BUILT FIRST, KEPT: TASK SYMMETRY SELECTS WHAT A RECURRENT NETWORK REPRESENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Several different sets of features can solve one task, and a trained network holds one of them. We ask what selects that set, and whether it is kept. We study the word problem of a finite abelian group, written as modular counters, with one bit of supervision per position. A recurrent network solves it, yet in most runs one counter is nearly missing from its features. A symmetry is a relabelling of the symbols that leaves the task unchanged. We prove that updates treating alike the symbols a symmetry exchanges can raise only a feature that cannot tell them apart. At counters of equal size, a feature that ignores which symbol advanced it is built first in every run that built a feature before its symmetry broke. Rewriting the task in a new alphabet, with the counters fixed, moves the symmetry and the first feature with it. Making that feature cost more to learn does not move the choice to cost while the symmetry lasts. Whatever feature the network builds first it keeps, in almost every run that solves, so the start of training settles the choice. We prove why under an idealisation of the network. At counters of equal size, the kept feature is one the task could do without, and it takes one counter's place. Thus, what a trained network holds depends on how its task was written down.

1 INTRODUCTION

Many tasks can be solved with more than one set of internal features, and an interpretation of a trained network finds only the set that training built. One such task is state tracking, the word problem of a group or monoid, a standard test of what sequence architectures can express and learn (Merrill et al., 2024; Sarrof et al., 2024; Ebrahimi et al., 2026). It has been studied under two kinds of supervision. When trained to output the group element, as in the modular addition of mechanistic interpretability, networks compute with a few Fourier features of the group (Nanda et al., 2023; Zhong et al., 2023; Chughtai et al., 2023; Marchetti et al., 2026), and which ones they build, and in what order, varies across seeds and hyperparameters (Chughtai et al., 2023; Ding et al., 2024). Three accounts of that choice have been given: features compete for capacity (Ding et al., 2024), the choice is arbitrary (Chughtai et al., 2023), and the statistics of the input encoding set the order (Marchetti et al., 2026). When trained instead to accept or reject, networks have been read back as automata (Giles et al., 1992; Weiss et al., 2018) and tested as recognizers across the Chomsky hierarchy (Butoi et al., 2025), and the same line studies them as transducers (Delétang et al., 2022; Strobl et al., 2024). An expressivity result does not say what training reaches (Butoi et al., 2025), yet these studies do not ask which features training selects. Here we ask that question of a recognizer, trained with one bit per position: which features does training select, and are they kept?

Specifically, we study the word problem of a finite abelian group, written as modular counters: each symbol advances one counter, a no-op advances none, and a string is accepted when every counter is at zero (Figure 1(a)). A recurrent network reads the string and is told at every position whether the prefix is accepted (van Rossem & Saxe, 2025). At counters of equal size the network solves the task, yet in most runs one counter is nearly missing from its features, the modular quantities its state tracks (Figure 1(b)). That counter can be left out because the others determine it together with the number of advances so far, modulo the common size; this count ignores which counter moved, and we call it symbol-blind. More precisely, we prove that the counters and this count are the cheapest features, that leaving out any one of them leaves a set that can solve the task, and that no smaller set can (section 3). A trained network holds about that many, so its choice is which of these candidates to leave out.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

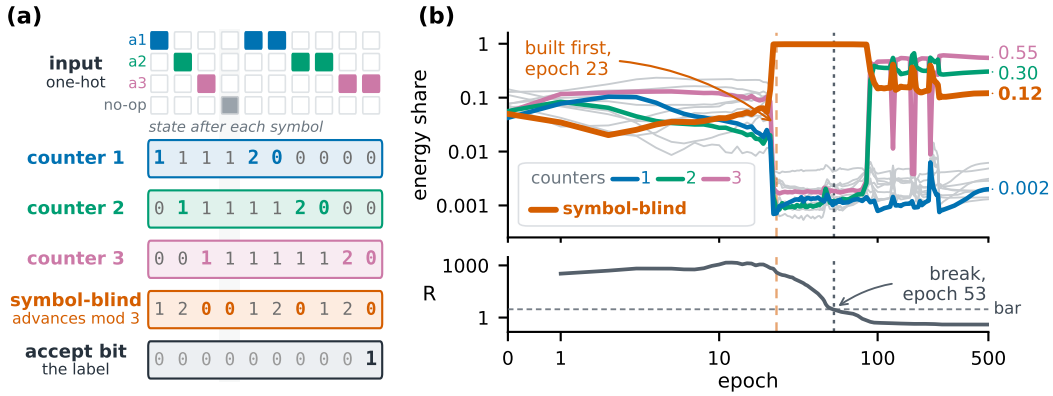


Figure 1: One string of the task, and one run trained on it. **(a)** A string at $(3, 3, 3)$, whose automaton has 27 states. Each column is one input symbol, given to the network one-hot; below it are the state after that symbol, the three counters and the symbol-blind count, and the accept bit, which is the label. **(b)** One run: each feature's share of the code's energy (section 2.3) against the training epoch, four features named and nine grey; the symbol-blind curve is (a)'s `symbol-blind` row. The rules mark the first build and the break; the strip is R, the agreement of the input columns (section 4.1), and the break is ruled where it falls below its bar.

We begin with what a network builds first, which is selected by a symmetry of the presentation, the alphabet and the distribution of strings. A symmetry is a relabelling of the symbols, from an automorphism of the group, that leaves the language and the distribution unchanged. For example, exchanging the symbols of two equal counters is one, and so is exchanging one counter's forward and backward symbols where the alphabet has both. We prove that input weights treating exchanged symbols alike stay so under gradient flow, and that meanwhile every feature the symmetry moves is capped by a factor falling geometrically with the length of the string (Lemma 3). The proof needs only a label the relabelling leaves unchanged, as the accept bit is and a label that names the state is not, and holds for any network that reads each symbol through its own input column, transformers and state-space models included. So a network whose input weights are still symmetric builds first the cheapest feature the symmetry fixes, which at counters of equal size is the symbol-blind count. Trained networks, however, start from random weights, and their updates treat exchanged symbols alike only until the **break**; when the break comes depends on the architecture and on the run (section 4.4). Still, every run of section 4 that built a feature before its break built the predicted one first, as the run in Figure 1(b) does (sections 4.2 and 4.4). The presentation also moves the choice: changing the alphabet with the counters fixed moves the first-built feature with the symmetry, while making that feature costlier does not move it before the break (section 4.3).

Whatever a network builds first, it then keeps, in almost every run that solves, and under an idealisation of the trained network we prove why (section 5). So the choice among the candidates is settled early in training, and at counters of equal size it costs a counter: the network completes a smallest set around the symbol-blind count, and in most runs one counter's own feature is the one left out. What an interpretation finds in a trained network thus depends on how its task was written down.

2 Problem Setting

2.1 The Task

Definition 1 (Product modular counter recognition)**.** Fix $k \geq 1$ and moduli $m_1, \ldots, m_k \geq 2$ that need not be equal, and let $G = \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k}$, written additively with identity 0, with j-th unit vector e_j and order $|G| = \prod_j m_j$. The **specification** is the pair $(G, \{0\})$. An alphabet is a finite set A with a shift $\sigma : A \to G$, and it gives the automaton with state set G, initial state 0, accept set $\{0\}$, and transition $c \mapsto c + \sigma(s)$ on $s \in A$, so the state after a string is the sum of its symbols' shifts. Take $A = \{a_0, a_1, \ldots, a_k\}$, with one **advance** symbol a_j per counter, $\sigma(a_j) = e_j$, and one **no-op**

symbol a_0, $\sigma(a_0) = 0$; we call this alphabet one-way, and $L(G)$ is the language of its automaton. Deciding membership in $L(G)$ from the string alone is product modular counter recognition.

$L(G)$ is the set of strings that return every counter to zero, and Figure 1(a) follows one string through the automaton. It is the word problem of the abelian group G, and it is a regular language whose minimal automaton has $|G|$ states. We call each factor $\mathbb{Z}_{m_j}$ a **counter**, of size m_j.

The supervision is one bit per position. A string $s_1 \cdots s_T$ carries one label per prefix, and no other function of the state is supervised:

$$y_t \;=\; \mathbf{1}\big[\,\sigma(s_1) + \cdots + \sigma(s_t) = 0\,\big], \qquad t = 1, \ldots, T. \tag{1}$$

Along the string this label is the accept bit of the automaton's state, so the task is state tracking with one bit of the state reported at each step. Every prefix is itself a string whose membership is decided, so the network is trained as a recognizer of $L(G)$ (Butoi et al., 2025) at every prefix. van Rossem & Saxe (2025) supervise the same pairs of prefix and label with a training set that holds every string up to a fixed length, and so every prefix of its strings; at lengths where no such set can be built we draw strings and label every prefix. The no-op is a neutral letter of $L(G)$ (Barrington et al., 2005): inserting or deleting it never changes membership, as with the 0 in the streaming parity of van Rossem & Saxe (2025). Since every prefix is labelled, a no-op is still a step with its own label, across which the network must hold its state.

2.2 The network and its code

The network reads the string one symbol at a time with a recurrent cell of width H, carrying a state

$$h_t \;=\; f_\theta(h_{t-1}, s_t) \;\in\; \mathbb{R}^H, \tag{2}$$

where θ are the cell's weights and each symbol enters as a one-hot vector, through its own column of the input weights; a linear readout of h_t gives one logit per position, trained against the label y_t. Section 4.1 names the cells, and appendix B gives the width. The label fixes what the network must compute and leaves open which functions of the state it represents. We read those functions through the network's **code**, a map $\varphi : G \to \mathbb{R}^H$ from the automaton's states to the hidden space: the mean of the hidden state over strings that end in each state.

2.3 Characters and features

For $a = (a_1, \ldots, a_k) \in G$ the **character**

$$\chi_a(c) \;=\; \prod_{j=1}^{k} \exp\!\big(2\pi i\, a_j c_j / m_j\big), \qquad c \in G, \tag{3}$$

is a homomorphism from G to the unit circle, and under pointwise multiplication the characters form the dual group $\widehat{G} \cong G$, with $\chi_0 \equiv 1$ the trivial character.

Definition 2 (Class, feature)**.** A character and its conjugate $\chi_{-a} = \overline{\chi_a}$ form a **class**, a single real character when $a = -a$. A **feature** is a class.

The **counter fundamentals** $\chi_{\pm e_j}$, one class per counter, read one counter and nothing else; they are the counters' own features of section 1. Every other class reads a combination of counters.

Every function on G decomposes over the dual, and so does the code:

$$\varphi(c) \;=\; \sum_{\chi \in \widehat{G}} \chi(c)\, v_\chi, \qquad v_{\bar{\chi}} = \overline{v_\chi}, \tag{4}$$

with one coefficient vector per character, $v_\chi = |G|^{-1} \sum_c \varphi(c)\, \overline{\chi(c)}$. We call $\Omega = \{\chi \neq \chi_0 : v_\chi \neq 0\}$ the **carried set** of the code, and we say the code, and the network, **carries** a class when its members are in Ω. A class's share is its characters' energy $\|v_\chi\|^2$ over that of all non-trivial characters; read at the checkpoints of section 4.1, the shares give curves such as those of Figure 1(b). The smallest number of generators of the dual, $d(\widehat{G})$, is k at equal moduli and 1 when the moduli are pairwise coprime; section 3 proves that a code that solves the task carries at least that many classes.

2.4 THE PRESENTATION

The word "task" has two senses here: the specification is what is to be learned, and the presentation is how it is written down.

Definition 3 (Presentation, instance)**.** The **presentation** of a specification is an alphabet and the distribution from which strings are drawn. An **instance** is a specification with a presentation.

One specification admits many presentations, and each defines a different learning problem: the presentation fixes which symbols the network reads, and how often. In the one-way alphabet each advance symbol moves one counter by one. The signed alphabet presents the same specification with $2k + 1$ symbols: a forward and a backward advance symbol per counter, with shifts e_j and $-e_j$, and the no-op; its language is again the strings that return every counter to zero.

The distribution draws symbols independently at each position, each advance symbol with probability $1/(w + k)$ and the no-op with probability $w/(w + k)$, where w is the **no-op weight**; under the signed alphabet $2k$ replaces k. The no-op thus fills a fraction $w/(w + k)$ of the positions, so w sets how long a string is for a given number of advances and leaves the language alone; it also sets which features are cheapest (section 3.2). Unless stated otherwise the alphabet is one-way and $w = 1$. A run trains on one draw of strings, a fixed number at each length in a range, and is tested on a second draw at the longest length; appendix B gives the numbers. We name an instance by its modulus vector $(m_1, \ldots, m_k)$, with signed before it and the no-op weight after it where they differ from these defaults: $(3, 3, 3)$, signed $(4, 6)$, $(3, 3, 3)$ at $w = 2$.

Definition 4 (Symmetry)**.** A **symmetry** of an instance is an automorphism of G that permutes the shifts of the advance symbols and, applied to the states and to the advance symbols together, leaves the language and the distribution over strings unchanged.

At equal moduli every permutation of the counters is a symmetry under either alphabet: it carries G to itself, fixes the accept state, and permutes symbols of equal probability. Under the one-way alphabet a symmetry can only permute counters of equal modulus, so an instance with no two moduli equal has only the identity. Under the signed alphabet, reversing one counter of modulus above 2, $c_j \mapsto -c_j$, exchanges that counter's two advance symbols and is a symmetry whatever the other moduli. We say an instance has a symmetry when it has one other than the identity.

Along a string the presentation defines one quantity the specification does not: the number of advance symbols read so far. Since a character is a homomorphism, reading a symbol s multiplies χ_a of the state by $\chi_a(\sigma(s))$, and a symmetry α acts on the characters by $\chi \mapsto \chi \circ \alpha$.

Definition 5 (Symbol-blind, protected)**.** A class is **symbol-blind** when every advance symbol multiplies it by the same root of unity, so that along a string it is a function of that number and of nothing else. A class is **protected** when every symmetry of the instance fixes each of its characters.

