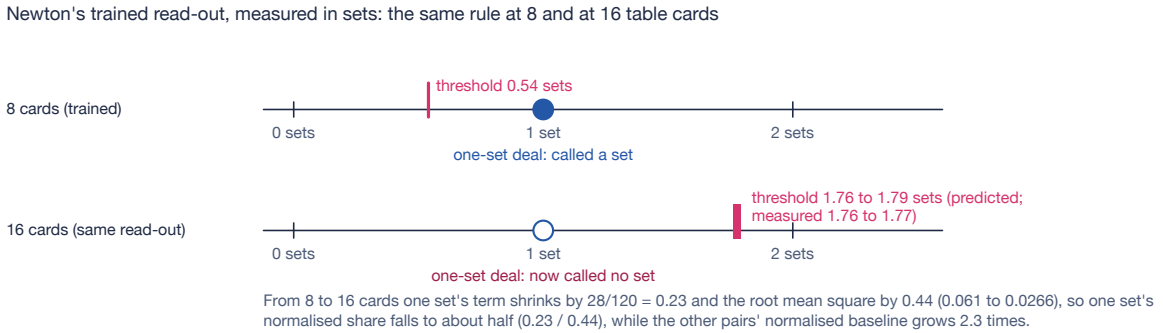


Figure 14 | Newton's 16-card failure on a number line. Deals are placed by how many set pairs they hold. At 8 cards the trained threshold, 0.54 sets, separates no set from one set. At 16 cards the same read-out, shrunk by the two divisions, needs 1.76 to 1.79 sets (predicted; measured 1.76 to 1.77), so a one-set deal (open circle) is called "no set".

the layer is *given the modulus 3*, told that hours wrap around after three, where the join learns its clocks from data.

The trick is that the exponential of a clock takes only two values, so it has an exact three-term Fourier series. With hour x and score $\beta(\cos(2\pi x/3) - 1)$, which is 0 at hour 0 and -1.5β at hours 1 and 2, write $\epsilon = \exp(-3\beta/2)$. The exponential is 1 at hour 0 and ϵ at hours 1 and 2, and

$$\exp\big(\beta(\cos(2\pi x/3) - 1)\big) = \alpha_0 + \alpha_1 \omega^x + \alpha_{-1} \omega^{-x}, \qquad \alpha_0 = \tfrac{1+2\epsilon}{3}, \quad \alpha_{\pm 1} = \tfrac{1-\epsilon}{3}.$$

Check: at $x = 0$ the right side is $(1 + 2\epsilon + 2 - 2\epsilon)/3 = 1$; at $x = 1$ it is $\frac{1+2\epsilon}{3} + \frac{1-\epsilon}{3}(\omega + \omega^2) = \frac{1+2\epsilon}{3} - \frac{1-\epsilon}{3} = \epsilon$, using $1 + \omega + \omega^2 = 0$. This is the **Jacobi–Anger expansion** of $\exp(\beta \cos\theta)$ as a Fourier series[21], folded onto three hours.

Worked example: three hours at two temperatures

At $\beta = 2$, $\epsilon = e^{-3} \approx 0.050$, so $\alpha_0 \approx 0.367$ and $\alpha_{\pm 1} \approx 0.317$ (our arithmetic). As β grows, $\epsilon \to 0$ and all three coefficients approach $\frac{1}{3}$: the expansion becomes $\frac{1}{3}(1 + \omega^x + \omega^{-x})$, the exact indicator of hour 0 from the counting box. At $\beta = 0$, $\alpha_0 = 1$ and $\alpha_{\pm 1} = 0$: every hour weighs the same. The temperature slides the layer between "no preference" and "exactly a set".

For d clocks, multiply d such expansions: each term of the product is a character times a fixed coefficient, so there are 3^d channels, and because a character splits over the three cards, each channel's sum over pairs of table cards is one Newton e_2.

Proposition 3 (Jacobi–Anger on $\mathbb{Z}_3$). *For every $\beta \geq 0$ and $x \in \mathbb{Z}_3^d$,*

$$\exp\Big(\beta \sum_{i=1}^{d} \big(\cos \tfrac{2\pi x_i}{3} - 1\big)\Big) = \sum_{\mathbf{s} \in \mathbb{Z}_3^d} \Big(\prod_{i=1}^{d} \alpha_{s_i}\Big)\, \omega^{\mathbf{s}\cdot x}$$

with α_0 and $\alpha_{\pm 1}$ as above. Hence, for a join whose score on the card in hand q and table cards j and k is $\beta \sum_i (\cos(2\pi x_i/3) - 1)$ with clock hours $x = x(q) + x(a_j) + x(a_k)$, the partition function and output of the softmax over the $\binom{n}{2}$ distinct pairs of table cards are sums over 3^d channels of Newton's e_2, computed in $O(3^d n)$ operations.

Four clocks give all 81 characters, with weights fixed by β (Figure 15c). The same expansion works for any finite abelian group, at a cost of the group's size times n, *provided the group is given.* In the trained layer each clock is a learned rule that reads an hour off each card (as a probability over the three hours), with one learned β.

Results. Trained as the join but with the modulus given, exponential Newton with four clocks first passes 90% at 8,192 deals, the join's budget, and answers 69% at 2,048 deals against the join's 51% (Figure 15a). But in 10 of 12 models the four learned clocks read one attribute twice and another not at all. Their errors on larger tables fall on deals holding a **pseudo-set**, a pair that completes a set on the three clocked attributes but not the fourth, and so scores exactly as high as a true set: a zero margin, which no temperature fixes (Chapter 5). With six clocks the layer has $3^6 = 729$ channels (not to be confused with the 729 *cards* of Chapter 5's six-attribute SET), and 5 of 6 models covered all four attributes. The six-clock layer passes 99% at 8,192 deals (99.9% and 100.0% in two repeats of the job with the same seeds). Trained on 8-card tables, it answers at least 97% of deals with new cards in hand on 16- and 20-card tables, for each of three seeds in both repeats, at its trained temperature (Figure 15e).[12]

Two limits belong here. At SET's table sizes $3^d n$ is larger than the $\binom{n}{2}$ pair terms it replaces (with 8 cards there are only 28 pairs, against 81×8 for four clocks), so the saving is for long sequences, where the paper has not timed it. And the layer is given the group, which the join learns.

Table 3 | Rows added in this chapter (from the paper's Table 1). Cost for all n queries; Newton's power sums are computed once and shared by every query.

Layer	Family	Closed form of Z	Cost	Read-out
Newton attention	m-subsets	$[\lambda^m] \prod_j (1 + \lambda\varphi_j)$	n per channel	count; trained as a mean
Exponential Newton	unordered pairs, clock scores	3^d characters	$3^d\, n$	average

Here $[\lambda^m] \prod_j (1 + \lambda\varphi_j)$ means "the coefficient of λ^m in the multiplied-out product", the generating function of the worked example above.

Check your understanding

1. In Figure 12, write out the two inequalities that a sum of single-card scores would need, and show that they contradict each other.
2. Use Newton's identities to compute e_2 of $(1, 4, 6)$ from its power sums, and check by listing the pairs.
3. How many characters does $(\mathbb{Z}_3)^4$ have, how many conjugate pairs, and how many characters make up the four clocks? Why can the join get by with four clocks while Newton attention needs 40 characters?
4. Why did Newton's mean read-out fail on 16-card tables, and why does training on two table sizes fix it?
5. A trained exponential Newton layer clocks number twice and never clocks colour. Why can no temperature rescue it on large tables?

[12] This is with the learning rate chosen on held-out cards in hand. With the rate picked on the training cards in hand, the registered test failed: one seed scored 52.3%.