Every symbol-blind class is protected, since the shifts of the advance symbols generate G. Under the one-way alphabet at equal moduli, $\chi_{\pm\mathbf{1}}$ is symbol-blind, where $\mathbf{1} = (1, \ldots, 1)$: it reads the number of advances modulo the common modulus, the symbol-blind count of section 1, and appendix A gives when such a class exists. An instance can have a symbol-blind class and no symmetry, as one-way $(4, 6)$ does (section 4.3): under the one-way alphabet this happens exactly when the moduli share a divisor above 1 and no two moduli are equal. With no symmetry every class is protected.

3 WHICH FEATURE SETS SOLVE THE TASK, AND WHICH CAN RISE FIRST

3.1 WHICH SETS ARE COMPLETE

The label is the accept bit, so the network's readout must tell the accept state from every other state. We say a linear readout $u \in \mathbb{R}^H$ **separates** the accept state with margin $\gamma > 0$ when $u \cdot \varphi(0) - u \cdot \varphi(c) \geq \gamma$ for every $c \neq 0$. A code solves the instance when some readout separates the accept state with positive margin. A network whose readout labels every string correctly has such a code, since the code averages its hidden states over the strings that end in each state.

Theorem 1. *If a code solves the instance, its carried set Ω generates $\widehat{G}$, so it carries at least $d(\widehat{G})$ classes.*

If Ω does not generate, the joint kernel $\bigcap_{\chi\in\Omega}\ker\chi$ holds a state $c\neq 0$, every carried character takes the value 1 at c, and the code takes the same value at c as at 0, so no readout separates them. When the non-zero v_χ are linearly independent over $\mathbb{C}$, the converse holds: a carried set that generates admits a separating readout (appendix A, where the condition is shown equivalent to the idealisation L of section 5.2). Because $d(\widehat{G})$ is fixed by the specification, the bound is known before training.

3.2 Which sets are cheapest

Training from small weights begins with a linear stage, in which each feature grows at its own rate and the fastest is acquired first (Saxe et al., 2014; Kunin et al., 2025). At this stage the hidden state is close to a linear function of the input history, and the component of the state along a character χ_a grows at a rate set by $|\mu_a|$, where the **per-step multiplier** μ_a is the expectation of χ_a over the shift of one symbol drawn from the presentation:

$$\mu_a \;=\; \mathbb{E}\,\chi_a(\sigma(s)) \;=\; \begin{cases} \dfrac{w+\sum_{j=1}^{k}\omega_j^{a_j}}{w+k} & \text{one-way,} \\[2ex] \dfrac{w+2\sum_{j=1}^{k}\cos(2\pi a_j/m_j)}{w+2k} & \text{signed,} \end{cases} \qquad \omega_j = e^{2\pi i/m_j}. \tag{5}$$

The no-op, which moves no counter, contributes the w, and under the signed alphabet μ_a is real and can be negative. A class is cheaper the larger its $|\mu_a|$, since it then grows faster; we call this order **cost**, and the **tier** is the set of cheapest non-trivial classes. Marchetti et al. (2026) order the features a two-layer network learns by a statistic built on the Fourier coefficients of the input encoding, with every group element equally likely at every position, and under a one-hot encoding these coefficients all have modulus 1. Our input is one-hot, and μ_a is the Fourier coefficient at χ_a of another function on G, the distribution of one symbol's shift, which the alphabet and the no-op weight set.

Lemma 2 (equal moduli, $w=1$)**.** *Take the one-way alphabet with $k\geq 2$ and $w=1$, and let every modulus equal m. Then the tier is the k counter fundamentals together with the symbol-blind class $\chi_{\pm\mathbf{1}}$: $k+1$ classes, where $\widehat{G}$ needs k generators. The only relation among them is $e_1+\cdots+e_k-\mathbf{1}=0$, with every coefficient a unit of $\mathbb{Z}_m$, so every k of the $k+1$ generate $\widehat{G}$.*

At $(3,3,3)$ the tier holds the three counters' own features and the symbol-blind count, whose values Figure 1(a) follows along one string, and any three of the four can solve the task. With the one-way alphabet and $w=1$ but unequal moduli, the tier holds only the fundamentals of the counters of largest modulus, so it contains no generating set of $\widehat{G}$. At equal moduli the tier is the symbol-blind class alone for $0<w<1$ and the k fundamentals alone for $w>1$. Appendix A derives the multiplier, gives the scope of the linear stage, proves these statements, and sums the multiplier's powers over the prefixes a string supervises, an accounting under which the fundamentals are cheaper at every w.

Theorem 1 and Lemma 2 meet at equal moduli and $w=1$. A code that solves the instance carries at least $d(\widehat{G})$ classes, and the cheapest $k+1$ classes, the candidates of section 1, contain $k+1$ generating sets of exactly that size. So cost leaves several complete sets of the same size exactly tied: the symbol-blind class and a fundamental tie by an identity, and the fundamentals tie with one another by the symmetry of Definition 4. Marchetti et al. (2026) prove their ordering under the assumption that the statistic takes distinct values on the features, which places this case outside it.

3.3 Which classes can rise first

Cost ranks the classes and, at equal moduli, leaves a tie. The symmetry of the instance limits which classes can rise. The symmetries split the advance symbols into orbits, the sets of symbols they carry into one another, and we count the no-op as an orbit of its own.

Lemma 3. *Take an instance with a symmetry, and input weights whose columns agree on the symbols of each orbit. Gradient flow on the expected loss keeps the columns in agreement. While they agree, the state after a string depends only on the orbit of each of its symbols, and on strings of length T the state's correlation with a character is at most $\kappa^{T/2}$ times the root mean square of the state, for a number κ of the character and the presentation (appendix A), with $\kappa<1$ unless every symmetry fixes the character.*

We call the weights whose input columns agree within each orbit the subspace. On it, the code's coefficient on a character that a symmetry moves is at most the root mean square of the state times a factor that falls geometrically with the length of the string (appendix A), so only protected classes (Definition 5) escape this cap. A protected class cannot tell apart two symbols a symmetry exchanges. The lemma's first clause is the principle of symmetric criticality (Palais, 1979) in the form deep learning uses (Chen et al., 2023; Ziyin, 2024). The second clause adds the cap, so where costs tie, as at equal moduli (section 3.2), the symmetry decides.

The lemma's hypothesis is a loss unchanged when two exchanged symbols swap. The accept bit has it, since a symmetry fixes the accept state; a head that names the state does not, since swapping two symbols changes the state a string ends in (section 4.4). The proof needs only that a symbol enters the network through its own input column and through no other weight, so the lemma holds for a transformer or a state-space model that reads its input through an embedding (appendix A).

On the subspace, the lemma and cost (section 3.2) give a prediction in three steps. Only protected classes can rise (Lemma 3). Among them the cheapest grows fastest and, at the linear stage, is acquired first (section 3.2). So a network on the subspace builds first the cheapest protected class, and a class the symmetry excludes stays under the lemma's cap however cheap it is. Under the one-way alphabet at equal moduli every protected class is symbol-blind, and $(3, 3, 3)$ has only one, so there the lemma alone names the class; cost chooses where several classes are protected, as at signed $(4, 6)$ and $(6, 3, 3)$. A network trained from random initial weights on a finite draw starts off the subspace, where the cap weakens gradually (appendix A).

Two accounts thus name the class built first: the prediction names the cheapest protected class, and cost names the tier. They agree where the tier holds a protected class and separate where it holds none, as at $(3, 3, 3)$ at $w = 2$ and at signed $(4, 6)$, where cost names a counter's class and the prediction the symbol-blind class.

4 What the network builds first

Section 3 predicts the class built first for a network on the subspace, and our networks start off it, from random initial weights. We read which class each run builds first, and when its break comes.

4.1 Training and reading

Every run trains one of three architectures, an LSTM (Hochreiter & Schmidhuber, 1997), a GRU (Cho et al., 2014) or an Elman network with tanh units (Elman, 1990), by Adam on the cross-entropy of every label, with no weight decay. A run's seed sets only its initial weights, Xavier-normal at gain 0.1 (the standard initialisation), and its batch order. The gain is small, so training starts from small weights, as section 3.2 assumes. A run solves the instance if its accuracy on the second draw is at least 0.999 at the end of training. Counts include runs that do not solve unless stated. Appendix B tabulates the settings and lists the runs by instance and architecture, including the few trained longer, on longer strings or from larger initial weights; every count in sections 4 and 5 is over these runs.

We read the code of section 2.2 at checkpoints during training, on a fixed set of random strings (appendix B). Reading the hidden state by the automaton state has a precedent: Michalenko et al. (2019) decode the state from the hidden vector and find that the decoding merges small sets of states. In the code, two states on which every carried character agrees are read as one (section 3.1). A class is built when its share first reaches one half, which no other class can then exceed, at a checkpoint where the total energy of the non-trivial characters is above a floor. The first class a run builds is its first-built class, and appendix B gives the checkpoints and the floor.

Lemma 3 holds while the input columns of exchanged symbols agree, and random initial columns do not. At equal moduli we therefore measure whether the updates treat the advance symbols alike. With W_j the change in the column of a_j since initialisation and $\bar{W}$ their mean,

$$R \;=\; \frac{\|\bar{W}\|^2}{\frac{1}{k}\sum_{j=1}^{k} \|W_j - \bar{W}\|^2} \tag{6}$$

is infinite while the changes agree, and its chance level, for independent columns, is $1/(k-1)$. We take the break to be the first checkpoint at which R falls below 3.0 (appendix B varies this bar).

Table 1: The symbol-blind class built first, by the symmetry at the build. Runs trained as in section 4.1 that built a class, solving or not, by whether the task has a symmetry and, where it has one, whether the symmetry was still there at the build. The first row counts the runs at $(2, 4)$ and $(4, 6)$, which have a symbol-blind class and no symmetry; the other two count the runs at equal moduli. Every row sums the architectures, and appendix B splits it by instance and architecture.

the task, at the build	runs that built	built the symbol-blind class first
has no symmetry	84	0
has a symmetry, already broken	53	19
has a symmetry, still there	119	119

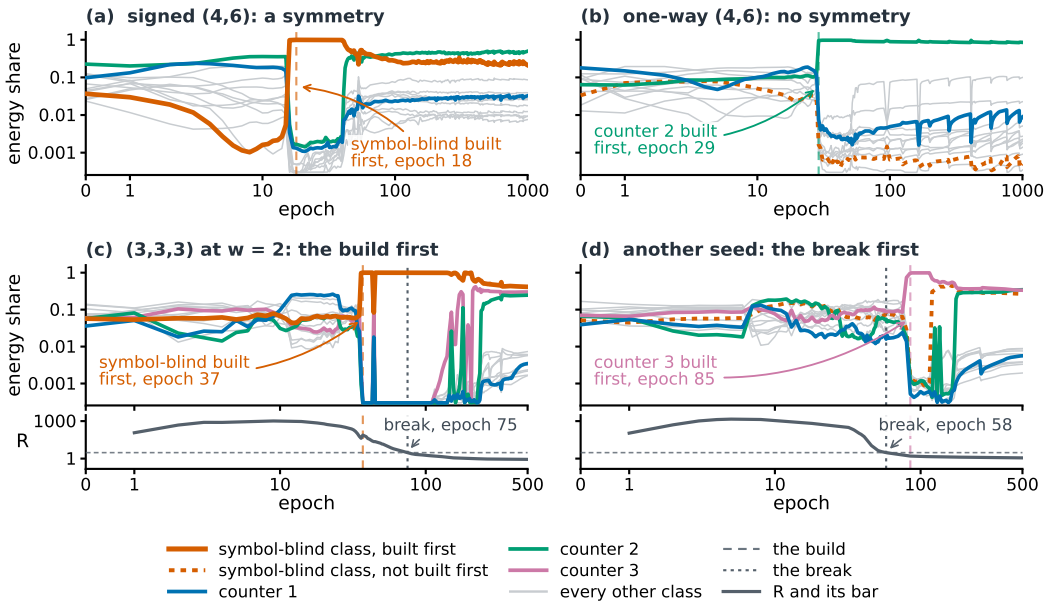


Figure 2: The alphabet, and the order of the build and the break. Four runs, drawn as in Figure 1(b): each class's share of the energy against the training epoch, with the build ruled in the colour of the class built. (a), (b): LSTM runs at signed and at one-way $(4, 6)$, trained alike for 1000 epochs; the six signed runs end at accuracy 0.9977 to 0.9989. (c), (d): two GRU runs that differ only in the seed, at $(3, 3, 3)$ at $w = 2$; the strip under each is R, and the break is ruled where R falls below its bar.

4.2 The symmetry decides while it lasts

Where the symmetry lasts until the build, the prediction of section 3.3 names the symbol-blind class at equal moduli. At the instances with equal moduli, all 119 runs whose symmetry lasted until the build built it first, including those at $w = 2$, where cost names a counter (section 4.3). Some runs whose symmetry broke first built it as well, and every run that built a counter's class first had lost the symmetry before its build (Table 1). Without a symmetry the class has no advantage: at $(2, 4)$ and $(4, 6)$, which have a symbol-blind class and no symmetry, no run built it first (Table 1, first row). Instead cost ranks the counters: a run that builds one counter's class first builds the counter of larger modulus, and the others build a class that reads two counters (appendix B).

4.3 Moving the symmetry and the cost

Each part of the presentation can be changed with the specification and the network fixed. First the alphabet, at $(4, 6)$: the one-way alphabet gives a symbol-blind class, $\chi_{\pm(2,3)}$, and no symmetry, and the signed alphabet gives a symmetry that fixes it; under neither is the class in the tier. So under the

Table 2: The first-built class at the end of training. The runs of appendix B trained as in section 4.1 that solve, by whether the instance has a symmetry and, where it has none, a symbol-blind class. One run in the first row built no class above the floor. Appendix B splits every row by instance and architecture, and gives apart the single-counter instances under the one-way alphabet, where the counter's own class is the symbol-blind class.

instance	runs that solve	first-built class kept
has a symmetry	401	399 of 400
no symmetry, has a symbol-blind class	55	53 of 55
no symmetry, no symbol-blind class	102	101 of 102

signed alphabet the prediction names the symbol-blind class and under the one-way alphabet it has no advantage; every signed run builds it first and no one-way run does (Figure 2(a), (b)).

Next the no-op weight, at $(3, 3, 3)$, where raising w from 1 to 2 leaves the counters alone in the tier (section 3.2), so cost names a counter and the prediction the symbol-blind class. The LSTM builds that class first in all but one run and the GRU in half (appendix B). Every run that built a counter first had lost the symmetry before its build (section 4.2), so before the break the higher cost does not move the choice; the architectures differ in how often the break comes first (section 4.4).

4.4 How long the symmetry lasts

A run starts off the subspace, so what it builds first turns on the order of two events, the build and the break. Figure 2(c) and (d) show both orders in two GRU runs that differ only in the seed: the build first and the symbol-blind class built, and the break first and a counter's class built. Which comes first depends on the architecture: at $(3, 3, 3)$ at $w = 2$, with every other setting shared, the GRU and the Elman network build later than the LSTM and lose the symmetry first more often (appendix B). It also depends on the label: with a head that names the state in place of the accept bit, the symmetry is gone by the first checkpoint in every run that learns every state (appendix B).

Lemma 3's conditions can also be imposed. At $(3, 3, 3)$ at $w = 2$, on the same 24 seeds, the GRU builds the symbol-blind class first in 12 runs from the standard initialisation, in 17 with the advance columns tied at initialisation, and in 22 with the training draw closed under the symmetry as well, so that the loss is exactly invariant (appendix B). Even with the draw closed the tie does not hold exactly through training; in all three settings, every run whose symmetry lasted until the build built the symbol-blind class first (appendix B).

5 What the network keeps, and what it costs

An interpretation reads a trained network, so a first-built class matters to it only if it is kept.

5.1 The first-built class is kept

We call a class lost when it ends training with a share below 0.05 and with less than half of its own peak energy (section 2.3). Of the runs that solve and built a class, all but four keep the first-built class to the end (Table 2), and section 5.2 returns to the four. It is kept at instances with a symmetry and without, whether or not it is symbol-blind (appendix B).

5.2 Why it is kept

At the end of training we read the size of the code, the number of principal directions that hold 95 per cent of its variance. In most runs it equals the dimension of the smallest generating set containing the first-built class, two for a complex class and one for a real class; at $(4, 4)$ and $(4, 6)$ most codes are larger (appendix B). We conjecture that training completes such a set around the first-built class.

Lemma 4 explains the keeping under an idealisation of the trained network. The **idealisation** L is the condition of section 3.1, that the non-zero v_χ are linearly independent over $\mathbb{C}$; under it each

carried class spans a plane of the span of $\varphi(G)$, a line for a real class (appendix A.6). Lemma 4 idealises further, holding each plane fixed while gradient flow moves the class's amplitude on it and its readout vector; a network moves all its weights, by Adam in our runs (appendix A.5).

Lemma 4. *Under L, give each carried class an amplitude and a readout vector, and let gradient flow on the logistic loss, with no weight decay, move both. For each class the squared amplitude minus the squared norm of its readout vector is constant in time. Once the loss is below $p_{\min} \log 2$, with $p_{\min}$ the least probability of a state, every state is labelled correctly, and a class can leave the code only after another class has been built that separates from the accept state the states it alone separated. If the carried set is a minimal generating set, no class can then leave, and every class keeps a positive weight in the readout as the loss falls to zero.*

In a minimal generating set every class is needed: without it the rest do not generate, and no readout separates (Theorem 1). The conserved difference is the balance that gradient flow keeps between the incoming and the outgoing weights of every unit of a feed-forward network with homogeneous activations (Du et al., 2018), here between a class's amplitude and its readout vector. The lemma adds that at zero weight decay nothing pushes a needed class toward zero, so removing one takes a second build. Under the one-way alphabet at equal moduli, a smallest generating set around the first-built class has k classes, the fewest that generate (section 2.3), so it is minimal, and under the lemma a code of that form keeps the class. Of the four losses of section 5.1, those at $(4, 6)$ are declines where the lemma's model has the class grow, though it never left the code, and the others lie outside the lemma's condition (appendix B).

5.3 What keeping it costs

At equal moduli a kept symbol-blind class takes a counter's place. At $w = 1$ the cheapest classes are the k fundamentals and the symbol-blind class, and any k of them generate (Lemma 2). A smallest generating set of these classes that contains the symbol-blind class therefore leaves out exactly one counter's fundamental. So when the symbol-blind class is built first and the code is completed around it from the cheapest classes, one counter's own feature is missing from the code, as in section 1. At $(4, 4)$ every run that solves ends this way, at $(3, 3, 3)$ most do, and at $(2, 2, 2, 2, 2)$ few do; appendix B gives the counts and the missing counter's share at the end.

6 Related work

Beyond the work cited in sections 1 and 3, two further lines bear on our results. **Modular sums in recurrent networks.** Recurrent networks trained to output a sum modulo a prime carry Fourier features (Rangamani, 2025). Trained instead to report one bit, whether a sum of three residues is zero modulo 3, recurrent networks reached one of several solutions; a phase code appeared in only a few networks, all at one activation and time constant (Murray, 2023). We give the bit at every prefix, on groups with several counters. **Transient structure.** Structure built early in training can be lost later (Singh et al., 2023) or reorganised in later stages (Hoogland et al., 2024); in our networks the first-built class is kept in almost every run that solves (section 5.1). Appendix C compares our task with that of Marchetti et al. (2026) and gives the related work in full.

7 Conclusion

When a relabelling of the symbols leaves a task unchanged, updates that treat alike the symbols it exchanges can raise only a feature that cannot tell those symbols apart (Lemma 3). At counters of equal size, a recurrent network built such a feature first in every run that built a feature before its break (section 4). Whatever it built first it kept, in almost every run that solves (section 5). At counters of equal size, keeping the symbol-blind class costs a counter: in most runs that solve, one counter's own feature is nearly missing from the code at the end (section 5.3). Our results are on abelian groups and three recurrent architectures, trained mostly at one width, one initial gain and one optimiser setting, and the keeping is proved under an idealisation (section 5.2); appendix D sets out these limits and what is open. What an interpretation finds depends on the presentation as well as on the specification: at $(4, 6)$ every run under the signed alphabet built the symbol-blind class first, and no run under the one-way alphabet did (section 4.3).

AI USE STATEMENT

We used Claude Code (Anthropic) in both the research and the writing of this paper. We conceived the research idea and designed the theoretical framework and the experiments. Claude Code helped implement part of the code, managed the project's records, and discussed the work with us as we organised our ideas. Every result the paper reports is computed by a script from the stored records of the runs. We wrote the first draft of the paper, and polished it with Claude Code's help over several rounds to improve its readability. It also translated parts of our drafts into English. It helped write the plotting scripts and format the references. In the proofs, it helped us derive some of the lemmas and check them. We supervised and checked all of this work ourselves. We take responsibility for the final content of this paper.

ETHICS STATEMENT

This work trains small recurrent networks on synthetic strings built from abelian groups; it involves no human subjects and no data about people. We see no harmful use of its results. We declare no competing interests.

REPRODUCIBILITY STATEMENT

Section 2 and appendix B.1 give the task, the network and every setting of training and reading, including how the synthetic data are drawn. Appendix A proves the statements of sections 2, 3 and 5 and states their assumptions in full. Appendix B gives the counts of sections 4 and 5 by instance and architecture and names the runs in each figure. The supplementary material holds the code and a file of every run.

REFERENCES

Emmanuel Abbe, Enric Boix-Adsera, and Theodor Misiakiewicz. SGD learning on neural networks: leap complexity and saddle-to-saddle dynamics. arXiv:2302.11055v2, updated 2023-08-31, 2023. COLT 2023.

David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989. doi: 10.1016/0022-0000(89)90037-8.

David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *Journal of Computer and System Sciences*, 70(2):101–127, 2005. doi: 10.1016/j.jcss.2004.07.004.

Nora Belrose, Quintin Pope, Lucia Quirke, Alex Mallen, and Xiaoli Fern. Neural networks learn statistics of increasing complexity. arXiv:2402.04362v3, updated 2024-10-09, 2024.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. Training neural networks as recognizers of formal languages. In *International Conference on Learning Representations (ICLR)*, 2025.

Liam Carroll, Jesse Hoogland, Matthew Farrugia-Roberts, and Daniel Murfet. Dynamics of transient structure in In-Context linear regression transformers. arXiv:2501.17745v2, updated 2025-01-31, 2025.

Feng Chen, Daniel Kunin, Atsushi Yamamura, and Surya Ganguli. Stochastic collapse: How gradient noise attracts SGD dynamics towards simpler subnetworks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014. doi: 10.3115/v1/D14-1179.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. arXiv:2302.03025, 2023. ICML 2023.

Edo Cohen-Karlik, Avichai Ben David, Nadav Cohen, and Amir Globerson. On the implicit bias of gradient descent for temporal extrapolation. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 151 of *Proceedings of Machine Learning Research*, 2022a. arXiv:2202.04302.

Edo Cohen-Karlik, Itamar Menuhin-Gruman, Raja Giryes, Nadav Cohen, and Amir Globerson. Learning low dimensional state spaces with overparameterized recurrent neural nets. arXiv:2210.14064v3, updated 2023-03-23, 2022b. ICLR 2023.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the Chomsky hierarchy. arXiv:2207.02098, 2022.

Xiaoman Delores Ding, Zifan Carl Guo, Eric J. Michaud, Ziming Liu, and Max Tegmark. Survival of the fittest representation: A case study with modular addition. arXiv:2405.17420, 2024.

Simon S. Du, Wei Hu, and Jason D. Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. arXiv:1806.00900, 2018.

M. Reza Ebrahimi, Michaël Defferrard, Sunny Panchal, and Roland Memisevic. On the "induction bias" in sequence models. In *International Conference on Machine Learning (ICML)*, 2026.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1.

Joshua Engels, Eric J. Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. Not all language model features are one-dimensionally linear. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2405.14860.

C. Lee Giles, Clifford B. Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking State-Tracking in linear RNNs through negative eigenvalues. arXiv:2411.12537v5, updated 2025-03-18, 2024. ICLR 2025.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Jesse Hoogland, George Wang, Matthew Farrugia-Roberts, Liam Carroll, Susan Wei, and Daniel Murfet. Loss landscape degeneracy and stagewise development in transformers. arXiv:2402.02364v3, updated 2025-08-01, 2024. TMLR.

Arthur Jacot, François Ged, Berfin Şimşek, Clément Hongler, and Franck Gabriel. Saddle-to-saddle dynamics in deep linear networks: Small initialization training, symmetry, and sparsity. arXiv:2106.15933, 2021.

Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116: 450–464, 1965. doi: 10.1090/s0002-9947-1965-0188316-1.

Daniel Kunin, Giovanni Luca Marchetti, Feng Chen, Dhruva Karkada, James B. Simon, Michael R. DeWeese, Surya Ganguli, and Nina Miolane. Alternating gradient flows: A theory of feature learning in two-layer neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Chenyang Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Tianyi Zhou. Fourier circuits in neural networks and transformers: A case study of modular arithmetic with multiple inputs. arXiv:2402.09469v4, 2024.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.10749.

Lorenzo Livi. Time-scale coupling between states and parameters in recurrent neural networks. arXiv:2508.12121, 2025.

Giovanni Luca Marchetti, Christopher Hillar, Danica Kragic, and Sophia Sanborn. Harmonics of learning: Universal Fourier features emerge in invariant networks. In *Proceedings of the 37th Conference on Learning Theory (COLT)*, 2024. arXiv:2312.08550.

Giovanni Luca Marchetti, Daniel Kunin, Adele Myers, Francisco Acosta, and Nina Miolane. Sequential group composition: A window into the mechanics of deep learning. In *International Conference on Machine Learning (ICML)*, 2026.

Gavin McCracken, Gabriela Moisescu-Pareja, Vincent Letourneau, Doina Precup, and Jonathan Love. Uncovering a universal abstract algorithm for modular addition in neural networks. arXiv:2505.18266v1, updated 2025-05-23, 2025.

Gautam Neelakantan Memana. A counterexample to Fourier alignment in single-neuron modular addition. arXiv:2608.04451v2, 2026.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *International Conference on Machine Learning (ICML)*, 2024.

Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. arXiv:1902.10297v1, updated 2019-02-27, 2019. ICLR 2019.

Keith T. Murray. Phase codes emerge in recurrent neural networks optimized for modular arithmetic. arXiv:2310.07908v3, updated 2025-11-30, 2023.

Edi Muškardin, Martin Tappler, Ingo Pill, Bernhard K. Aichernig, and Thomas Pock. On the relationship between RNN hidden state vectors and semantic ground truth. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. arXiv:2306.16854.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. arXiv:2301.05217, 2023.

Oskar Nordenfors, Fredrik Ohlsson, and Axel Flinth. Optimization dynamics of equivariant and augmented neural networks. arXiv:2303.13458, 2023.

Franz Nowak, Ryan Cotterell, and Reda Boumasmoud. An algebraic view of the expressivity of recurrent language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2606.01765.

Richard S. Palais. The principle of symmetric criticality. *Communications in Mathematical Physics*, 69(1):19–30, 1979.

Alexandra Maria Proca, Clémentine Carla Juliette Dominé, Murray Shanahan, and Pedro A. M. Mediano. Learning dynamics in linear recurrent neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 49860–49902, 2025.

Akshay Rangamani. Low rank and sparse fourier structure in recurrent networks trained on modular addition. arXiv:2503.22059v1, updated 2025-03-28, 2025. ICASSP 2025.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The expressive capacity of state space models: A formal language perspective. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv:1312.6120, 2014.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116 (23):11537–11546, 2019. doi: 10.1073/pnas.1820226116.