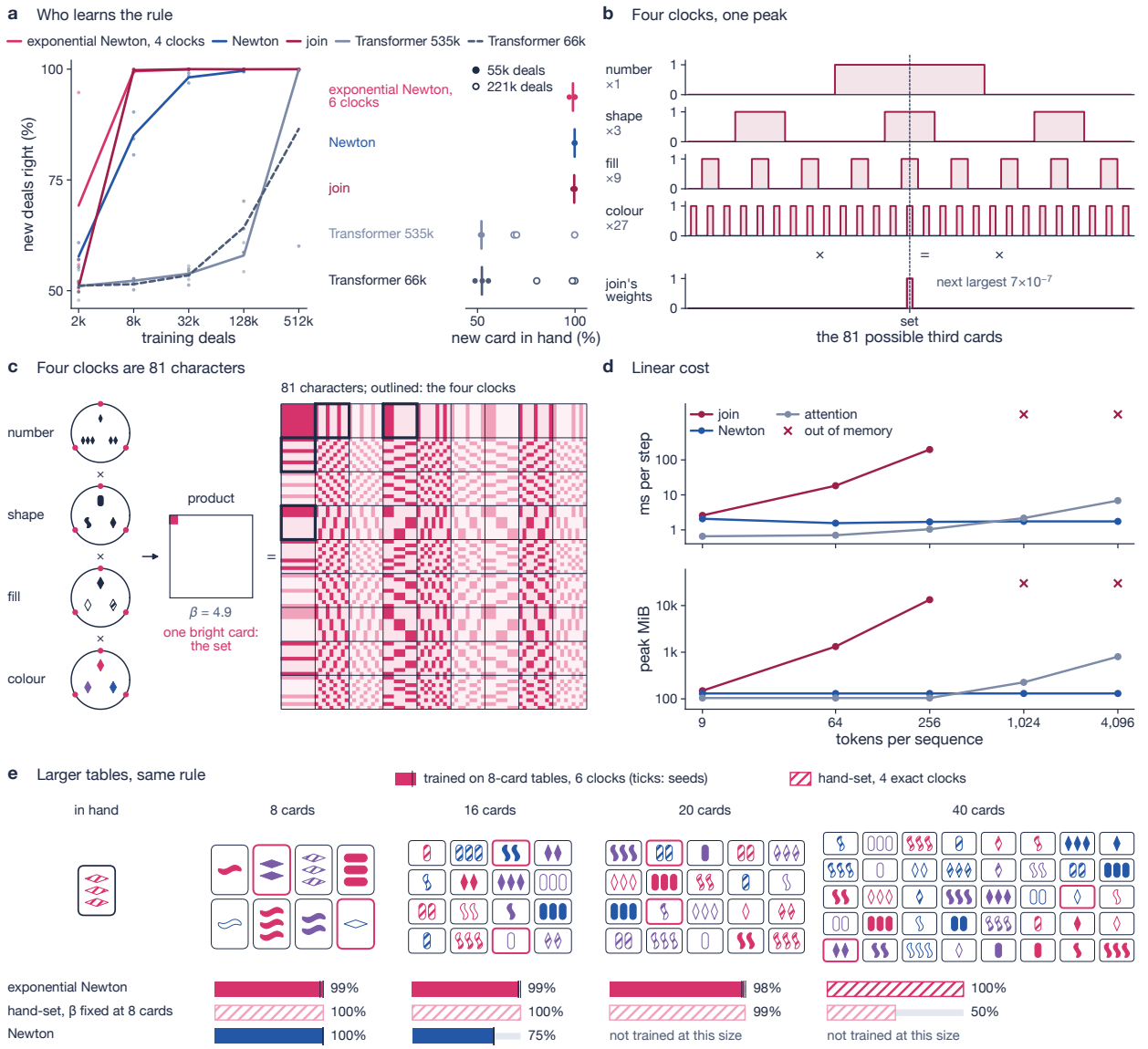


Figure 15 | Paper Fig. 3: who learns SET's rule, how the join decides, and why exponential gates hold as tables grow. a, Left: accuracy on new deals against training deals (mean of three seeds; exponential Newton with four clocks); the join and exponential Newton rise first, Newton follows, both Transformers stay near chance until the two largest budgets. Right: accuracy with a card in hand never held, after 55,296 deals with 27 cards in hand (filled; six-clock exponential Newton) and, for the Transformers, also after 221,184 deals (open). **b**, One trained join, over the 81 possible third cards (the set in the middle). The top four rows are the clock factors, each high only where its attribute's hour is 0; the labels ×1, ×3, ×9 and ×27 count how often each pattern repeats along the axis, because the cards are ordered with number changing slowest and colour fastest. Their product, the bottom row, is the join's softmax weight: one spike at the set. **c**, Four clocks multiply into one bright card ($\beta = 4.9$, learned by a six-clock layer), which is exactly a weighted sum of the 81 characters (the grid; its detail is not needed). The outlines mark the four clocks, one member of each conjugate pair. **d**, Time and memory per training step against tokens per sequence: the join runs out of memory at 1,024 tokens, Newton stays flat, and ordinary softmax attention (for reference) grows slowly; exponential Newton was not timed. **e**, One-set deals at 8, 16, 20 and 40 table cards (set pair outlined) and accuracies. Solid bars: trained on 8-card tables, new cards in hand (six-clock exponential Newton, Newton; mean of three seeds, ticks for seeds). Hatched: hand-set layers with four exact clocks, using the size rule for β at 40 cards, or the 8-card β held fixed. Newton's mean read-out falls at 16 cards, the fixed temperature at 40.

5 Temperature and size: the size law

Chapters 3 and 4 showed layers that keep working on inputs much larger than any in training, and layers that break. This chapter explains the difference with the paper's second theorem, the size law, and then shows it measured.

One voice against a room of whispers

Many questions here are existence questions: is there a walk from this child to the goal? Is there a pair on the table that completes a set? Suppose a layer answers by scoring every candidate witness and adding up exponential gates. Call a witness *true* if its score is at least a level a and *false* if its score is at most $a - \Delta$, where $\Delta > 0$ is the **margin**. Measure each gate relative to the level:

$$Z = \sum_{w \in F} \exp\big(\beta(s(w) - a)\big).$$

Two facts follow at once. If some witness is true, its own term is at least $e^0 = 1$ and all other terms are positive, so $Z \geq 1$. If no witness is true, every term is at most $e^{-\beta\Delta}$, so $Z \leq |F|\, e^{-\beta\Delta}$, where $|F|$ is the number of candidates. That second step is a **union bound**: a sum of many small terms is at most their number times the largest. So if $|F|\, e^{-\beta\Delta} < 1$, the threshold "$Z \geq 1$" separates the two cases; taking logarithms, $\beta\Delta > \log|F|$ (Figure 16).

The bound is also tight. An adversary can put every false witness exactly at the margin, so that their whispers add up to exactly $|F|\, e^{-\beta\Delta}$, while a true input can have Z just above 1, with one witness at the level and the rest far below. If $|F|\, e^{-\beta\Delta} > 1$ the false input is louder than the true one, and no threshold separates them.

> **Theorem 2** (Size law). *Let F be a finite nonempty family with scores $s(w)$, a level a, a margin $\Delta > 0$ and $\beta \geq 0$, such that every witness is true, $s(w) \geq a$, or false, $s(w) \leq a - \Delta$. Let $Z = \sum_{w \in F} e^{\beta(s(w)-a)}$. If $\beta\Delta > \log|F|$, some witness is true exactly when $Z \geq 1$. If no witness is true, $Z \leq |F|\, e^{-\beta\Delta}$, with equality when every score is $a - \Delta$. Hence, for $|F| \geq 2$ and scores free within the margin, some threshold on Z decides existence on every input if and only if*
>
> $$\beta\Delta \geq \log|F|,$$
>
> *with the strict threshold $Z > 1$ at equality. Under the same condition, when a true witness exists, the softmax weight on the true witnesses is at least one half.*

LEAN: `Framework.size_law_iff`, `Framework.size_law_equality`, `Framework.gated_existence`, `Framework.gated_existence_log`, `Framework.background_le`, `Framework.tight`, `Framework.softmax_existence`

The whole theorem is checked in Lean: both directions of the "if and only if", the equality case and the softmax statement. For single tokens the law restates a known requirement, that attention logits must grow with the logarithm of the sequence length[22–24]. What the paper adds is the statement for families of witnesses and its worst-case tightness.

Analogy: one voice against a growing room of whispers

A true witness speaks at full voice. Each false one whispers, at a volume $e^{-\beta\Delta}$ set by the margin and the temperature. One whisper is nothing, but whispers add up, and a large enough room drowns the voice. Lowering the temperature (raising β) quiets every whisper exponentially; it is the only lever that keeps pace with a growing room. *Where it breaks:* the theorem assumes the worst case, every false witness as loud as the margin allows. In real inputs most false witnesses are far below the margin and barely audible, so the room is effectively smaller; counting only the loudest whisperers predicts the threshold well, as the measurements below show.

What the law says for each family

The required temperature depends on the family's size:

- **Single tokens**: $|F| = n$, so $\beta\Delta$ must grow like $\log n$.
- **Ordered pairs**: $|F| = n(n-1)$, about n^2, so $\beta\Delta$ must grow like $2\log n$, twice as fast.
- **Subsets of** m: about $m \log n$.
- **Walks of every length**, without the row cap: the total walk weight W of the road gates takes the place of $|F|$, and a witness of ℓ steps whose gates are each $g < 1$ carries only g^ℓ, so the margin also pays a path-length cost, $\ell \log(1/g)$. A gate ceiling well below one therefore costs accuracy on long routes.

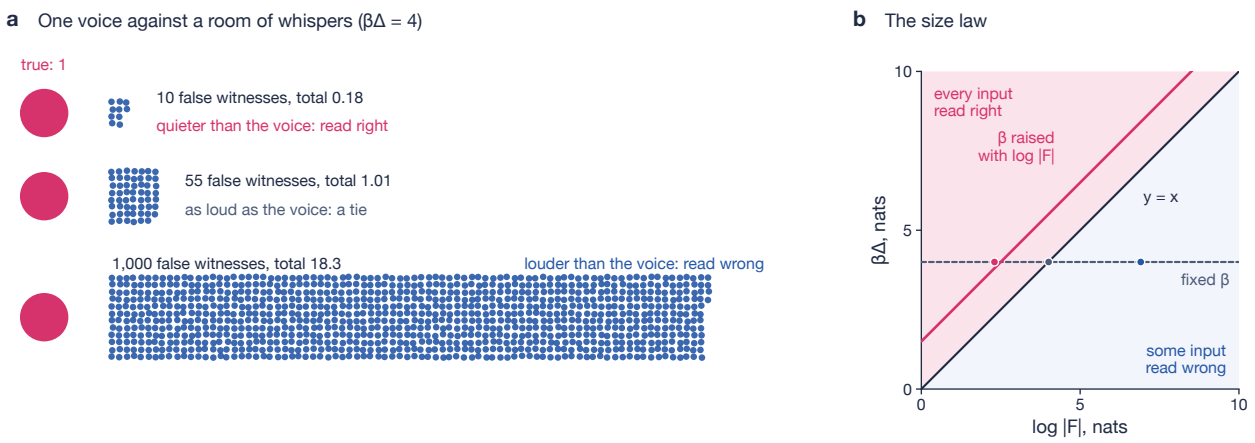


Figure 16 | The size law as one voice against a room of whispers. a, One true witness contributes 1 to Z (rose disc); each false witness at the margin contributes $e^{-\beta\Delta}$ (blue dots, drawn with area proportional to their contribution). At $\beta\Delta = 4$, ten false witnesses add up to 0.18, quieter than the voice; 55 add up to 1.01, a tie; 1,000 add up to 18.3. **b**, The plane of $\log|F|$ against $\beta\Delta$. Above the line $y = x$ existence is read right on every input; below it, some input is read wrong. A fixed temperature (dashed) crosses the line once the family is large enough; the three dots are the rows of panel a. A temperature that rises with $\log|F|$ (rose) stays above. (Toy illustration.)

Exponential gates against cancellation

There is another way to silence false witnesses: make them cancel. Newton attention's characters do this; in the one-attribute example of Chapter 4 a non-set pair contributes $1 + 2 \times (-\frac{1}{2}) = 0$ exactly, and exact cancellation by group characters works at every size. But trained characters are only approximately right. Suppose each false witness leaves a small independent residual of size δ, positive or negative at random. Random residuals add like a random walk, growing like δ times the square root of their number, and the paper shows that once there are $1/\delta^2$ false witnesses the read-out errs with probability at least 3/32. Exponential gates need a temperature that grows like $\log|F|$ at fixed precision; approximate cancellation needs every term's precision to improve as the family grows.

Read-outs divided by the number of candidates

A third failure is subtler. A read-out that divides its existence signal by the number of candidates, with its threshold fixed at one size, fails at larger sizes at any precision: one witness's share of the total shrinks as the candidates multiply. Newton's mean read-out, trained on 8 cards, answers 73 to 76% at 16 (Chapter 4). The looped pair layer that sums over the middle vertex with the fixed factor $1/\sqrt{n}$, trained at 49 vertices, answers 61 to 73% at 125, averaged over lookaheads.

Not every average fails. Triangle attention averages too, but it divides by the *total weight* (a softmax), not by the number of candidates, as the softmax half of the size law allows. Layers read through exponential gates, including sigmoid gates working in their low tail, transfer: trained at 49 vertices and tested at 125, the looped OR join answers at least 99.6% and looped triangle attention at least 99.2% at every lookahead; the propagator transfers to 250 vertices; exponential Newton from 8 cards to 20.

The law, measured

The paper tests the law on hand-set layers, nothing trained, where the margin is known.

- **The OR join on search.** A hand-set looped OR join needs an input gain (the slope of its sigmoid gates, which plays the role of β) rising from 13.5 at 50 vertices to 17.8 at 1,600: growth with the logarithm of the size.
- **Exponential Newton on SET.** A pair one attribute away from a set has one clock at $-\frac{3}{2}$ instead of 0, so the margin is $\Delta = \frac{3}{2}$, and the read-out threshold is one half. The rule $\beta = \frac{2}{3}\ln(2P)$, for P table pairs, makes $\beta\Delta = \ln(2P)$, so each of the at most P false pairs contributes at most $e^{-\beta\Delta} = 1/(2P)$, and together at most one half, the threshold, while a true set pair contributes at least 1. With this rule the hand-set layer is exact on tables of 8 to 40 cards, while the temperature that suits 8 cards answers 66.7% at 32 cards. On a larger made-up variant of SET with six attributes ($3^6 = 729$ cards, so tables of up to

364 cards can lack a set), the rule is right to 364 cards, the 8-card temperature fails from 48 cards and the 32-card temperature from 256 (Figure 17c).

The paper then measures the critical $\beta\Delta$, the smallest that answers every question, on 42 instances of four hand-set layers (Figure 17a). The worst case, $\log|F|$, lies 1.2 to 4.1 nats *above* the measured value for the SET layers and the OR join: real inputs are kinder than the worst case. Evaluated on the realised scores, the theorem's own condition bounds the false total $\sum_w e^{\beta(s_w-a)}$, which is about $Me^{-\beta\Delta}$ when M **competitors sit at the margin** and the rest are far below; for SET they are the pairs one attribute away from a set. With $\log M$ in place of $\log|F|$, every measured flip lies within 0.3 nats of the line $\beta\Delta = \log M$, except the OR join's beyond 200 vertices (0.8 nats at 1,600). This is a **data collapse**: four layers on two tasks fall on one line once the axes are chosen right. For the propagator, $\log W$ lies 0.1 to 1.4 nats below the measured value, because its margin also pays the path-length cost.

The law also shaped a trained model. On CLRS-30 (Chapter 6), the trained OR-join block needed its input gates shifted by a learned multiple of $\log n$: without the shift it came within a point of triangle attention on 8 of 12 tasks, with it on 11.

Check your understanding

1. From "a true witness contributes at least 1" and "each false witness contributes at most $e^{-\beta\Delta}$", derive the condition $\beta\Delta > \log|F|$.
2. If n doubles, by how much must $\beta\Delta$ grow for a layer over single tokens, and for one over ordered pairs?
3. Why does a read-out divided by the number of candidates fail at larger sizes even when every feature is exact? Why does a softmax average not?
4. Why does approximate cancellation break as the family grows, while exponential gates hold?
5. Why is the SET temperature rule $\beta = \frac{2}{3}\ln(2P)$ and not $\frac{2}{3}\ln P$?

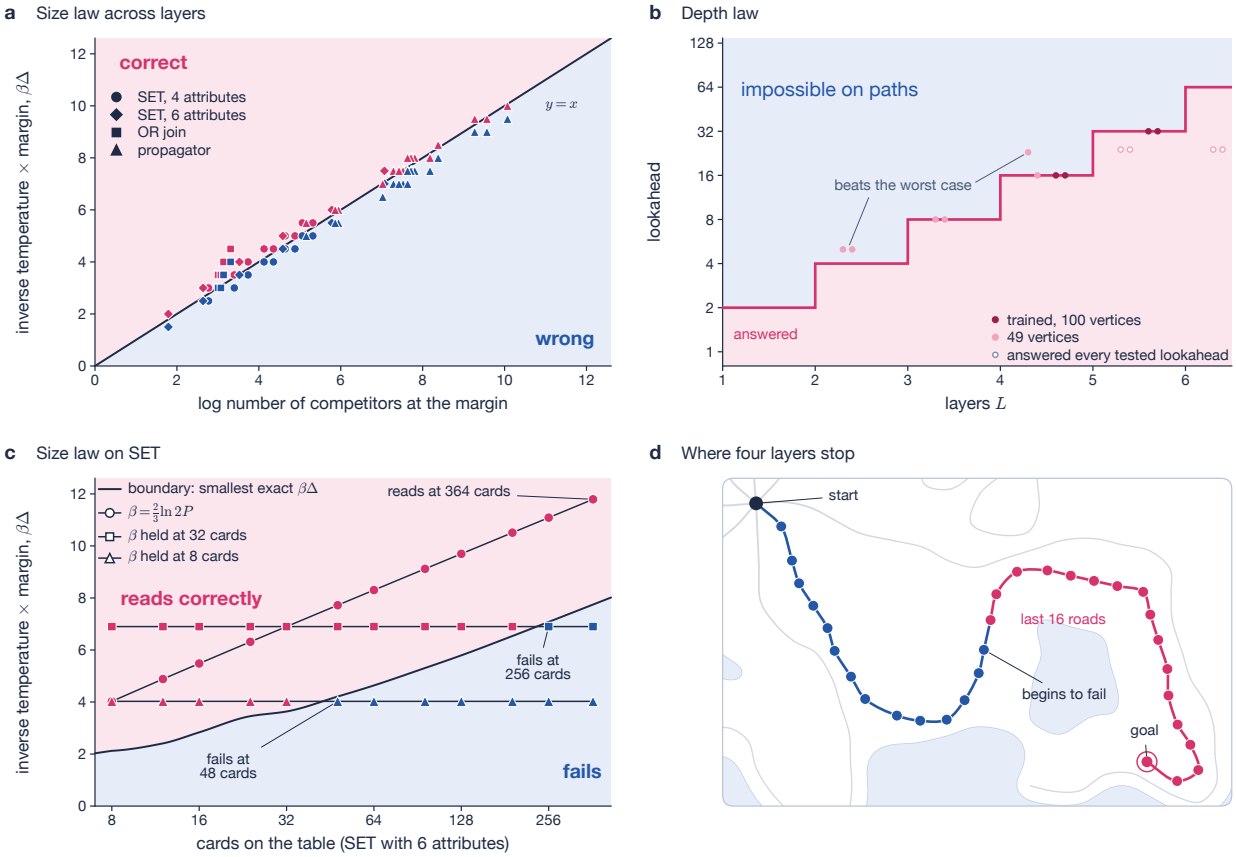


Figure 17 | Paper Fig. 4: the two laws, measured (panels b and d were read in Chapter 3). **a**, The size law across four hand-set layers, 42 instances: exponential Newton on SET with four attributes (8 to 40 cards) and six (8 to 364 cards), the one-layer OR join (random acyclic graphs of 50 to 1,600 vertices) and the propagator without the row cap (50 to 800 vertices). Each instance is drawn at the two grid values of $\beta\Delta$, 0.5 nats apart, that bracket its flip (rose, every question right; blue, some wrong), against the logarithm of its competitors at the margin. The flips hug $y = x$. **b**, The depth law (Figure 9). **c**, Six-attribute SET. Dark curve: the smallest $\beta\Delta$ right on all 1,000 set-free deals at each table size. Markers: three temperature schedules on those deals and 200 one-set deals; the rule $\beta = \frac{2}{3}\ln 2P$ (circles) stays right up to 364 cards, β held at its 32-card value (squares) fails from 256 cards, and at its 8-card value (triangles) from 48. **d**, Where four layers stop (Figure 9).