Mehran Shakerinava, Behnoush Khavari, Siamak Ravanbakhsh, and Sarath Chandar. The expressive limits of diagonal SSMs for State-Tracking. arXiv:2603.01959v2, updated 2026-08-14, 2026. ICLR 2026.

Pratyaksh Sharma, Alexandra M. Proca, Lucas Prieto, and Pedro A. M. Mediano. Temporal superposition and feature geometry of RNNs under memory demands. In *International Conference on Learning Representations (ICLR)*, 2026.

Aaditya K. Singh, Stephanie C. Y. Chan, Ted Moskovitz, Erin Grant, Andrew M. Saxe, and Felix Hill. The transient nature of emergent In-Context learning in transformers. arXiv:2311.08360v3, updated 2023-12-11, 2023.

Yonatan Slutzky, Yotam Alexander, Noam Razin, and Nadav Cohen. The implicit bias of structured state space models can be poisoned with clean labels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2410.10473.

Lena Strobl, Dana Angluin, David Chiang, Jonathan Rawski, and Ashish Sabharwal. Transformers as transducers. arXiv:2404.02040, 2024.

Aleksandar Terzić, Michael Hersche, Giacomo Camposampiero, Thomas Hofmann, Abu Sebastian, and Abbas Rahimi. On the expressiveness and length generalization of selective state-space models on regular languages. arXiv:2412.19350v2, 2024.

Yuandong Tian. Provable scaling laws of feature emergence from learning dynamics of grokking. arXiv:2509.21519v5, 2025.

Loek van Rossem and Andrew M. Saxe. Algorithm development in neural networks: Insights from the streaming parity task. In *International Conference on Machine Learning (ICML)*, 2025.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning (ICML)*, 2018.

Junjie Yao, Liangkai Hang, and Zhi-Qin John Xu. Context staircase: Signature-aligned dynamics of token embeddings under small initialization. arXiv:2608.30315v1, 2026.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Liu Ziyin. Symmetry induces structure and constraint of learning. In *International Conference on Machine Learning (ICML)*, 2024.

Appendix A proves the statements of sections 2, 3 and 5. Appendix B gives the training and reading details of section 4.1, sorts the instances into five cases, gives the counts of sections 4 and 5 by instance and architecture, and names the runs the figures draw. Appendix C compares our task with the closest one and gives the related work in full. Appendix D sets out the limits of our results and what is open.

A PROOFS

This appendix proves the statements of sections 2, 3 and 5 in the order the body states them.

Notation. The group is $G = \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k}$, a state is written c, and e_j is the j-th unit vector; $\sigma(s)$ is the shift of a symbol s, a_0 is the no-op and w its weight (sections 2.1 and 2.4). The characters χ_a of equation 3 form $\widehat{G}$, which we write additively in a where that is shorter: e_j then also names counter j's fundamental, and $\mathbf{1} = (1, \ldots, 1)$ the symbol-blind character of the one-way alphabet at equal moduli. χ_0 is the trivial character, $\omega_j = e^{2\pi i/m_j}$, μ_a is the multiplier of equation 5, and $d(\widehat{G})$ is the least number of generators of $\widehat{G}$. A code φ has coefficient vectors v_χ (equation 4), carried set Ω and span V, the real span of $\varphi(G)$ in $\mathbb{R}^H$; L is the idealisation of section 5.2. Γ is the group of symmetries of an instance (Definition 4), u_s is the input column of symbol s, and h_T and c_T are the network's state and the automaton's state after a string of length T.

A.1 WHEN A SYMBOL-BLIND CLASS EXISTS

Lemma 5. *Under the one-way alphabet a non-trivial symbol-blind class exists exactly when the greatest common divisor g of the moduli exceeds 1. The symbol-blind characters are then the χ_a with $a_j = t\,m_j/g$ for $t = 1, \ldots, g-1$; at equal moduli they are the $\chi_{t\mathbf{1}}$, among them $\chi_{\pm\mathbf{1}}$. Under the signed alphabet a non-trivial symbol-blind class exists exactly when every modulus is even, and it is the one real class with $a_j = m_j/2$ for every j.*

Proof. The no-op multiplies every character by 1, so χ_a is symbol-blind when $\chi_a(\sigma(s))$ is one root of unity ζ for every advance symbol s. If $\zeta = 1$ the character is trivial on the shifts, which generate G, so it is trivial. Under the one-way alphabet $\chi_a(e_j) = \exp(2\pi i a_j/m_j)$, and the condition is $a_j/m_j \equiv r \pmod 1$ for one r and every j. Then $m_j r$ is an integer for every j, so the order of r in $\mathbb{Q}/\mathbb{Z}$ divides g; conversely every $r = t/g$ gives the integers $a_j = t\,m_j/g$. A choice with $r \not\equiv 0$ exists exactly when $g > 1$. Under the signed alphabet the two advance symbols of counter j give $\chi_a(e_j)$ and its conjugate, so ζ is real, and $\zeta \neq 1$ forces $\zeta = -1$, that is $a_j/m_j = 1/2$ for every j. □

Under the one-way alphabet a symmetry can exchange only counters of equal modulus, so an instance has a symbol-blind class and no symmetry exactly when its moduli share a divisor above 1 and no two are equal, as section 2.4 states. At $(4, 6)$ the symbol-blind class is $\chi_{\pm(2,3)}$ under both alphabets.

A.2 THEOREM 1 AND ITS CONVERSE

The proof of Theorem 1 in section 3.1 uses one standard fact of finite abelian groups: for a subgroup S of $\widehat{G}$, the states at which every character of S takes the value 1 form a subgroup of order $|G|/|S|$. So the joint kernel of Ω is trivial exactly when Ω generates $\widehat{G}$. A class and either of its characters generate the same subgroup, so a carried set that generates holds at least $d(\widehat{G})$ classes.

Lemma 6. *The non-zero coefficient vectors v_χ, $\chi \in \widehat{G}$, of a code are linearly independent over $\mathbb{C}$ exactly when, for each j, a linear map F_j on the span V of the code has $F_j\varphi(c) = \varphi(c + e_j)$ for every state c. Under L, if the carried set Ω generates $\widehat{G}$, some linear readout separates the accept state with margin at least $1 - \cos(2\pi/e)$, where e is the largest order of an element of G.*

Proof. Suppose the maps F_j exist. Since $\varphi(G)$ spans V and $F_j^{m_j}\varphi(c) = \varphi(c)$, each F_j satisfies $F_j^{m_j} = I$ on V. Extend the F_j complex-linearly to $V_{\mathbb{C}} = V + iV$, which holds every v_χ. There

each F_j has a minimal polynomial dividing $x^{m_j}-1$, whose roots are distinct, so it is diagonalisable, and maps that commute and are each diagonalisable have a common eigenbasis. A joint eigenvalue $(\lambda_1,\dots,\lambda_k)$ has $\lambda_j^{m_j}=1$, so it is $(\chi(e_1),\dots,\chi(e_k))$ for exactly one character χ; write V_χ for its joint eigenspace and decompose $\varphi(0)=\sum_\chi w_\chi$ with $w_\chi\in V_\chi$. Then $\varphi(c)=F_1^{c_1}\cdots F_k^{c_k}\varphi(0)=\sum_\chi \chi(c)\,w_\chi$, and, the characters being orthogonal, $|G|^{-1}\sum_c \varphi(c)\,\overline{\chi(c)}=w_\chi$ (Fourier inversion on G), that is $w_\chi=v_\chi$. Non-zero vectors in distinct joint eigenspaces are linearly independent. Conversely, if the non-zero v_χ are linearly independent, set $F_j v_\chi=\chi(e_j)\,v_\chi$ on their span, which is $V_{\mathbb{C}}$, since each $\varphi(c)$ is a combination of the v_χ and each v_χ of the $\varphi(c)$. It carries $\overline{v_\chi}=v_{\bar\chi}$ to the conjugate of $F_j v_\chi$, so it restricts to a real map on V; $F_j\varphi(c)=\sum_\chi \chi(c+e_j)\,v_\chi=\varphi(c+e_j)$, the maps commute, and $F_j^{m_j}=I$.

For the separating readout, the real and imaginary parts of v_χ over one character of each complex carried class, with v_χ for each real one, are then linearly independent over $\mathbb{R}$. So some real u has $u\cdot v_\chi=1$ for one character of each carried class, and then for its conjugate as well, since $u\cdot v_{\bar\chi}$ is the conjugate of $u\cdot v_\chi$. For every state c,

$$u\cdot\varphi(0)-u\cdot\varphi(c)\;=\;\sum_{\chi\in\Omega}\big(1-\chi(c)\big)\;=\;\sum_{\chi\in\Omega}\big(1-\operatorname{Re}\chi(c)\big), \tag{7}$$

because Ω is closed under conjugation. Every term is at least 0. For $c\neq 0$ some $\chi\in\Omega$ has $\chi(c)\neq 1$, since the joint kernel is trivial, and $\chi(c)$ is a root of unity of order dividing e, so that term is at least $1-\cos(2\pi/e)$. □

Without L the converse fails. At $G=\mathbb{Z}_3$ take the one-dimensional code $\varphi(c)=\sin(2\pi c/3)$. It carries the one non-trivial class, which generates $\widehat{G}$, and $\varphi(0)=0$ lies between $\varphi(1)=\sqrt{3}/2$ and $\varphi(2)=-\sqrt{3}/2$, so no linear readout separates the accept state. Its coefficient vectors $v_{\chi_1}=-i/2$ and $v_{\chi_2}=i/2$ are dependent in $\mathbb{C}^1$.

A.3 The multiplier, the tier and Lemma 2

The multiplier. The presentation draws the symbols of a string independently, so for a string of length t the state c_t is a sum of t independent shifts and $\mathbb{E}\,\chi_a(c_t)=\mu_a^t$, where $\mu_a=\mathbb{E}\,\chi_a(\sigma(s))$ over one drawn symbol s. This is equation 5. By the orthogonality of characters the label is $y_t=|G|^{-1}\sum_\chi\chi(c_t)$, so for the symbol s_τ at any one position $\tau\le t$,

$$\mathbb{E}\big[\,y_t\mid s_\tau\,\big]\;=\;\frac{1}{|G|}\sum_{\chi\in\widehat{G}}\chi(\sigma(s_\tau))\,\mu_\chi^{t-1}. \tag{8}$$

While the weights are small the network's logit at position t is close to a sum of one term per symbol read. Among such sums the best in mean square is $\mathbb{E}\,y_t+\sum_{\tau\le t}\big(\mathbb{E}[y_t\mid s_\tau]-\mathbb{E}\,y_t\big)$. The positions are independent, so functions of different positions with mean zero are orthogonal, and the projection of y_t onto the sums is the constant $\mathbb{E}\,y_t$ plus, for each position τ, the projection onto the functions of s_τ with mean zero, which is $\mathbb{E}[y_t\mid s_\tau]-\mathbb{E}\,y_t$. Through each character every position carries the factor μ_χ^{t-1}, the multiplier raised to the number of the other positions. This is the input-output correlation along which the linear stage moves the weights (Saxe et al., 2014), and the sense in which $|\mu_a|$ sets a class's rate in section 3.2. The reading needs the logit to be additive over positions, which holds at small weights; the multiplier and the tier are fixed by the presentation.

Summed over the prefixes a string supervises, the factor becomes $\sum_t\mu_a^{t-1}$, which tends to $(1-\mu_a)^{-1}$ over long strings. At equal moduli $1-\mu_{e_j}=(1-\omega)/(w+k)$ and $1-\mu_{\mathbf{1}}=k(1-\omega)/(w+k)$, so in that limit each fundamental's sum has k times the modulus of the symbol-blind class's at every no-op weight, and the fundamentals tie. The runs of section 4.2 with equal moduli build the symbol-blind class first against this factor wherever the symmetry lasts until the build.

Lemma 7. *Take the one-way alphabet with no-op weight $w>0$. (i) At equal moduli the tier is the symbol-blind class $\chi_{\pm\mathbf{1}}$ for $w<1$, the k counter fundamentals for $w>1$, and both at $w=1$. (ii) At $w=1$ with unequal moduli the tier is the fundamentals of the counters of largest modulus, and it contains no generating set of $\widehat{G}$.*

Proof. Write $\phi_j = 2\pi a_j/m_j$. Expanding the square,

$$\Big|w+\sum_{j=1}^{k} e^{i\phi_j}\Big|^2 = (w+k)^2 - 2\,C_w(a), \qquad C_w(a) = w\sum_j \big(1-\cos\phi_j\big) + \sum_{i<j}\big(1-\cos(\phi_i-\phi_j)\big), \tag{9}$$

so the tier is the set of non-trivial a that minimise C_w. Let M be the largest modulus, $c_M = 1-\cos(2\pi/M)$, and d the number of counters with $a_j \neq 0$. Each of these has $1-\cos\phi_j \geq 1-\cos(2\pi/m_j) \geq c_M$, with equality exactly when $m_j = M$ and $a_j = \pm 1$. Keeping in the second sum only the pairs of a counter with $a_j \neq 0$ and one with $a_i = 0$ gives

$$C_w(a) \;\geq\; (w+k-d)\sum_{j:\,a_j\neq 0}\big(1-\cos\phi_j\big) \;\geq\; (w+k-d)\,d\,c_M. \tag{10}$$

A fundamental $\pm e_j$ with $m_j = M$ costs $(w+k-1)\,c_M$, and every other a with $d=1$ costs more. For $1<d<k$ the bound exceeds $(w+k-1)\,c_M$ by $(d-1)(w+k-d-1)\,c_M > 0$. For $d=k$, $C_w(a) \geq w\sum_j(1-\cos\phi_j) \geq w\,k\,c_M$, with equality exactly when every modulus is M, every $a_j = \pm 1$ and every pair term vanishes, which forces $a = \pm\mathbf{1}$: two counters of opposite sign cost $1-\cos(4\pi/M) > 0$ when $M \geq 3$, and at $M=2$ the signs agree.