6 What the principles predict elsewhere

So far the principles have explained two tasks the paper chose. This chapter asks what they predict on a standard benchmark built by others, and on language and chess, where the paper expects pairs to matter little.

CLRS-30 in brief

The **CLRS-30** benchmark[25] asks networks to imitate 30 classical algorithms from the textbook by Cormen, Leiserson, Rivest and Stein; the paper uses its twelve graph tasks. Models train on graphs of up to 16 nodes and are tested on graphs of 64. The benchmark can supply **hints**, the algorithm's intermediate states, as extra supervision; the paper trains **without hints**, from inputs and final outputs only. Scores are accuracies in percent, and differences between them are in **points** (percentage points). The twelve tasks:

- **BFS** and **DFS**: each node's parent in the breadth-first and the depth-first search. CLRS's DFS takes neighbours in index order and restarts from the lowest-numbered unvisited node when it runs out, so its output is the *lexicographic* DFS **forest** (a set of trees).
- **Topological sort**: an order of a DAG's nodes in which every edge points forward.
- **Articulation points** and **bridges**: the nodes, and the edges, whose removal disconnects the graph.
- **Strongly connected components (SCC)**: the groups of nodes that can all reach each other; the output names one representative node per group.
- **Kruskal** and **Prim**: two ways to build a minimum spanning tree (the cheapest spanning tree); Kruskal's output marks the tree's edges, Prim's gives each node's parent in the tree grown from a start.
- **Bellman–Ford**, **Dijkstra** and **DAG shortest paths**: each node's predecessor on its shortest path from a source. Without hints, Bellman–Ford and Dijkstra have the same test inputs and answers.
- **Floyd–Warshall**: the predecessor on the shortest path between *every pair* of nodes. Its answer is a pair.

One small looped block against a large model

The paper's model is one looped block over a pair state of width 128 (each pair holds 128 numbers), with 186,000 parameters, iterated eight times at 64 nodes. Half of its channels are OR joins. The other half take a **power mean** over the middle node[26],

$$\Big(\sum_k (x_{uk}\, y_{kv})^\beta\Big)^{1/\beta},$$

where x and y are learned gates; as β grows a power mean approaches its largest term, so if the gates are exponentials of minus distances, it picks out the shortest route through k. Both halves shift their input gates by a learned multiple of $\log n$, as the size law asks (Chapter 5). The design was chosen at width 32 on the benchmark's official 64-node test graphs, confirmed at width 32 on freshly drawn test graphs and new seeds (within a point of looped triangle attention on 10 of 12 tasks), and the width-128 layer was then registered with its predictions and run once.

The comparison is **FloydNet**[27], a pair-state network of about 73 million parameters at its smallest depth, trained without hints. On the official test file the paper's block matches (within half a point) or beats FloydNet on 9 of 12 tasks; it is below on Bellman–Ford (by 0.9 points), Prim's algorithm (2.1) and strongly connected components (2.2).[13] On 128 freshly drawn test graphs per task it matches or beats on 8 of 12. The comparison crosses protocols: FloydNet's scores are from its paper (online training, 32 to 80 layers), the block's from 80,000 training steps on a fixed pool of graphs, three seeds and final parameters.

The cleanest test of the first principle is Floyd–Warshall, whose answer is a pair. The two pair-state models score 96.2% (FloydNet) and 98.3% (the paper's block), while the no-hint models of Bevilacqua et al.[28] and Rodionov and Prokhorenkova[29], which keep a state per node, score 40.8% and 42.2%. A node-state model must squeeze a pair-valued answer into single-node states: the index-card problem again.

Parallel and sequential: NC and P

Two tasks, DFS and SCC, need a new idea: problems that can be solved fast in parallel against problems that seem to need a long sequence of steps.

[13] On Dijkstra it is 0.49 points below FloydNet, inside the half-point band by 0.01. On the fresh graphs Dijkstra falls 0.003 points short, which is why the fresh-graph count is 8. Of the registered predictions, matching or beating on at least 8 of 12 tasks, beating FloydNet's 75.1% on DFS and reaching its 96.2% on Floyd–Warshall held; matching or beating on all 12, reaching 99.0% on Prim and 90.1% on SCC failed (9 of 12; 97.6; 87.9).

P (not the number of table pairs of Chapter 5) is the class of problems a computer can solve in a number of steps polynomial in the input size n. **NC** is the class that can be solved in *polylogarithmic* time, a number of steps bounded by a power of $\log n$ such as $(\log n)^2$, using polynomially many processors in parallel. Adding up n numbers is in NC: add them in pairs, halving the list each round, and finish in $\log_2 n$ rounds. Every problem in NC is in P; whether the classes are equal is a famous open question, and most researchers believe they are not. A **reduction** from problem A to problem B is a fast way to turn any question of A into a question of B with the same answer, so that a fast method for B would give one for A. A problem in P is **P-complete** if every problem in P reduces to it; such a problem has no fast parallel algorithm unless NC = P[30].

The closed forms of Chapter 2 are parallel: matrix inverses and determinants lie in NC[31], and so do repeated squaring and Newton's recursion. A layer that applies one of them plus entrywise operations is an NC computation, and so is a stack of polylogarithmically many such layers. **Lexicographic depth-first search** is P-complete[32]: deciding whether node u is visited before node v, when DFS always takes the lowest-numbered unvisited neighbour, is as hard as anything in P. So, unless NC = P, no stack of polylogarithmically many such layers computes it on all graphs, and a looped layer must run more than polylogarithmically often.[14]

> **Analogy: a treasure hunt that remembers**
>
> Some jobs split across many workers, such as adding up a column of numbers. Others look like a treasure hunt in which each clue sends you somewhere new. DFS is a treasure hunt with a memory: where you go next depends on every place you have already been, because you skip any node already visited. *Where it breaks* (**Background**): following a plain chain of pointers, with no memory, can be done in parallel by "pointer jumping", which is repeated squaring in disguise. What makes DFS hard, as far as anyone knows, is the dependence on the whole history of visited nodes.

SCC's representative. CLRS finds components with Kosaraju's algorithm: one lexicographic DFS of the graph, recording each node's **finish time** (the moment the search has explored everything below it), then a DFS of the graph with all edges reversed, taking nodes in order of decreasing finish time. The paper shows what the library's representative is, and how hard it is:

> **Proposition 4** (The component representative is P-complete). *The representative that CLRS-30 assigns to a strongly connected component is the component's first vertex discovered by lexicographic depth-first search. Deciding whether a given vertex is the representative of its component is P-complete.*

The proof is a reduction from "is u visited before v?". Add two new vertices u' and v', numbered 1 and 2, below every original vertex except the start 0, with roads $u \to u'$, $v \to v'$ and both ways between u' and v' (the other vertices are renumbered upwards). Then $\{u', v'\}$ is a component of its own, and whichever of u and v the search reaches first steps into it first: its representative is u' exactly when u is visited before v (Figure 18a). The paper checks the gadget against the library's own function on 400 of 400 random graphs. The components themselves can be found in parallel, through reachability; only the representative is P-complete. Topological sort is DFS too, but on acyclic graphs, where lexicographic DFS lies in NC by a result of de la Torre and Kruskal[33]; the block scores 99.9% on it.

Errors sit where the theory puts them

If the theory is right, the block's DFS and SCC errors should sit on the parts of the output that depend on the sequential order. They do.

- **SCC.** Every wrong SCC pointer points inside the correct component: the block finds the components and misses only the representative. Accuracy is 98.8 to 100% where the representative is the component's smallest index, which a parallel rule finds, and 66.5 to 70.4% where the first DFS pass fixes it.
- **DFS.** The block is right on every node up to depth 7 in the true DFS tree, and on 67 to 72% of nodes at depths 32 to 63.
- **More loops.** Trained with 4 to 24 loops instead of 4 to 8 and run 34 times at 64 nodes, the same layer's DFS accuracy rose from 80.5% to 90.7% (mean of three seeds; 89.8 to 92.4), increasing with each loop up to the training maximum and nearly flat after it (Figure 19b); FloydNet without hints scores 75.1%. This shows accuracy rising with the number of loops. It cannot tell apart how fast the needed loop count grows with graph size: one size, 64 nodes, and 34 loops are consistent with polylogarithmic and with faster growth alike.

[14] With a technical condition on how the weights may depend on the graph size. The argument does not say how many loops are enough.

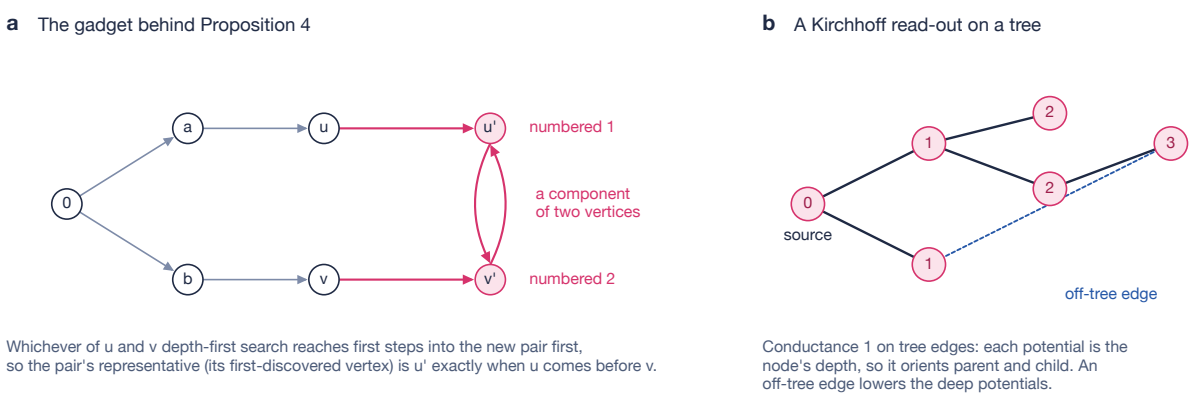


Figure 18 | Two pictures for Chapter 6. a, The reduction behind Proposition 4: two new vertices u' and v' (rose), numbered first, reached from u and from v and joined both ways. The search enters the pair from whichever of u and v it visits first, and that decides the pair's representative. **b**, A Kirchhoff read-out on a small tree, grounded at the source: with conductance 1 on the tree's edges, each node's potential (written inside) is its depth, which orients every parent–child edge. An edge outside the tree (dashed) is a shortcut that lowers the deep potentials. (**Background** illustration.)

- **More loops for SCC.** The same change on strongly connected components was registered with the prediction that only the order-dependent representatives would gain. Every wrong pointer stayed inside its component and the smallest-index representatives stayed at 98.8%, but the order-dependent ones rose only from 69.0% to 72.8%, within the spread of seeds, and overall accuracy levels off near 14 loops (89.0% at 34, against FloydNet's 90.1%). The account locates these errors; at this width, more loops do not remove them, unlike DFS's.

Near ties, and one exception

The remaining errors on weighted tasks sit at **near ties**, where the margin of the size law is small. A decision's **relative margin** is how much worse the best alternative's path weight or distance is than the true choice's. On Bellman–Ford and Dijkstra, the median relative margin of the decisions the block gets wrong is 0.5 to 0.6%, against 20.5% over all decisions; on Kruskal's algorithm the two medians are 1.3% and 42% (Figure 19c).

Prim's algorithm is the exception. Its errors are not near ties (median 30% against 51% over all decisions), two thirds of them point to the node's own child (the right tree edge in the wrong direction), and they grow with depth in the tree. The principles do not explain them. The paper tried the family they suggest, spanning trees, through a **Kirchhoff read-out**. Treat each edge as a wire with a learned **conductance** (how easily current flows through it); the graph's **Laplacian** is the matrix with each node's total conductance on the diagonal and minus the conductance of each edge off it. Holding the source at zero (grounding) and solving one linear system gives every node a potential; the matrix-tree theorem says the same Laplacian, through its determinant, sums over all spanning trees. With conductance 1 on the true tree's edges the read-out is exact on all 2,048 official test nodes (Figure 18b). But the trained layer gives confident conductance to edges outside the tree at 64 nodes, and in two registered tests neither the Kirchhoff read-out (97.20%) nor a learned $\log n$ shift of its conductances (96.84%) removed Prim's errors; without them the block scores 97.57%. Prim remains open.

Where joins add little: language and chess

The first principle also predicts where pair layers should *not* help: tasks whose witnesses are mostly single tokens.

Language. In an 8-layer language model of 51 million parameters, a join over the previous 16 tokens, added in every second block, gains 0.011 nats per token after 200 million tokens of FineWeb-Edu[34]; two more ordinary layers gain 0.026 (Figure 19d). The models were trained at one scale, budget and window; joins over long contexts gain more[35], a regime the paper did not test.

Chess. In models that pick the solving move of Lichess puzzles, joins beside or in place of attention never gain more than 0.1 points of accuracy. On the 6,681 puzzles about pins, skewers, x-rays and discovered attacks, motifs that pair two pieces on a line and where the first principle predicts a gain, joins beside attention score

84.6 to 85.3% against 85.0% without.

Why. A join's **interaction share** is the fraction of the variance of its scores over pairs that is *not* a sum of one score per partner, relative to the same fraction for random scores: near 0, the join is really two lookups; near 1, it uses pair structure as much as random scores would. Trained SET joins have shares of 0.98 to 1.02; trained chess joins 0.11 to 0.29, where scores remain mostly sums of lookups on single squares. A share near one shows that a join is *used*, not that it is *needed*: chess joins whose keys have their mean over the squares subtracted reach 1.03 and still add nothing.

Check your understanding

1. Why can no stack of polylogarithmically many closed-form layers compute lexicographic DFS on all graphs, unless NC = P? What does the 34-loop experiment show, and what can it not show?
2. Why do node-state models score around 40% on Floyd–Warshall while pair-state models score above 96%?
3. Every wrong SCC pointer of the block points inside the correct component. What does that tell you about what the block has and has not learned?
4. What is a near tie, and why does the size law predict that errors concentrate there?
5. Chess joins with centred keys have an interaction share of 1.03 and still add nothing. What does that say about the share as evidence?

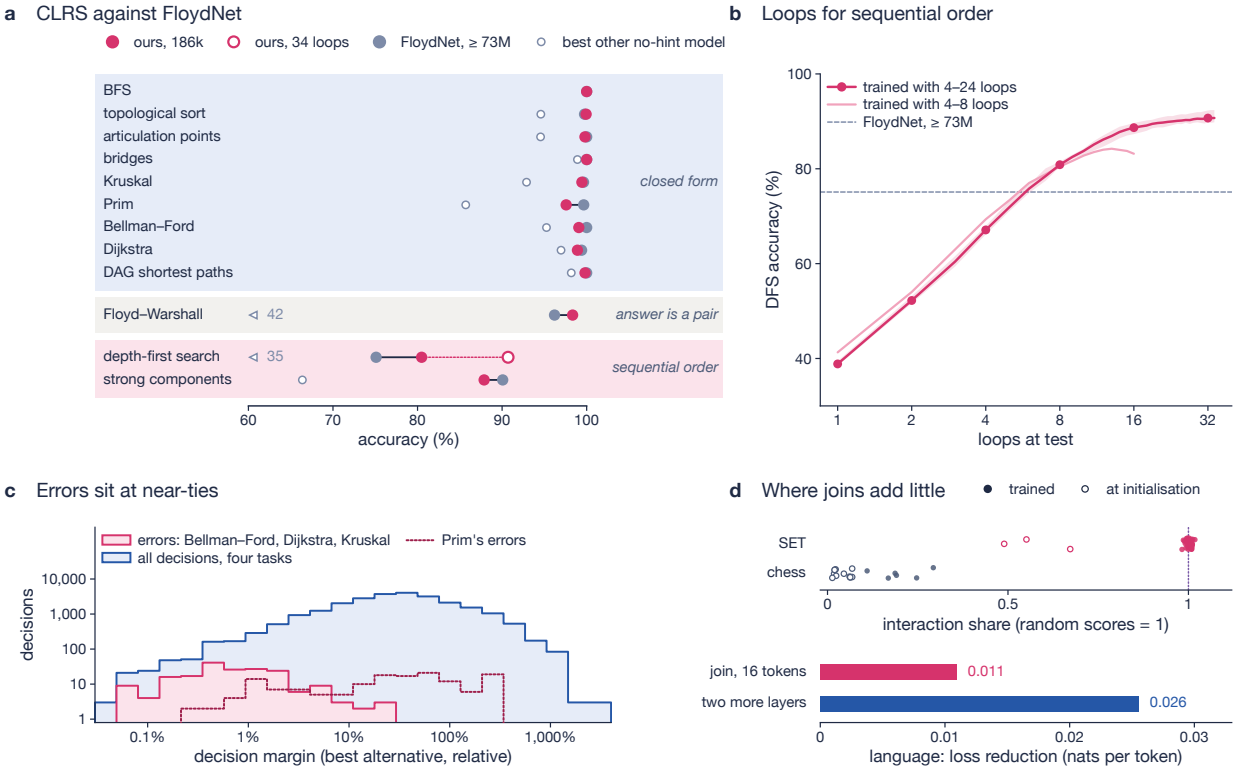


Figure 19 | Paper Fig. 5: the principles across CLRS-30, and where joins add little. **a**, Accuracy on the twelve CLRS-30 graph tasks at 64 nodes, trained on up to 16 without hints, grouped by the paper's classes: tasks the paper classes as having a closed form (blue band; it does not say which closed form serves each task), the task whose answer is a pair (grey), and the tasks whose output follows sequential order (rose). Rose dots: the paper's 186,000-parameter block (mean of three seeds, eight loops; open, DFS at 34 loops). Grey dots: FloydNet (published). Small open circles: the best other no-hint model, with arrows for values off the scale. **b**, DFS accuracy against loops at test, trained with 4 to 24 loops (dark, band over three seeds) or 4 to 8 (light); dashed, FloydNet. **c**, Relative decision margins: all decisions on four tasks (blue), the errors on Bellman–Ford, Dijkstra and Kruskal (rose, far left: near ties) and Prim's errors (dashed, not near ties). **d**, Interaction shares of trained joins (filled) and joins at initialisation (open), SET against chess; below, the language model's gain from a join over 16 tokens and from two more layers.