(i) At equal moduli the candidates are the fundamentals, at $(w+k-1)\,c_M$, and $\chi_{\pm\mathbf{1}}$, at $w\,k\,c_M$; the first exceeds the second by $(k-1)(1-w)\,c_M$, which is positive, zero or negative as w is below, at or above 1. (ii) At unequal moduli no a with $d=k$ reaches $w\,k\,c_M = k\,c_M$, so the tier is the fundamentals $\pm e_j$ with $m_j = M$. Every character they generate takes the value 1 at e_i for each counter i of smaller modulus, so they do not generate $\widehat{G}$. □

Lemma 2 takes the one-way alphabet with $k \geq 2$, $w = 1$ and every modulus equal to m, and states three things: the tier is the k counter fundamentals and $\chi_{\pm\mathbf{1}}$; the only relation among them is $e_1 + \cdots + e_k - \mathbf{1} = 0$; and every k of these $k+1$ classes generate $\widehat{G}$, which needs k generators.

Proof of Lemma 2. The tier is part (i) of Lemma 7 at $w=1$. In $\widehat{G}$, written additively, $\mathbf{1} = e_1+\cdots+e_k$. Dropping $\mathbf{1}$ leaves $e_1,\ldots,e_k$, which generate; dropping e_j leaves $\mathbf{1}$ and the other fundamentals, and $e_j = \mathbf{1} - \sum_{i\neq j} e_i$. A relation $\lambda\mathbf{1} + \sum_j \lambda_j e_j = 0$ forces $\lambda_j = -\lambda$ for every j, so every relation is a multiple of this one. For a prime p dividing m, $\widehat{G}/p\widehat{G}$ is a vector space of dimension k over $\mathbb{Z}_p$, so fewer than k elements do not generate $\widehat{G}$. □

A.4 Lemma 3 in full, and the cap

Setting. A symmetry α acts on the advance symbols by $\sigma(\alpha\cdot s) = \alpha(\sigma(s))$, which is well defined because the advance symbols have distinct shifts; they do under both alphabets unless a signed counter has modulus 2, and no instance here has one. Write $\mathcal{O}$ for the orbits of Γ on the advance symbols and count the no-op as an orbit of its own. Let the symbols be drawn independently from a distribution p that gives the symbols of each orbit the same probability. The presentation does, since a symmetry leaves it unchanged (Definition 4), and so do the random strings of section 4.1, whose symbols are uniform. Write $p(O)$ for the probability of a symbol of O. For a character χ and an orbit O,

$$\hat{\chi}(O) \;=\; \frac{1}{|O|}\sum_{s\in O}\chi(\sigma(s)), \qquad \hat{\chi}(\{a_0\}) \;=\; 1, \qquad \kappa_\chi \;=\; \sum_{O\in\mathcal{O}} p(O)\,|\hat{\chi}(O)|^2. \tag{11}$$

Under the signed alphabet, when the reversal of counter j is a symmetry and no other symmetry moves that counter's symbols, its two advance symbols form one orbit, on which $\hat{\chi}_a = \cos(2\pi a_j/m_j)$; at equal moduli under the one-way alphabet the advance symbols form one orbit, on which $\hat{\chi}_a = k^{-1}\sum_j \omega_j^{a_j}$. The network reads the symbol at each position only through its column u_s of the input weights: the state after a string is $h_T = F(u_{s_1},\ldots,u_{s_T};\theta')$, and no weight in θ' depends on the symbol. The LSTM, the GRU and the Elman network with one-hot input have this form, with F the unrolled cell, and for the LSTM h_T is the pair of hidden and cell state. So do a causal transformer whose symbols enter through an embedding matrix, with learned positions, and a state-space model whose transition and input at each step are read from the symbol's column. The columns agree on $S_\Gamma = \{\theta : u_s = u_{s'}$ whenever s and s' share an orbit$\}$.

The statement in full. Let the instance have a symmetry. (i) Gradient flow on the expected loss leaves S_Γ invariant, and so does Adam started from zero moments. (ii) On S_Γ the state after a string depends on the string only through the orbits of its symbols. (iii) On S_Γ, for strings of length T and every character χ,

$$\left\|\mathbb{E}\left[h_T\,\overline{\chi(c_T)}\right]\right\|^2 \;\le\; \kappa_\chi^T\,\mathbb{E}\,\|h_T\|^2. \tag{12}$$

(iv) $\kappa_\chi = 1$ if every symmetry fixes χ, and $\kappa_\chi < 1$ otherwise.

Proof. (i) Write $\theta = (U, \theta')$, with U the input columns. A symmetry α acts on the weights by $P_\alpha\theta = (U', \theta')$, where $u'_{\alpha\cdot s} = u_s$ moves the column of each symbol s to the place of $\alpha \cdot s$, and on a string $x = s_1 \cdots s_T$ by $\alpha(x) = (\alpha \cdot s_1)\cdots(\alpha \cdot s_T)$, with $\alpha \cdot a_0 = a_0$. As the Setting says, the network reads each symbol only through its column; with weights $P_\alpha\theta$ it reads $\alpha \cdot s_t$ through $u'_{\alpha\cdot s_t} = u_{s_t}$, so on $\alpha(x)$ it computes exactly what the network with weights θ computes on x. The distribution of strings is unchanged, since $p(\alpha \cdot s) = p(s)$, and so are the labels, since the state of each prefix of $\alpha(x)$ is α of the state of that prefix of x and α sends only 0 to 0. So the expected loss $\mathcal{L}$ satisfies

$$\mathcal{L}(P_\alpha\theta) \;=\; \mathcal{L}(\theta) \qquad \text{for every } \theta \text{ and every } \alpha \in \Gamma. \tag{13}$$

P_α permutes coordinates, so it is orthogonal, and differentiating gives

$$\nabla\mathcal{L}(P_\alpha\theta) \;=\; P_\alpha\,\nabla\mathcal{L}(\theta). \tag{14}$$

The weights that every P_α fixes are those whose columns agree on each orbit, that is S_Γ, a linear subspace. At $\theta \in S_\Gamma$ the display gives $P_\alpha\nabla\mathcal{L}(\theta) = \nabla\mathcal{L}(\theta)$ for every α, so the gradient lies in S_Γ, and the flow $\dot\theta = -\nabla\mathcal{L}(\theta)$ restricted to S_Γ has a solution that stays in it. When the network is smooth in its weights, as the cells of section 4.1 are, and the loss is the logistic loss, $\nabla\mathcal{L}$ is locally Lipschitz, the flow from each point is unique, and so it is that solution. Adam keeps two moments per coordinate, each an average of that coordinate's past gradients, and moves each coordinate by a function of its own two moments, so its update commutes with every P_α. If the iterate and both moments are fixed by every P_α, then so is the gradient, and so are the next moments and the next iterate; zero moments are fixed, so from a point of S_Γ Adam's iterates stay in S_Γ. (ii) On S_Γ, u_s depends on s only through its orbit, and h_T depends on the string only through $u_{s_1}, \ldots, u_{s_T}$. (iii) Condition on the orbit sequence $\varpi = (O(1), \ldots, O(T))$. Given ϖ the symbols are independent and each is uniform on its orbit, so, χ being a homomorphism, $\mathbb{E}[\chi(c_T) \mid \varpi] = \prod_t \hat\chi(O(t))$. By (ii), $h_T = \hat h(\varpi)$, so $\mathbb{E}[h_T\,\overline{\chi(c_T)}] = \mathbb{E}[\hat h(\varpi)\,Z(\varpi)]$ with $Z(\varpi) = \prod_t \overline{\hat\chi(O(t))}$. The Cauchy-Schwarz inequality, coordinate by coordinate, gives $\|\mathbb{E}[\hat h Z]\|^2 \le \mathbb{E}\|\hat h\|^2\,\mathbb{E}|Z|^2$, and the orbits are independent, each O drawn with probability $p(O)$, so $\mathbb{E}|Z|^2 = \kappa_\chi^T$. (iv) $|\hat\chi(O)| \le 1$, with equality exactly when $\chi \circ \sigma$ is constant on O, and every orbit has positive probability, so $\kappa_\chi = 1$ exactly when $\chi \circ \sigma$ is constant on every orbit, that is when $\chi(\alpha(\sigma(s))) = \chi(\sigma(s))$ for every $\alpha \in \Gamma$ and advance symbol s. The shifts generate G, so this is $\chi \circ \alpha = \chi$ for every α. □

The code. Section 2.2's code on strings of length T is $\varphi(c) = \mathbb{E}[h_T \mid c_T = c]$, whose coefficient is $v_\chi = \mathbb{E}\left[h_T\,\overline{\chi(c_T)}\,r(c_T)\right]$ with $r(c) = 1/(|G|\,\mathbb{P}(c_T = c))$. Expanding the distribution of c_T over the dual, $|G|\,\mathbb{P}(c_T = c) = 1 + \sum_{\psi\neq\chi_0} \overline{\psi(c)}\,\mu_\psi^T$ with μ_ψ the multiplier under p, so $|r(c) - 1| \le E_T/(1 - E_T)$ for $E_T = \sum_{\psi\neq\chi_0} |\mu_\psi|^T$ when $E_T < 1$, and

$$\|v_\chi\| \;\le\; \left(\mathbb{E}\,\|h_T\|^2\right)^{1/2}\left(\kappa_\chi^{T/2} + \frac{E_T}{1 - E_T}\right). \tag{15}$$

The second term is the non-uniformity of the end state, the same for every network and every character.

Off the subspace. Let the weights be arbitrary, write $\bar u_O$ for the mean of the columns of an orbit O and $d_s = u_s - \bar u_{O(s)}$, so that $d_s = 0$ for the no-op. For a set K of positions let h_T^K be the state after the string when each position in K reads its own column and every other position reads its orbit's mean, and for every set J of positions let $g_J = \sum_{K\subseteq J}(-1)^{|J|-|K|}\,h_T^K$. By Möbius inversion on the subsets of $\{1, \ldots, T\}$, $h_T^K = \sum_{J\subseteq K} g_J$ for every K, and at $K = \{1, \ldots, T\}$ this is $h_T = \sum_J g_J$. The term $g_\emptyset = h_T^\emptyset$ is the state of the network whose columns are replaced by their orbit means. Each h_T^K reads its string's own columns only at the positions of K, so it depends on the string only

Table 3: The cap on the random strings. Lemma 3's cap on the random strings of section 4.1 (length 40, symbols uniform over the alphabet): the symmetries of the instance, the largest κ over the characters some symmetry moves, and the bound on such a character's energy in the code, as a fraction of the state's mean square, $(\kappa^{T/2} + E_T/(1 - E_T))^2$. A class with two characters has twice the bound. The distribution of the random strings does not depend on w, so each row holds at every w, and $(3, 3, 3)$ at $w = 2$ has the row of $(3, 3, 3)$. At $(6, 3, 3)$ the bound is set by E_T, the slow mixing of the counter of modulus 6; the symmetry's term, κ^T, is 6.8×10^{-9}.

instance	symmetries	largest κ	energy bound
signed (4)	2	0.333	8.2×10^{-20}
signed (8)	2	0.667	4.1×10^{-7}
signed $(4, 6)$	4	0.700	1.1×10^{-6}
$(3, 3)$	2	0.500	9.1×10^{-13}
$(4, 4)$	2	0.667	1.2×10^{-7}
$(3, 3, 3)$	6	0.500	2.2×10^{-12}
$(2, 2, 2, 2)$	24	0.400	3.1×10^{-16}
$(2, 2, 2, 2, 2)$	120	0.467	6.1×10^{-13}
$(6, 3, 3)$	2	0.625	1.1×10^{-3}

through its orbits and its symbols at those positions; every K in the sum for g_J lies in J, so g_J depends on the string only through its orbits and its symbols at the positions of J.

Lemma 8. *For every character χ,*

$$\left\| \mathbb{E}\left[h_T \, \overline{\chi(c_T)} \right] \right\| \;\le\; \sum_{J \subseteq \{1,\dots,T\}} \kappa_\chi^{(T-|J|)/2} \left(\mathbb{E}\, \|g_J\|^2 \right)^{1/2}. \tag{16}$$

Proof. Given the orbit sequence ϖ, the symbols at the positions outside J are independent of g_J and uniform on their orbits, so $\mathbb{E}[g_J \, \overline{\chi(c_T)} \mid \varpi] = \beta_J(\varpi) \prod_{t \notin J} \overline{\hat{\chi}(O(t))}$ with $\beta_J(\varpi) = \mathbb{E}[g_J \prod_{t \in J} \overline{\chi(\sigma(s_t))} \mid \varpi]$, and $\|\beta_J(\varpi)\|^2 \le \mathbb{E}[\|g_J\|^2 \mid \varpi]$. The Cauchy-Schwarz inequality, as in (iii), bounds the norm of the expectation over ϖ by $(\mathbb{E}\|g_J\|^2)^{1/2} \, \kappa_\chi^{(T-|J|)/2}$, and the sum over J gives the claim. □

The term of $J = \emptyset$ is the cap of (iii) for the network whose columns agree. A set of j positions carries the factor $\kappa_\chi^{(T-j)/2}$ of the other $T - j$, so the part of the state that the disagreement changes at a few positions is still capped. For a state that is a sum of one term per position, as for a linear recurrence at the small weights of section 3.2, $g_J = 0$ whenever J holds two positions or more, and the cap keeps its rate off the subspace, with a prefactor linear in the deviations d_s and in T.

The cap on the random strings. Table 3 evaluates this bound on the random strings of section 4.1, of length $T = 40$ with symbols uniform over the alphabet, at every instance with a symmetry that section 4 or Table 2 counts. A class is built when its share reaches one half, so its energy is then at least half the code's non-trivial energy; a class has at most two characters. On S_Γ, then, a class some symmetry moves can be built only while the code's whole non-trivial energy is below four times its row's bound, as a fraction of $\mathbb{E}\, \|h_T\|^2$.