7 What it means

The account in one paragraph

Higher-order attention is a question of *family*, not of *arity* (the number of tokens one score combines). A layer over pairs helps when a task's witnesses are pairs or are composed from them, as in search and SET, and not when they are single tokens, as in the paper's language and chess models. Much of the depth that pair models spend emulates a family that has a closed form: eight pair layers compose two-step walks into the reach that one propagator layer reads from $(I - G)^{-1}$. Where no parallel closed form exists, as for lexicographic depth-first order, depth is the right tool, and the depth needed grows with the input. And temperature decides how far an answer carries: exponential gates with a logarithmic margin transfer to larger inputs, while read-outs divided by the number of candidates, trained at one size, do not.

The whole zoo

Table 4 is the paper's Table 1, with every layer now introduced. Two phrases need explaining. "A family only after their threshold or OR": stacked or looped pair layers each sum over two-step walks of the previous layer's relation, and the whole stack sums over walks of up to 2^L steps only if each layer cleans its output back to 0 or 1, by a threshold or an OR; otherwise counts from different layers mix and no single family describes the stack. And the Kirchhoff layer's read-out, **marginals**, means each edge's probability of belonging to a spanning tree drawn at random with probability proportional to the product of its edge weights (**Background:** the inverse of the Laplacian gives them).

The form is not universal. A layer that pools its inputs into one sum and then applies a learned non-linear map, as Deep Sets do[36], is not one of these read-outs, and linear attention fits the form only with signed potentials (weights that can be negative), outside the size law.

What was known, and what is new

Each piece of the account has a precedent, which the paper names: attention as an expectation with its own partition function[37]; one parsing algorithm run in Boolean, counting and shortest-path arithmetic[38]; dynamic programming smoothed at a temperature[10]; the noisy-OR[9]; differentiable chaining of relations through a middle element[39–41]; **algorithmic alignment**, the idea that a network learns an algorithm more easily when its structure lines up with the algorithm's steps[42–44], the closest earlier form of the first principle; logits that grow with $\log n$, the token case of the size law[22–24,45]; exponential against polynomial energies in associative memories[46,47]; and sums over distinct tuples of neighbours at linear cost[48]. New in the paper are the account across witness families, the two laws in that form, the P-completeness reduction for the strongly connected components' representative, three layers (the propagator, Newton attention and exponential Newton) and the registered tests.

Limits

- The positive results are on synthetic and algorithmic tasks, with small models; language was tested at one scale, budget and join window, and long contexts were not tested.
- The first two principles are hypotheses supported by experiments. The size law is a worst-case statement; the depth law's lower bound is proved from anonymous starts, on paths, and the trained models' match to 2^L is empirical.
- The CLRS-30 design was chosen on the official test graphs (hence the fresh-graph check, 8 of 12), and the comparison with FloydNet crosses training protocols.
- Exponential Newton is given the group, and its cost advantage is untested on long sequences.
- More loops raised DFS accuracy but did not remove SCC's order-dependent errors at this width.
- Newton attention's exact count form did not train to the rule; Prim's errors remain unexplained after two registered fixes failed.

Open questions you could work on

1. **Prim.** What family, or what training, removes the wrong-direction errors that the Kirchhoff read-out did not?
2. **Named starting states.** The depth law's lower bound assumes anonymous starts. What can be proved when vertices have names, and why do trained models with names still stop near 2^L?

Table 4 | The paper's Table 1, the layer zoo. Each layer as a family, the closed form of its sum, its cost for all n queries (or all n^2 pairs) and its read-out (Equation 4). Separate pair layers and looped ones are a family only after their threshold or OR. The Kirchhoff layer was tested only as a read-out, on Prim's algorithm.

Layer	Family	Closed form of Z	Cost	Read-out
Softmax attention	single tokens	a sum	n^2	average
2-simplicial join	ordered pairs of tokens	none (enumerated)	n^3	average
Pair layer, sum over k	two-step walks	matrix product	n^3	count, scale $1/\sqrt{n}$
Triangle attention	two-step walks	none (enumerated over k)	n^3, memory n^3	average
OR join	two-step walks	matrix product	n^3, memory n^2	OR
L pair layers, or one looped L times	walks of at most 2^L steps	repeated squaring	$L\,n^3$	OR, threshold
Propagator	walks of every length	$(I-G)^{-1}$	one solve, n^3	OR, free energy
Newton attention	m-subsets	$[\lambda^m]\prod_j(1+\lambda\varphi_j)$	n per channel	count; trained as a mean
Exponential Newton	unordered pairs, clock energies	3^d characters	$3^d\,n$	average
Kirchhoff layer	spanning trees	determinant	n^3	marginals

3. **Learning the group.** Can a layer discover its clocks and their group from data, and keep exponential Newton's sharp softmax and linear cost? At what sequence length does $3^d n$ beat the $\binom{n}{2}$ pairs in practice?
4. **Families for language.** The paper points out that parses are trees and spans, families with closed forms of their own[37,49], and that a pair state over the *entities* a text names, rather than its tokens, would place the search layers inside a language model. Its Supplement takes a first step: 49-vertex test graphs described in English, one sentence per road, were turned back into edge lists by a language model (Qwen3-8B[50]) with 99.9% precision and recall, and a looped triangle-attention layer trained only on the benchmark's tokens then answered 97 to 100% at every lookahead. The language model answering alone reached 90 to 100% with a reasoning budget of 16,384 tokens, and was near chance beyond two steps without reasoning.
5. **The size law on real inputs.** Counting competitors at the margin predicts the measured threshold within 0.3 nats, except the OR join beyond 200 vertices. What explains that drift?
6. **Making the first principle refutable.** When exactly does building a missing family inside token states make learning hard?

A Glossary

Acyclic graph, DAG. A directed graph with no walk that returns to its start.

Adjacency matrix A. $A_{uv} = 1$ if there is an edge $u \to v$; entry (u, v) of A^ℓ counts walks of ℓ edges.

Algorithmic alignment. A network learns an algorithm more easily when its structure matches the algorithm's steps; an earlier, general form of the first principle.

Arity. The number of tokens one score combines: one for a lookup, two for a pair join.

AUC. Area under the ROC curve of a classifier: 1 is perfect, 0.5 is chance.

Average, count, OR and free-energy links. The four read-outs of a partition function: Y/Z; a fixed multiple of Z; $1 - e^{-Z}$; $\beta^{-1} \log Z$.

Characters of $(\mathbb{Z}_3)^4$. The 81 functions $\chi_{\mathbf{s}}(x) = \omega^{\mathbf{s}\cdot x}$, $\omega = e^{2\pi i/3}$: one trivial and 40 conjugate pairs. They turn sums of cards into products, and their average detects a set.

Clock. A rule that reads an hour (0, 1 or 2) and scores it by $\cos(2\pi \cdot \text{hour}/3)$; as characters, one conjugate pair on one attribute. SET has four.

Closed form. A formula for a family's partition function that does not list its members: a product for subsets, a matrix inverse for walks, a determinant for spanning trees.

CLRS-30. A benchmark of 30 classical algorithms; the paper uses its 12 graph tasks, training on up to 16 nodes and testing on 64.

Competitors at the margin. False witnesses whose scores sit just one margin below the level; they dominate the false total in practice.

Conductance. A positive weight on an edge, read as how easily current flows through it; a Kirchhoff read-out learns them.

Census and window (depth law). After ℓ rounds a pair sees the labels within $2^\ell - 1$ roads of each end (its window) and one count over the whole path of how many vertices see each small window (the census).

Data collapse. Measurements from different systems falling on one curve once the axes are chosen right.

Depth law. L pair layers decide reach within $2^L - 1$ steps from a child and, from anonymous states, no further.

Depth-first search (lexicographic); forest; finish time. DFS that always takes the lowest-numbered unvisited neighbour; its trees form a forest; a node's finish time is when everything below it has been explored. Deciding its visiting order is P-complete.

Elementary symmetric polynomial e_m. The sum over all m-subsets of a list of numbers of their product (not Euler's number e).

Exponential gate. A weight $\exp(\beta(s - a))$: at least 1 when the score reaches the level, shrinking exponentially below it. A sigmoid gate behaves like one in its low tail.

Exponential Newton. The paper's layer that computes the join's softmax over pairs exactly, for clock scores with the modulus given, through 3^d channels of Newton's e_2.

Family (of witnesses). The candidate witnesses a layer sums over for one query.

FloydNet. A pair-state network of about 73 million parameters at its smallest depth.

Folklore 2-WL. The Weisfeiler–Leman test on pairs; pair layers can distinguish no more than it does.

Gate. A positive weight that multiplies what passes through; a sigmoid gate lies between 0 and 1.

Gershgorin's circle theorem. Every eigenvalue lies in a disc around a diagonal entry with radius the rest of that row's absolute sum, so row sums bound the spectral radius.

Group (abelian). A set with an addition that has a zero and can be undone; abelian if order does not matter. SET's cards modulo 3 are one.

Hand-set layer. A layer whose weights are set by hand to compute a known rule.

Hints (CLRS-30). An algorithm's intermediate states, optionally used as extra supervision.

Interaction share. The fraction of a join's score variance over pairs that is not a sum of one score per partner, relative to random scores.

Inverse temperature β. The multiplier of scores inside the exponential; large β means sharp weights.

Jacobi–Anger expansion. The Fourier series of $\exp(\beta \cos\theta)$; on three hours it has exactly three terms.

Join. A layer that scores combinations of tokens; 2-simplicial attention, with a trilinear score over pairs, is

the simplest.

Kirchhoff read-out; Laplacian; marginals. A read-out over spanning trees: the Laplacian holds each node's total conductance on its diagonal and minus each edge's conductance off it; its determinant sums over spanning trees, and its inverse gives potentials and each edge's probability of lying in a random tree (its marginal).

Kosaraju's algorithm. Components from two DFS passes; CLRS names each by its first vertex discovered in the first pass.

Lean, Mathlib. A proof assistant in which a computer checks every step, and its mathematics library.

Lookahead. The number of steps needed to tell the correct branch from the others in a search question.

Lookup. Ordinary attention: a query scores each token on its own.

Looped layer. One block with one set of weights, applied several times in a row.

Margin Δ. The score gap between the lowest true witness and the highest false one.

Message-passing network. A graph network whose vertices repeatedly update their states from their own state and their neighbours'.

Modulus 3. The rule that hours wrap around after three; exponential Newton is given it.

Nats; points. Natural-log units; percentage points.

NC, P, P-complete, reduction. Problems solvable fast in parallel (time a power of $\log n$, polynomially many processors); in polynomial time; in P and as hard as anything in P under reductions, fast ways to turn one problem's questions into another's.

Newton attention. The paper's layer that sums over subsets of partners through Newton's identities, at cost linear in their number.

Newton's identities. Formulas that give elementary symmetric polynomials from power sums, such as $e_2 = (p_1^2 - p_2)/2$.

OR join. A pair layer that sums over middle vertices through sigmoid gates on both legs and reads the total with the OR link.

Pair state. A vector for every ordered pair of elements, updated through a third element.

Partition function Z. The sum over a family's witnesses of each witness's weight, a product of its parts' potentials.

Potential. The positive weight of one part of a witness.

Power mean. $(\sum_k x_k^\beta)^{1/\beta}$; a soft maximum for large β.

Power sum p_m. $\sum_j \varphi_j^m$.

Propagator. The paper's layer that computes the walk sum $X = S(I - G)^{-1}$ from gates and a source learned on a pair state, in one linear solve.

Pseudo-set. A pair that completes a set on the attributes a layer clocks but not on the one it misses; a zero margin.

Registered. Written down, with predictions, budget and stopping rule, before the runs that test it.

Relative margin (CLRS). How much worse the best alternative's path weight or distance is than the true choice's.

Repeated squaring. Doubling the walk length covered at each step by composing a relation with itself.

Resolvent. $(I - G)^{-1}$, the sum over walks of every length.

Row cap. Scaling each row of the propagator's gates to sum to at most $t_0 = 0.99$, so the walk sum converges on any graph.

SET. A card game on $(\mathbb{Z}_3)^4$: three cards form a set when they sum to zero, that is, when every attribute is all the same or all different.

Size law. In the worst case, existence is read correctly at every size if and only if $\beta\Delta \geq \log |F|$.

Spanning tree. A set of edges that connects all vertices without a cycle.

Spectral radius. The largest absolute value of a matrix's eigenvalues; below one, the geometric series converges.

Strongly connected component. A maximal group of nodes that can all reach each other.

Temperature $1/\beta$. The scale of the log-potentials.

Triangle attention. A pair layer with a softmax over the middle element of a score coupling both legs.

2-simplicial attention. A join that scores ordered pairs of tokens with a trilinear score.

Union bound. A sum of many terms is at most their number times the largest.

Walk, path. A sequence of edges, each starting where the last ended; a (simple) path never revisits a vertex.

Weisfeiler–Leman test. Colour refinement: repeatedly recolour each vertex (or pair) by its colour and the collection of its neighbours' colours.

Witness. An object a task's answer sums over: a token, a pair, a walk, a subset, a tree.

B Answers to the questions

Chapter 2. (1) Single tokens; the average, Y/Z. (2) A walk $u \to k \to v$ weighs $A_{uk}A_{kv}$, 1 if both roads exist, so the sum over k counts two-road walks, $(A^2)_{uv}$. (3) $\exp(\beta(f+g)) = \exp(\beta f)\exp(\beta g)$, so the sum over pairs is a product of two single sums; the trilinear score couples both partners in one number and does not factor. (4) The OR link, $1 - e^{-Z}$: near 0 when Z is near 0 and near 1 once Z is a few units, however many witnesses there are. (5) For very negative inputs $\sigma(x) \approx e^x$, so for candidates far below the level a sigmoid gate shrinks exponentially, as an exponential gate does.

Chapter 3. (1) A key is computed from one vertex's state before the question is known, so "the goal is reachable from v" must already be in v's state for every possible goal: its reach set. A pair state holds a relation for every pair and builds reach by composing pairs. (2) $2^3 - 1 = 7$ roads from the child, lookahead 8. (3) It shows that, from anonymous starts, no network of the class does better than the construction, for every L; the trained models start from names on graphs that are not paths, so their match to 2^L is empirical. (4) The sum $\sum_\ell G^\ell$ converges only if repeated multiplication by G shrinks vectors; with every row of a non-negative G summing to at most $t_0 < 1$, walks of ℓ steps carry at most t_0^ℓ on any graph. (5) Gates on reversed roads are open (0.97 to 0.98), others closed (below 10^{-3}), and the source sits on (current vertex, goal), so weight flows backwards from the goal and reaches child v exactly when v reaches the goal.

Chapter 4. (1) $f(a_j) + f(a_k) > f(a_k) + f(a_l)$ gives $f(a_j) > f(a_l)$; $f(a_l) + f(a_m) > f(a_j) + f(a_m)$ gives $f(a_l) > f(a_j)$. (2) $p_1 = 11$, $p_2 = 53$, $e_2 = (121 - 53)/2 = 34 = 4 + 6 + 24$. (3) 81 characters: 1 trivial and 40 conjugate pairs; the four clocks are 8 characters. The softmax's exponential turns the sum of four clocks into their product, the AND over all 81 characters; Newton's read-out is linear in each channel, so it must hold the 40 needed characters separately. (4) Dividing by $\binom{n}{2}$ and a root mean square shrinks one set's share and grows the background at 16 cards, moving the threshold from 0.54 sets to about 1.76 sets, so one-set deals are called negative; from two sizes the layer learns a threshold that depends on size. (5) A pair that completes a set on number, shape and fill but not colour scores exactly like a true set: the margin is zero, and the size law needs $\Delta > 0$; such pairs become common as tables grow.

Chapter 5. (1) True: $Z \geq 1$; false: $Z \leq |F|e^{-\beta\Delta}$; the threshold 1 separates them if $|F|e^{-\beta\Delta} < 1$, that is, $\beta\Delta > \log|F|$. (2) By $\log 2 \approx 0.69$ nats for single tokens, about $2\log 2 \approx 1.39$ for ordered pairs. (3) One witness's share of a total divided by the number of candidates shrinks as candidates are added, so a fixed threshold is crossed from the wrong side; a softmax divides by the total weight, which a true witness at low temperature dominates. (4) Independent residuals of size δ grow like δ times the square root of their number, so errors appear at about $1/\delta^2$ false witnesses; exponential gates shrink each false witness by $e^{-\beta\Delta}$, and raising β with $\log|F|$ keeps the total small. (5) The read-out threshold is one half: with $\beta\Delta = \ln(2P)$ the P false pairs add up to at most $P/(2P) = \frac{1}{2}$.

Chapter 6. (1) Closed-form layers compute NC functions and a polylogarithmic stack of them stays in NC, while lexicographic DFS is P-complete. The experiment shows accuracy rising with loops at one size; it cannot show how the needed loop count grows with n. (2) The answer is a pair; a node-state model must store, in each node's state, its relation to every other node, the wrong family. (3) It has learned the components, which are computable in parallel, and misses only the representative, which depends on the first DFS pass's order, the P-complete part. (4) A decision whose best alternative is almost as good: the margin is small, so a fixed temperature cannot separate the two. (5) The share shows that a join's scores use pair structure, not that the task needs it.

C Reading list

The problem.

- Saparov et al.[5], the search benchmark and the reach-set mechanism of trained Transformers.
- Sanford et al.[6], why attention can represent connectivity at logarithmic depth.
- Vaswani et al.[7], the Transformer itself.

Layers over pairs.

- Jumper et al.[1], AlphaFold, where pair states updated through a third element are central.
- Bergen et al.[2], the Edge Transformer and triangle attention.
- Clift et al.[8] and Roy et al.[3], 2-simplicial attention, the join of this tutorial.
- Yu et al.[27], FloydNet, the large pair-state model compared on CLRS-30.

Attention as a sum over structures.

- Kim et al.[37], attention as the expectation of a structured distribution with its own partition function.
- Goodman[38], one parsing algorithm run in different arithmetics; the idea behind the links.
- Mensch and Blondel[10], dynamic programming smoothed at a temperature.
- Xu et al.[42], algorithmic alignment, the closest earlier statement of the first principle.

Temperature and size.

- Chiang and Cholak[22] and Nakanishi[23], why attention logits must grow with $\log n$.
- Hayase and Karakida[24], the critical inverse temperature of single-token attention, which the size law extends to families.
- Veličković et al.[45], why softmax disperses as inputs grow.

Graphs, algorithms and their limits.

- Veličković et al.[25], the CLRS-30 benchmark.
- Maron et al.[13], pair networks and the 2-dimensional Weisfeiler–Leman test.
- Reif[32] and Greenlaw et al.[30], why depth-first search is inherently sequential, and P-completeness in general.
- Johnson et al.[17] and Zhu et al.[18], resolvents and path sums over given graphs.