Numerical check. At 8 instances with a symmetry we enumerated every string of length 7 to 10 and computed each expectation exactly under the presentation's distribution, for LSTM, GRU and Elman cells of width 16 in double precision at initial gains from 0.1 to 3, each with every orbit's columns replaced by their mean. None of the 144 reads exceeds the cap on a moved character; the largest ratio to the cap is 0.75. Under the signed alphabet the two-dimensional linear cell $z_t = \hat{\chi}(O(t))\, z_{t-1}$, which reads the symbols of an orbit alike, attains the cap: its ratio to it is 1.000000. The state $h_T = (\operatorname{Re} \chi_{e_1}(c_T), \operatorname{Im} \chi_{e_1}(c_T))$, which tracks counter 1 as L allows and tells apart the symbols of an orbit, exceeds the cap on a moved character by a factor of 30 to 29525 on the same strings, at the 7 instances where a symmetry moves counter 1's fundamental; the cap comes from the agreement of the columns.

A.5 THE IDEALISATION L

Section 5.2 states L as the condition of section 3.1, that the non-zero v_χ are linearly independent over $\mathbb{C}$. By Lemma 6 it holds exactly when moving every state by e_j extends to a linear map F_j on the span V of the code, $F_j\varphi(c) = \varphi(c + e_j)$, for $j = 1, \ldots, k$. The maps commute and each has finite order. Since $\varphi(G)$ spans V, $F_j^{m_j}\varphi(c) = \varphi(c + m_j e_j) = \varphi(c)$ and $F_iF_j\varphi(c) = \varphi(c + e_i + e_j) = F_jF_i\varphi(c)$ for every state c, so $F_j^{m_j} = I$ and the maps commute.

A code whose $|G|$ points are linearly independent meets L, since a map given on a basis extends linearly. At every instance of section 4 the automaton has fewer states than the network of section 4.1 has hidden units, so a code in general position meets L; the codes that fail it are those whose coefficient vectors of distinct characters are linearly dependent, like the code of appendix A.2, whose accept state lies between two reject states on one line.

Lemma 4 idealises further. It holds each carried class on a fixed plane, a line for a real class, and lets gradient flow move only the class's amplitude and its readout vector (appendix A.6), where a network moves all its weights and the runs use Adam. Its loss is the runs' loss, the logistic loss of the accept bit with no weight decay (section 4.1). The size count of section 5.2, two dimensions for each complex class and one for a real class, is the size of a code built from whole planes.

A.6 LEMMA 4, AND WEIGHT DECAY

Under L, Lemma 4 gives each carried class an amplitude and a readout vector, moved by gradient flow on the logistic loss with no weight decay, and states three things: for each class the squared amplitude minus the squared norm of the readout vector is constant; once the loss is below $p_{\min} \log 2$, with $p_{\min}$ the least probability of a state, every state is labelled correctly and a class can leave the code only after another built class separates the states it alone separated; and if the carried set is a minimal generating set, no class leaves, and every class keeps a positive weight as the loss falls to zero.

The model. Under L the carried classes lie in linearly independent subspaces of V (Lemma 6), a plane for a complex class and a line for a real one. For a carried class pick one of its characters χ, and write $z_\chi(c) = (\operatorname{Re}\chi(c), \operatorname{Im}\chi(c))$ for a complex class and $z_\chi(c) = \chi(c)$ for a real one. Lemma 4 keeps each subspace fixed and gives the class an amplitude a_χ and a readout vector r_χ, so that the logit at state c is

$$f(c) \;=\; \sum_\chi a_\chi \,\langle r_\chi, z_\chi(c)\rangle + b. \tag{17}$$

With $y_c = +1$ at the accept state and -1 elsewhere, and $p_c > 0$ the probability of state c among the supervised prefixes, the loss is $\mathcal{L} = \sum_c p_c \log(1 + e^{-y_c f(c)})$, and gradient flow with weight decay $\eta \geq 0$ moves the a_χ, the r_χ and b on $\mathcal{L} + \frac{\eta}{2}\sum_\chi(a_\chi^2 + \|r_\chi\|^2)$. Lemma 4 is the case $\eta = 0$. With $\pi_c = p_c/(1 + e^{y_c f(c)})$ and $\nu_\chi = \sum_c \pi_c\, y_c\, z_\chi(c)$,

$$\dot a_\chi = \langle r_\chi, \nu_\chi\rangle - \eta\, a_\chi, \qquad \dot r_\chi = a_\chi\, \nu_\chi - \eta\, r_\chi. \tag{18}$$

We call a class built while its weight $|a_\chi|\,\|r_\chi\|$ is not zero, and say it alone separates a state $c \neq 0$ when it is the only built class with $\chi(c) \neq 1$.

Proof of Lemma 4. At $\eta = 0$, $\frac{d}{dt}a_\chi^2 = 2a_\chi\langle r_\chi, \nu_\chi\rangle = \frac{d}{dt}\|r_\chi\|^2$, so $a_\chi^2 - \|r_\chi\|^2$ is constant. Once $\mathcal{L} < p_{\min}\log 2$, with $p_{\min} = \min_c p_c$, every state is on its correct side, since a state with $y_c f(c) \leq 0$ contributes $p_c \log 2$ by itself; gradient flow does not raise $\mathcal{L}$, so this holds from then on. Then for every $c \neq 0$,

$$0 \;<\; f(0) - f(c) \;=\; \sum_\chi a_\chi \langle r_\chi, z_\chi(0) - z_\chi(c)\rangle, \tag{19}$$

and only built classes with $\chi(c) \neq 1$ contribute. A class can therefore reach zero weight only once every state it alone separated is separated by another built class. If the carried classes form a minimal generating set, dropping any one χ leaves a set that does not generate, whose joint kernel holds a state $c^* \neq 0$ at which χ is the only carried class with $\chi(c^*) \neq 1$; so no class of the set reaches zero weight, and $f(0) - f(c^*) \leq 2\,|a_\chi|\,\|r_\chi\|$. As $\mathcal{L}$ falls to zero, $f(0) \to +\infty$ and $f(c^*) \to -\infty$, so the weight of every class grows without bound. □

Table 4: The settings of section 4.1. Table 10 lists the runs trained otherwise.

setting	value
architecture	LSTM, GRU or Elman network with tanh units, hidden width 128
initial weights	Xavier-normal at gain 0.1, the standard initialisation
seed	sets the initial weights and the batch order, and nothing else
optimiser	Adam at learning rate 10^{-3}, no weight decay
loss	cross-entropy of the label at every position, averaged over the batch
training set	500 strings at each length from 20 to 50, drawn once for an instance and shared by its runs
batch	128 strings of one length; each epoch visits the lengths in a fresh random order
budget	500 epochs, an epoch being one pass over the training set
held-out set	300 strings of length 50, a second draw; accuracy over every position after the first 8
solving	held-out accuracy at least 0.999 at the end of training
random strings	4000 strings of length 40, each symbol drawn uniformly, drawn once for an instance; at least 50 of them end in each state
checkpoints	at initialisation, then every epoch up to 60, then every fifth epoch, 149 in all
build	the first checkpoint where one class's share reaches one half and the non-trivial energy is at least 1.0
break	the first checkpoint where R falls below 3.0

Lemma 9. *In the model of Lemma 4 with weight decay $\eta > 0$, $a_\chi^2 - \|r_\chi\|^2$ decays as $e^{-2\eta t}$, and a class with $a_\chi > 0$ shrinks exactly when $\langle r_\chi, \nu_\chi \rangle < \eta\, a_\chi$. Once the regularised loss is below $p_{\min} \log 2$, a class reaches zero weight only once every state it alone separated is separated by another built class.*

Proof. The two equations of motion give $\frac{d}{dt}(a_\chi^2 - \|r_\chi\|^2) = -2\eta\,(a_\chi^2 - \|r_\chi\|^2)$, and the second claim is the first equation. The regularised loss bounds $\mathcal{L}$ from above and does not rise along the flow, so the argument of Lemma 4 applies from the time it falls below $p_{\min} \log 2$. □

B RUNS AND COUNTS

This appendix gives the training and reading details of section 4.1, sorts the instances into five cases, gives the counts of sections 4 and 5 by instance and architecture, and names the runs the figures draw. A row of a table is one instance and one architecture. Runs of different groups trained alike at one instance and architecture are counted together. Tables 1 and 7 take only runs whose R was read, so Table 9 can hold more runs than Table 7 at the same instance and architecture.

B.1 TRAINING AND READING

Table 4 lists the settings of section 4.1.

Reading. The code $\varphi(c)$ is the mean final hidden state over the random strings that end in c. For the LSTM the hidden state is its output h_t, which the readout reads, without the cell state. Every run of Table 7 was read at the checkpoints of Table 4, and in every run that built and broke, the two fell at different checkpoints. 108 runs of Table 2 and 15 of the runs on longer strings (Table 10) were read at 45 checkpoints in 500 epochs: at initialisation, then at every epoch up to 10, then at every fifth up to 100, then at every 25th. The 108 of Table 2 store no R, so Tables 1 and 7 do not count them. The build's energy floor sets aside checkpoints where a share is a ratio of small energies.

The break and its bar. R is read at every checkpoint from epoch 1, since the change in the input weights is zero at initialisation. Table 5 reads the runs of Table 1's second and third rows with the

Table 5: The break at other bars. The runs of Table 1's second and third rows, with the break read at other bars on R. The bar of section 4.1 is 3.0.

	symmetry still there at the build		broken before the build	
bar	runs	symbol-blind first	runs	symbol-blind first
1	164	137	8	1
2	129	124	43	14
3	119	119	53	19
5	114	114	58	24
10	110	110	62	28

break at other bars. At the bars 3, 5 and 10, every run whose symmetry lasted until the build built the symbol-blind class first. At 2 and 1 some built another class first.

Hardware and code. The 610 runs of Tables 7 and 10 were trained on one RTX A2000 (12 GB) GPU with PyTorch 1.12.1. A run of section 4.1's budget took a median of 13 minutes. The code is in the supplementary material.

B.2 THE FIVE CASES

Figure 3 draws three automata: $(3, 3)$ under both alphabets, which under the signed one is the instance of cases 4 and 5, and $(4, 6)$, the instance of Figure 2.

Two questions sort an instance with a symmetry. Does the symmetry protect every cheapest class? If not, what is the cheapest class it protects? With the instances that have no symmetry, the answers give five cases (Table 6), and Figure 4 draws one run of each. On the subspace (section 3.3), only protected classes can rise (Lemma 3). In case 2, agreement, cost and the symmetry name the same class. In case 3, symbol-blind override, the symmetry names the symbol-blind class, and cost names a counter's class or ties one with it. In case 4, block, no class but the constant is protected, so on the subspace none can rise. In case 5, orbit override, the symmetry names a class that takes one value on each orbit and tells advance symbols apart.

Under the one-way alphabet at equal moduli and $w = 1$, the counters' classes tie with the symbol-blind class for cheapest (Lemma 2). The symmetry moves the counters' classes, so these instances are in case 3. At balanced signed $(3, 3)$, the one instance of case 4, all 6 LSTM runs that built did so after the break.

Case 5 has one instance, outside section 4: signed $(3, 3)$ with each up symbol drawn at 0.24, each down symbol at 0.16 and the no-op at 0.20. Reversing a counter exchanges its up and down symbols, which are no longer drawn equally often, so the one symmetry left is the exchange of the two counters. The one class it protects, $\chi_{\pm(1,1)}$, tells up from down and has $|\mu| = 0.243$, against 0.406 for a counter's own class. At every other instance with a symmetry in Table 6, the class the symmetry names is symbol-blind or a cheapest class, or there is none. The runs, all LSTM, were trained for 1000 epochs, and the build was read with the floor at 0.05. All 6 runs built a class: 3 built the protected class first and 3 a counter's own class, which the exchange moves. The exchange was still there at the build in 5 runs, among them all 3 that built a counter's class first; it is read as R on each pair of columns the exchange carries into each other, the two up symbols and the two down symbols. Of all 6 runs, one solves at the end. Figure 4 draws one run of each kind, both with the exchange still there at the build.

Lemma 3 concerns columns that agree exactly; like every trained run, these start with columns that differ, and at the build theirs agreed only nearly. The instance shows that near agreement need not act as exact agreement, and appendix D leaves open how near is enough. The gap between the multipliers does not account for it: at signed (8) a class the symmetry moves is cheaper than the protected class by a larger factor, and every run built the protected class first (appendix B.3). Two things single out this instance: its protected class tells advance symbols apart, and its multiplier is the smallest of any class the symmetry names in Table 6. One instance cannot tell which of the two matters.

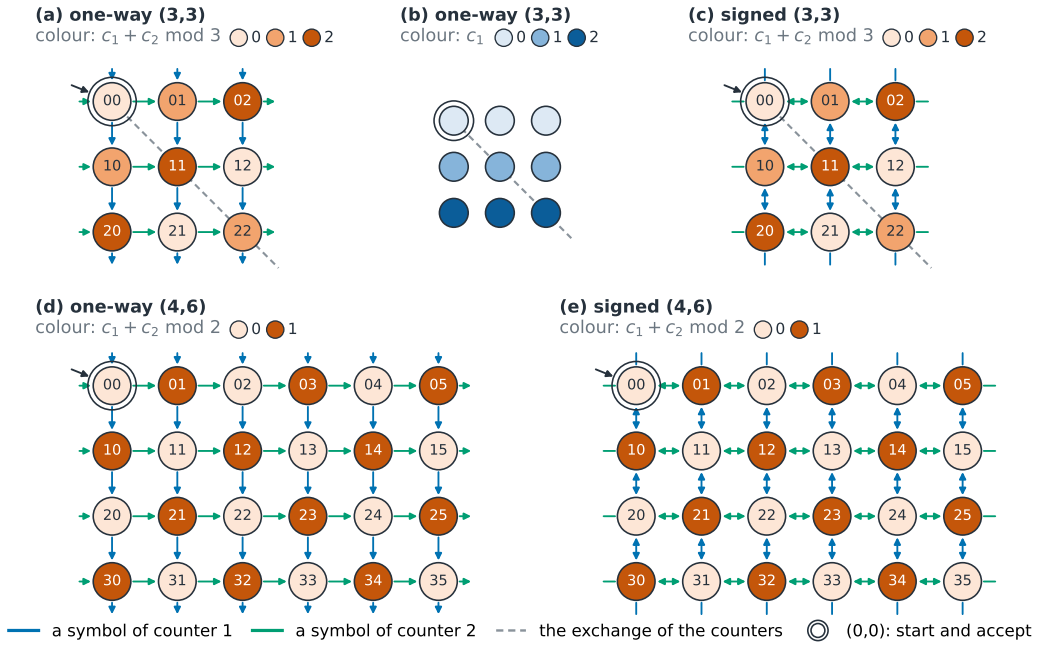


Figure 3: The automata of three instances. A state (c_1, c_2) sits at row c_1 and column c_2, and $(0, 0)$ is the start and the only accept state. The no-op's loops are not drawn, and an edge from $m - 1$ to 0 is drawn as two stubs at the border. (a) One-way $(3, 3)$, coloured by $c_1 + c_2 \bmod 3$, which the symbol-blind class reads; the exchange of the counters reflects the grid in the dashed diagonal, swapping the two symbols' arrows and leaving the colours in place. (b) The same states coloured by c_1, which the first counter's own class reads; the reflection turns it into c_2. (c) Signed $(3, 3)$, coloured as (a); an up and a down symbol move the colour in opposite directions. (d), (e) $(4, 6)$ under the two alphabets, coloured by the parity of $c_1 + c_2$, which the symbol-blind class reads under both; every advance symbol flips it.

B.3 Section 4 by instance and architecture

Table 7 splits Table 1's second and third rows by instance and architecture, and gives the runs trained as in section 4.1 at the instances with no symmetry. Two runs trained as in section 4.1 built no class, an LSTM run and an Elman run at $(3, 3, 3)$ at $w = 2$; no table of this section counts them, and the LSTM run solves. Table 1's first row sums the rows of Table 7 at $(2, 4)$ and $(4, 6)$, the two instances of its second block that have a symbol-blind class.

Where the symmetry protects every cheapest class. At $(6, 3, 3)$ the only symmetry exchanges the two counters of modulus 3, nine classes are protected, and the cheapest class is the modulus-6 counter's own, which the exchange fixes. The 12 LSTM runs trained as in section 4.1 whose R was read on the columns of the two exchanged counters all built it first, with the exchange still there at the build in 11. At $(3, 3, 3)$ at $w = 0.5$ every LSTM run built the symbol-blind class, the cheapest class there, first (Table 7), and so did 30 of the 36 GRU runs that solve there and the 6 runs at signed (4) at $w = 0.5$ (Table 9).

Where it does not. At the ties of $w = 1$ every LSTM run of Table 7 built the symbol-blind class first, and the GRU at $(3, 3, 3)$ did in 16 of 24 runs. At $(3, 3, 3)$ at $w = 2$, where cost names the counters, the symbol-blind class came first in 28 of 29 LSTM runs, 12 of 24 GRU runs and 10 of 23 Elman runs, and the median build falls at epoch 34, 65 and 155 on the three architectures. In every row of the first block each run that built one counter's class first had lost the symmetry before its build. At the signed instances of case 3 the symbol-blind class came first in every run: 6 of 6 at

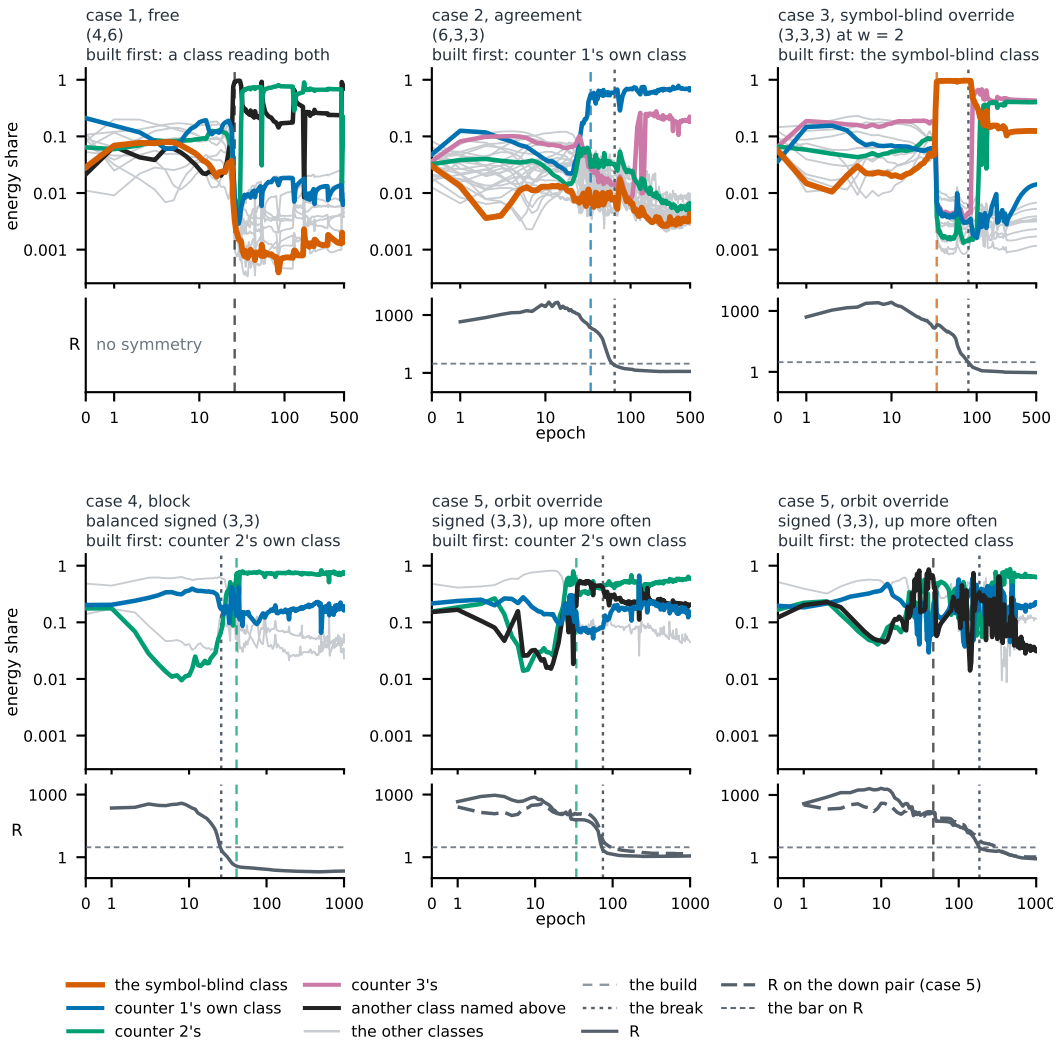


Figure 4: One run of each case, two of case 5. Each column is one LSTM run, drawn as in Figure 1(b): each class's share of the energy against the epoch, and below it R, read on the columns the instance's symmetry exchanges. In case 2 these are the columns of the two counters of modulus 3, in case 4 the four advance columns, which form one orbit, and in case 5 the two up symbols and the two down symbols. Cases 4 and 5 were trained for 1000 epochs and their builds read with the floor at 0.05. Appendix B.6 gives how each run was chosen.

signed (8) at $w = 0.5$ and 6 of 6 at signed (8), trained as in section 4.1, and 6 of 6 at signed $(4, 6)$, trained for 1000 epochs (Figure 2(a)).

Without a symmetry. No run of Table 7 built the symbol-blind class first at $(2, 4)$ or $(4, 6)$, the two instances of its second block that have one. Each run that built one counter's class first built the counter of larger modulus, on every architecture. The others built a class that reads both counters.

A head that names the state. With 27 outputs, one for each state of $(3, 3, 3)$, in place of the accept bit, and every other setting as in section 4.1, 23 GRU runs and 21 Elman runs learned every state. In each of them R was below 3.0 at every checkpoint where it is read. Of those runs, none on the GRU and 12 on the Elman network built the symbol-blind class first.

Table 6: The five cases, and the instances trained in each. Case 5's instance draws each up symbol at 0.24 and each down symbol at 0.16.

case	name	does the symmetry protect every cheapest class?	the cheapest class it protects	instances trained
1	free	no symmetry: every class is protected		$(2,4)$, $(3,5)$, $(3,6)$, $(3,7)$, $(4,6)$, $(5,6)$, $(6,5)$; (3), (4), (8)
2	agreement	yes	a cheapest class	$(6,3,3)$; $(3,3,3)$ at $w=0.5$; signed (4) at $w=0.5$
3	symbol-blind override	no	the symbol-blind class	$(3,3,3)$, $(4,4)$, $(2,2,2,2)$, $(2,2,2,2,2)$ and, in the gain ladder, $(3,3)$; $(3,3,3)$ at $w=2$ and 4; signed (8) at $w=0.5$ and 1; signed $(4,6)$
4	block	no	none but the constant	balanced signed $(3,3)$
5	orbit override	no	a class that tells advance symbols apart	signed $(3,3)$ with the up symbols drawn more often

The columns tied at initialisation. At $(3,3,3)$ at $w=2$ the GRU was trained from its standard initialisation with the three advance columns of the input weights set equal (a copy of the first, nothing else changed), on the seeds of its runs in Table 7, and again with the training draw and every minibatch closed under the symmetry as well, so that the loss is exactly invariant. Table 8 reads the three groups side by side, each on its own, with the build gated as in section 4.1. With the columns tied the symbol-blind class is built first in 17 of 24 runs (16 of the 23 that solve) against 12 of 24 from the standard initialisation, and with the draw closed as well in 22 of 24. The tie moves the build earlier and the break little; the closed draw moves the break later. In every run of the three groups whose symmetry lasted until the build, the class built first was the symbol-blind one. The closed draw did not hold the tie: the spread of the three columns about their mean, relative to the mean, passed 0.001 at a median epoch of 65, though the loss is exactly invariant; in a trial run it was at rounding level after one epoch.

B.4 Section 5 by instance and architecture

Table 2 sums the first three blocks of Table 9. Among the runs Table 2 leaves out, 54 solve and were trained otherwise: 30 on another length range, 7 for longer, 12 with a penalty that held down the share of one class, the symbol-blind class or a counter's own, and 5 with the label given at only some positions, every second one or only the late ones. From initial gains above the standard one, with every other setting as in section 4.1, 6 runs that solve and built a class lose the first-built class.

Whether or not it is symbol-blind. The first-built class is not symbol-blind in 226 of the 557 runs of Table 2 that built a class, among them every run without a symmetry, and all but three keep it, the runs at $(3,5)$ and $(4,6)$ of the paragraph on the losses below. At signed (4) and signed (8) the symbol-blind class reads the counter's parity, which the counter's own class determines, so no minimal generating set contains it and Lemma 4 does not cover it; every run there kept it (Table 9).

The size of the code. The size equals the bound in 310 runs, exceeds it in 223 and falls below it in 24 (Table 9). In the runs below it a class the code needs holds too little of the variance to enter the 95 per cent count.

The losses. Of the runs of Table 2, four lose the first-built class. The run at $(3,5)$ built a class that reads both counters at epoch 23 and first solved at epoch 55. At epoch 150 the total energy of the non-trivial characters fell from 27.2 to 7.0 and held-out accuracy to 0.9333; the first-built class

Table 7: Section 4 by instance and architecture. Runs at the standard initialisation that built a class, trained as in section 4.1, with two or more counters under the one-way alphabet: whether the symmetry was still there at the build, and the first-built class. At unequal moduli every run that built one counter's class first built the counter of larger modulus. A dash marks a question that does not apply: no symmetry, or no symbol-blind class.

				first-built class		
instance	architecture	runs	still there	symbol-blind	one counter's	other
with a symmetry						
$(3,3,3)$ at $w = 0.5$	LSTM	24	24	24	0	0
$(3,3,3)$	LSTM	36	36	36	0	0
	GRU	24	4	16	8	0
$(3,3,3)$ at $w = 2$	LSTM	29	28	28	1	0
	GRU	24	8	12	12	0
	Elman	23	8	10	13	0
$(4,4)$	LSTM	6	6	6	0	0
$(2,2,2,2,2)$	LSTM	6	5	6	0	0
without a symmetry						
$(2,4)$	LSTM	12	–	0	12	0
$(3,5)$ at $w = 0.5$	LSTM	6	–	–	2	4
$(3,5)$	LSTM	18	–	–	15	3
$(3,7)$ at $w = 0.5$	LSTM	6	–	–	6	0
$(3,7)$	LSTM	6	–	–	6	0
$(4,6)$	LSTM	24	–	0	7	17
	GRU	24	–	0	23	1
	Elman	24	–	0	23	1
$(5,6)$ at $w = 0.5$	LSTM	18	–	–	8	10
$(5,6)$	LSTM	12	–	–	6	6
$(6,5)$	LSTM	24	–	–	24	0

Table 8: The advance columns tied at initialisation. GRU runs at $(3,3,3)$ at $w = 2$ on the seeds of Table 7's GRU runs there, whose row is the first. Each group stands on its own. The build is gated as in section 4.1, the break is the first checkpoint with R below its bar, and a median over an even count is the mean of the two middle epochs.

				still there at the build		median epoch	
initialisation	runs	solve	symbol-blind first	runs	symbol-blind first	build	break
standard	24	24	12	8	8	65	57
columns tied	24	23	17	15	15	40	65
tied, draw closed	24	19	22	22	22	85	160

fell below half its peak energy then and never recovered. The modulus-3 counter's class reached a share of 0.05 at epoch 160, and the run solved again at epoch 175. The code collapsed once more at epoch 250 and recovered. Two LSTM runs at $(4,6)$ built $\chi_{\pm(1,1)}$, which reads both counters, first, at epochs 31 and 27; they solved at epochs 85 and 45, the class fell below half its peak energy at epochs 210 and 340, and each run ended on the modulus-6 counter's class. A GRU run at $(3,3,3)$ at $w = 2$ built the symbol-blind class at epoch 75, lost it at epoch 160 and first solved at epoch 425. Against Lemma 4: the GRU run lost its class before it first solved, and the run at $(3,5)$ in a collapse during which it stopped solving, so neither loss came while every state was labelled correctly, the lemma's condition. The runs at $(4,6)$ solved on both sides of the loss, and at no checkpoint from their first solve to the loss did another class with a share of at least 0.05 separate the states $\chi_{\pm(1,1)}$ alone separated, though in one of the two runs a class of smaller share did. The first-built class declined where the lemma's model has a needed class grow, and it was still carried at the end.