Mathematics used.

- Macdonald[20], symmetric functions and Newton's identities.
- Segol and Lipman[51] and Zaheer et al.[36], set networks built from sums over elements.
- Olver et al.[21], the Jacobi–Anger expansion (equation 10.35.1).
- Gerschgorin[15], the circle theorem behind the row cap.

References

1. Jumper, J. et al. Highly accurate protein structure prediction with AlphaFold. Nature **596**, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2
2. Bergen, L., O'Donnell, T. J. & Bahdanau, D. Systematic Generalization with Edge Transformers. In Advances in Neural Information Processing Systems (2021). https://arxiv.org/abs/2112.00578
3. Roy, A. et al. Fast and Simplex: 2-Simplicial Attention in Triton. Preprint at arXiv:2507.02754 (2025). https://arxiv.org/abs/2507.02754
4. Das, A., Gupta, D. & Gupta, D. Linearized 2-Simplicial Attention. Preprint at arXiv:2608.09307 (2026). https://arxiv.org/abs/2608.09307
5. Saparov, A. et al. Transformers Struggle to Learn to Search. In International Conference on Learning Representations (2025). https://arxiv.org/abs/2412.04703
6. Sanford, C., Hsu, D. & Telgarsky, M. Transformers, Parallel Computation, and Logarithmic Depth. In International Conference on Machine Learning, 43276–43327 (2024). https://arxiv.org/abs/2402.09268
7. Vaswani, A. et al. Attention Is All You Need. In Advances in Neural Information Processing Systems, 5998–6008 (2017). https://arxiv.org/abs/1706.03762
8. Clift, J., Doryn, D., Murfet, D. & Wallbridge, J. Logic and the 2-Simplicial Transformer. In International Conference on Learning Representations (2020). https://arxiv.org/abs/1909.00668
9. Pearl, J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988).
10. Mensch, A. & Blondel, M. Differentiable Dynamic Programming for Structured Prediction and Attention. In International Conference on Machine Learning, 3462–3471 (2018). https://arxiv.org/abs/1802.03676
11. de Moura, L. & Ullrich, S. The Lean 4 Theorem Prover and Programming Language. In Automated Deduction – CADE 28, 625–635 (2021).
12. The mathlib Community. The Lean Mathematical Library. In Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, 367–381 (2020).
13. Maron, H., Ben-Hamu, H., Serviansky, H. & Lipman, Y. Provably Powerful Graph Networks. In Advances in Neural Information Processing Systems, 2156–2167 (2019). https://arxiv.org/abs/1905.11136
14. Fischer, M. J. & Meyer, A. R. Boolean Matrix Multiplication and Transitive Closure. In 12th Annual Symposium on Switching and Automata Theory, 129–131 (1971).
15. Gerschgorin, S. Über die Abgrenzung der Eigenwerte einer Matrix. Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles (Izvestiya Akademii Nauk SSSR), 749–754 (1931).
16. Gasteiger, J., Bojchevski, A. & Günnemann, S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In International Conference on Learning Representations (2019). https://arxiv.org/abs/1810.05997
17. Johnson, D. D., Larochelle, H. & Tarlow, D. Learning Graph Structure With A Finite-State Automaton Layer. In Advances in Neural Information Processing Systems (2020). https://arxiv.org/abs/2007.04929
18. Zhu, Z., Zhang, Z., Xhonneux, L.-P. & Tang, J. Neural Bellman-Ford Networks: A General Graph Neural Network Framework for Link Prediction. In Advances in Neural Information Processing Systems (2021). https://arxiv.org/abs/2106.06935
19. Zweig, A. & Bruna, J. Exponential Separations in Symmetric Neural Networks. In Advances in Neural Information Processing Systems, 33134–33145 (2022). https://arxiv.org/abs/2206.01266
20. Macdonald, I. G. Symmetric Functions and Hall Polynomials. 2nd edn, Oxford University Press (1995).
21. Olver, F. W. J., Lozier, D. W., Boisvert, R. F. & Clark, C. W. NIST Handbook of Mathematical Functions. Cambridge University Press (2010).
22. Chiang, D. & Cholak, P. Overcoming a Theoretical Limitation of Self-Attention. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 7654–7664 (2022). https://arxiv.org/abs/2202.12172
23. Nakanishi, K. M. Scalable-Softmax Is Superior for Attention. arXiv preprint arXiv:2501.19399 (2025). https://arxiv.org/abs/2501.19399
24. Hayase, T. & Karakida, R. A Unified Framework for Critical Scaling of Inverse Temperature in Self-Attention. arXiv preprint arXiv:2605.12697 (2026). https://arxiv.org/abs/2605.12697
25. Veličković, P. et al. The CLRS Algorithmic Reasoning Benchmark. In International Conference on Machine Learning (2022). https://arxiv.org/abs/2205.15659
26. Li, G., Xiong, C., Thabet, A. & Ghanem, B. DeeperGCN: All You Need to Train Deeper GCNs. Preprint at arXiv:2006.07739 (2020). https://arxiv.org/abs/2006.07739
27. Yu, J., Zeng, M. & Ye, Q. FloydNet: A Learning Paradigm for Global Relational Reasoning. Preprint at arXiv:2601.19094 (2026). https://arxiv.org/abs/2601.19094
28. Bevilacqua, B. et al. Neural Algorithmic Reasoning with Causal Regularisation. In International Conference on Machine Learning (2023). https://arxiv.org/abs/2302.10258
29. Rodionov, G. & Prokhorenkova, L. Neural Algorithmic Reasoning Without Intermediate Supervision. In Advances in Neural Information Processing Systems (2023). https://arxiv.org/abs/2306.13411

30. Greenlaw, R., Hoover, H. J. & Ruzzo, W. L. Limits to Parallel Computation: P-Completeness Theory. Oxford University Press (1995).
31. Csanky, L. Fast Parallel Matrix Inversion Algorithms. SIAM Journal on Computing **5**, 618–623 (1976).
32. Reif, J. H. Depth-First Search is Inherently Sequential. Information Processing Letters **20**, 229–234 (1985). https://doi.org/10.1016/0020-0190(85)90024-9
33. de la Torre, P. & Kruskal, C. P. Fast Parallel Algorithms for All-Sources Lexicographic Search and Path-Algebra Problems. Journal of Algorithms **19**, 1–24 (1995).
34. Penedo, G. et al. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In Advances in Neural Information Processing Systems (2024). https://doi.org/10.52202/079017-0970
35. Sieberling, O. et al. Triadic Linear Attention: Three-Dimensional Recurrent States for Long-Context Sequence Modeling. arXiv:2609.36529 (2026). https://arxiv.org/abs/2609.36529
36. Zaheer, M. et al. Deep Sets. In Advances in Neural Information Processing Systems (2017). https://arxiv.org/abs/1703.06114
37. Kim, Y., Denton, C., Hoang, L. & Rush, A. M. Structured Attention Networks. In International Conference on Learning Representations (2017). https://arxiv.org/abs/1702.00887
38. Goodman, J. Semiring Parsing. Computational Linguistics **25**, 573–606 (1999).
39. Dong, H. et al. Neural Logic Machines. In International Conference on Learning Representations (2019). https://arxiv.org/abs/1904.11694
40. Cohen, W. W. TensorLog: A Differentiable Deductive Database. Preprint at arXiv:1605.06523 (2016). https://arxiv.org/abs/1605.06523
41. Yang, F., Yang, Z. & Cohen, W. W. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In Advances in Neural Information Processing Systems (2017). https://arxiv.org/abs/1702.08367
42. Xu, K. et al. What Can Neural Networks Reason About?. In International Conference on Learning Representations (2020). https://arxiv.org/abs/1905.13211
43. Xu, K. et al. How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks. In International Conference on Learning Representations (2021). https://arxiv.org/abs/2009.11848
44. Dudzik, A. & Veličković, P. Graph Neural Networks are Dynamic Programmers. In Advances in Neural Information Processing Systems (2022). https://arxiv.org/abs/2203.15544
45. Veličković, P., Perivolaropoulos, C., Barbero, F. & Pascanu, R. Softmax is not Enough (for Sharp Size Generalisation). In International Conference on Machine Learning, 61190–61211 (2025). https://arxiv.org/abs/2410.01104
46. Krotov, D. & Hopfield, J. J. Dense Associative Memory for Pattern Recognition. In Advances in Neural Information Processing Systems, 1172–1180 (2016). https://arxiv.org/abs/1606.01164
47. Ramsauer, H. et al. Hopfield Networks is All You Need. In International Conference on Learning Representations (2021). https://arxiv.org/abs/2008.02217
48. Ho, C. H., Gutleb, T. S. & Ortner, C. Atomic Cluster Expansion without Self-Interaction. Journal of Computational Physics **515**, 113271 (2024). https://arxiv.org/abs/2401.01550
49. Liu, Y. & Lapata, M. Learning Structured Text Representations. Transactions of the Association for Computational Linguistics **6**, 63–75 (2018). https://arxiv.org/abs/1705.09207
50. Qwen Team. Qwen3 Technical Report. Preprint at arXiv:2505.09388 (2025). https://arxiv.org/abs/2505.09388
51. Segol, N. & Lipman, Y. On Universal Equivariant Set Networks. In International Conference on Learning Representations (2020). https://arxiv.org/abs/1910.02421