Table 9: Section 5 by instance and architecture. The runs of Table 2, and the one-way single counters it sets apart: the runs that solve, those that keep the first-built class, those whose first-built class is symbol-blind, and the size of the code at the end against the dimension of the smallest generating set that contains the first-built class. At $(3, 3, 3)$ at $w = 2$, one run built no class above the floor.

					size against the bound		
instance	architecture	solve	kept	symbol-blind	equal	larger	smaller
with a symmetry							
signed (4) at $w = 0.5$	LSTM	6	6	6	6	0	0
signed (8) at $w = 0.5$	LSTM	6	6	6	6	0	0
signed (8)	LSTM	6	6	6	4	2	0
$(3, 3, 3)$ at $w = 0.5$	LSTM	24	24	24	19	3	2
	GRU	36	36	30	32	3	1
$(3, 3, 3)$	LSTM	108	108	108	77	27	4
	GRU	33	33	20	26	7	0
$(3, 3, 3)$ at $w = 2$	LSTM	28	27	26	24	2	1
	GRU	24	23	12	19	5	0
$(3, 3, 3)$ at $w = 4$	LSTM	4	4	3	4	0	0
$(4, 4)$	LSTM	36	36	36	5	31	0
	GRU	36	36	36	9	27	0
$(2, 2, 2, 2)$	LSTM	12	12	12	4	8	0
$(2, 2, 2, 2, 2)$	LSTM	6	6	6	3	3	0
$(6, 3, 3)$	LSTM	36	36	0	11	22	3
no symmetry, a symbol-blind class							
$(2, 4)$	LSTM	12	12	0	11	0	1
$(4, 6)$	LSTM	22	20	0	5	16	1
	GRU	20	20	0	0	20	0
	Elman	1	1	0	0	1	0
no symmetry, no symbol-blind class							
$(3, 5)$ at $w = 0.5$	LSTM	6	6	–	2	4	0
$(3, 5)$	LSTM	18	17	–	12	3	3
$(3, 7)$ at $w = 0.5$	LSTM	5	5	–	3	0	2
$(3, 7)$	LSTM	4	4	–	1	0	3
$(5, 6)$ at $w = 0.5$	LSTM	14	14	–	8	6	0
$(5, 6)$	LSTM	11	11	–	3	7	1
$(6, 5)$	LSTM	23	23	–	16	5	2
	GRU	21	21	–	0	21	0
a single counter, one-way							
(3)	LSTM	6	6	6	6	0	0
(4)	LSTM	6	6	6	6	0	0
(8)	LSTM	5	5	5	3	2	0

The left-out counter. A run ends with one counter left out when the k classes of largest share at the end lie in the tier and the tier class missing from them is a counter's. At $(4, 4)$ every run that solves ends this way, 36 LSTM runs and 36 GRU runs, the left-out class holding a share of 0.01 to 0.19 at the end. At $(3, 3, 3)$ 88 of 108 LSTM runs and 30 of 33 GRU runs end this way, with a median left-out share of 0.0052. At $(2, 2, 2, 2, 2)$ 1 of 6 runs does, with a share of 0.0014, and at $(2, 2, 2, 2)$ 3 of 12 do, with a median share of 0.0029.

B.5 Runs trained otherwise

Table 10 lists the runs at the standard initialisation that were trained on longer strings, for longer, at other widths or by another optimiser, and the gain ladder.

Table 10: Runs trained otherwise. Runs at the standard initialisation trained otherwise than in section 4.1, and the gain ladder, trained as in section 4.1 but for the initial gain. Every run whose symmetry was still there at the build built the symbol-blind class first. A dash marks a count not given: at signed $(4, 6)$ the symmetry at the build, elsewhere a question that does not apply (no symmetry, or no symbol-blind class). The longer strings take their lengths from the instance: 48 to 144 at $(2, 4)$, $(2, 2, 2, 2, 2)$ and $(3, 3, 3)$ at $w = 2$, 72 to 216 at $(3, 6)$ and 84 to 252 at $(3, 7)$, with 500 strings at each length.

instance	architecture	differs from section 4.1	runs	still there	symbol-blind first
with a symmetry					
$(3, 3, 3)$	LSTM	2000 epochs	8	8	8
$(3, 3, 3)$	LSTM	width 32	6	6	6
$(3, 3, 3)$	LSTM	width 64	6	6	6
$(3, 3, 3)$	LSTM	width 256	6	6	6
$(3, 3, 3)$	LSTM	SGD at learning rate 0.3, gains 0.5 and 5	6	3	4
$(3, 3, 3)$ at $w = 2$	LSTM	longer strings	24	20	23
$(2, 2, 2, 2, 2)$	LSTM	longer strings	6	4	5
signed $(4, 6)$	LSTM	1000 epochs	6	–	6
$(3, 3, 3)$, $(3, 3)$, $(2, 2, 2, 2)$	LSTM, GRU	initial gains 0.3 to 5	157	28	98
without a symmetry					
$(2, 4)$	LSTM	longer strings	18	–	3
$(3, 6)$	LSTM	longer strings	9	–	0
$(3, 7)$	LSTM	longer strings	6	–	–
$(4, 6)$	LSTM	1000 epochs	6	–	0

B.6 THE RUNS IN THE FIGURES

Each run drawn in a figure is chosen by a rule on its cohort, a group of runs trained alike at one instance and architecture. A median run is the lower of the two middle runs when their number is even, and ties go to the lower seed.

Figure 1(b). One LSTM run at $(3, 3, 3)$ trained as in section 4.1. Among its cohort's runs that solve, it is the median run by its worst-read counter. A counter is read by the balanced accuracy of a logistic probe on the hidden state at the end of training; the supplementary material's runs table gives it for every counter of every run. The symbol-blind class is built at epoch 23. Its share peaks at 0.98 and ends at 0.12, while its energy ends at 0.91 of its own peak and after the peak never falls below 0.52 of it.

Figure 2. Panels (a) and (b) are LSTM runs at signed $(4, 6)$ and at one-way $(4, 6)$, trained for 1000 epochs (Table 10). Each is the median run of its group by build epoch. Panels (c) and (d) are GRU runs at $(3, 3, 3)$ at $w = 2$. Panel (c) is the median run, by the ratio of break epoch to build epoch, among the runs whose symmetry was still there at the build and that built the symbol-blind class first. Panel (d) is the same among the runs whose symmetry broke before the build and that built another class.

Figure 4. Each column comes from one cohort of LSTM runs: at $(4, 6)$, $(6, 3, 3)$ and $(3, 3, 3)$ at $w = 2$, trained as in section 4.1, and the runs of case 4 and of case 5. In each cohort the column is the median run by build epoch. Case 5 has two columns, one from the runs that built a counter's own class, which the exchange moves, and one from the runs that built the protected class, all with the exchange still there at the build.

C EXTENDED RELATED WORK

This appendix extends the related work of the body. It compares our task with the closest one in more detail, gives in full the lines of work the body summarises, and then takes up three further lines.

C.1 THE CLOSEST TASK

Marchetti et al. (2026) study sequential group composition. A network reads a sequence of group elements, each encoded as a vector, and is trained with the squared error to output the encoding of their product. Trained from vanishing initialisation, a two-layer network learns the task one irreducible representation at a time, in an order set by the Fourier statistics of the encoding. To learn it exactly, such a network needs a hidden width exponential in the length of the sequence. They construct recurrent weights that compose the elements one step at a time, with a width that does not grow with the length. A recurrent network they trained on a product of two cyclic groups of equal order acquired the irreducible representations one at a time. They leave a dynamical theory of recurrent networks open.

Our task differs from theirs in three ways. First, their target is the whole encoding of the product. A network that learns it exactly carries every irreducible representation, so what training sets is the order in which they come. Our label is one bit a step, whether the state is zero. A code that supplies it carries a generating set of classes (Theorem 1), and under the idealisation L a generating set is enough (appendix A). So training sets both which classes the code carries and the order in which it builds them.

Second, their ordering assumes that the statistic takes distinct values on the representations. To see the order in experiments they choose encodings whose representations differ in power, noting that representations of similar power tend to be learned together. Under the one-way alphabet at equal moduli and $w = 1$ several of our classes share the largest multiplier (Lemma 2), and Lemma 3 limits which of them can rise first.

Third, their recurrent result is a construction with a quadratic activation. We train LSTM, GRU and Elman networks with tanh units and measure which class each builds first (section 4).

Their discussion proposes semiautomata, and the regular languages they recognise, as a next step for their framework. Our task is one such language, the word problem of an abelian group (section 2.1), in which every symbol permutes the states.

C.2 THE LINES OF THE BODY

Automata read from networks. Networks trained as acceptors of regular languages have been read back as automata, by partitioning the hidden state (Giles et al., 1992) or by querying the network (Weiss et al., 2018). The reading returns states, some of them non-minimal (Muškardin et al., 2024). Michalenko et al. (2019) decode the minimal automaton's state from the hidden vector, and the decoding merges small sets of its states. We ask what the states are built from: which features of the group the network carries, and in what order training builds them.

State tracking. Butoi et al. (2025) train recurrent networks as recognizers, and we train ours the same way, with a label at every prefix. Networks have also been trained as transducers, which output a string, across the Chomsky hierarchy (Delétang et al., 2022), and the transductions transformers express have been related to classes of finite transducers (Strobl et al., 2024). The state-tracking line asks which groups and automata an architecture can express and learn (Merrill et al., 2024; Sarrof et al., 2024; Ebrahimi et al., 2026), and negative eigenvalues widen what linear recurrences can track (Grazzi et al., 2024). An abelian group is among the benchmark groups, and the hidden state is left unopened. The gap between expressing a solution and learning one has been named (Shakerinava et al., 2026). Our question starts in that gap: we open the hidden state on abelian groups and ask what training builds on the way to a solution.

Implicit bias in recurrent networks. Under conditions on the initialisation, gradient descent on linear recurrent networks learns a low-dimensional state that extrapolates to longer sequences

(Cohen-Karlik et al., 2022a;b). Some correctly labelled training examples can distort this bias in structured state-space models until generalisation fails (Slutzky et al., 2025). These results concern the solution that training reaches; we ask in what order nonlinear and gated cells build features on the way to one.

Which Fourier features a network uses. Networks trained to output the group element compute with a few Fourier features of the group (Nanda et al., 2023; Zhong et al., 2023; Chughtai et al., 2023). Which ones they build, and in what order, varies across seeds and hyperparameters (Chughtai et al., 2023; Ding et al., 2024). In one account the features compete for capacity (Ding et al., 2024), in another the choice is arbitrary (Chughtai et al., 2023), and in a third the statistics of the input encoding set the order (Marchetti et al., 2026). On modular addition, small changes of hyperparameters or initialisation lead a network to different algorithms from one training set (Zhong et al., 2023). For one-hidden-layer networks on modular addition with several inputs, margin maximisation makes each neuron focus on one Fourier frequency (Li et al., 2024). In a two-layer network the features that emerge are the local maxima of an energy, which each hidden unit climbs while the units learn independently (Tian, 2025). For a single neuron, alignment with one frequency can fail from an open set of initial conditions (Memana, 2026). In these works the network reads all its inputs at once and the whole answer is supervised.

Fourier features in recurrent networks. Recurrent networks trained on modular addition carry Fourier features as well (Rangamani, 2025), and a count of them has been predicted and attained (McCracken et al., 2025). In these works the whole sum is supervised. Continuous-time recurrent networks trained to report one bit, whether a sum of three residues is zero modulo 3, reached one of several solutions; a phase code appeared in 6 of 90 networks, all at one activation and time constant (Murray, 2023). Marchetti et al. (2026) treat the sequential case, and section 3.2 compares our multiplier with their statistic. We supervise one bit a step, whether the state is zero, and ask which of the group's features is built first.

Symmetry in training. Lemma 3's first clause is the principle of symmetric criticality (Palais, 1979), which has deep-learning forms as an invariant set (Chen et al., 2023), a mirror symmetry (Ziyin, 2024) and, for equivariant layers, an invariant set of layers (Nordenfors et al., 2023). A network invariant to a finite group is known to hold the group's Fourier features at the end of training (Marchetti et al., 2024). We add the lemma's second clause, a cap on every class the symmetry moves, so that on the subspace (section 3.3) only protected classes can rise.

Stages and transients. Training from small initialisation acquires features in stages ordered by rate (Saxe et al., 2014; Kunin et al., 2025), through saddles (Jacot et al., 2021; Abbe et al., 2023), and features of equal rate arrive together (Saxe et al., 2019). We find that at such a tie a symmetry decides which class rises first, the case the tie-free ordering of Marchetti et al. (2026) leaves out. Structure built early can be lost or reorganised later, from low-order statistics (Belrose et al., 2024) to circuits for in-context learning (Singh et al., 2023) and stages of development at large (Hoogland et al., 2024; Carroll et al., 2025). In our networks what is built first is kept in almost every run that solves. We trace the keeping to the balance that gradient flow conserves between the layers of a homogeneous network (Du et al., 2018), here between a class's amplitude and its readout vector (section 5.2).

C.3 Further lines of work

What recurrent and state-space models can recognise. A selective state-space model with dense transition matrices emulates a range of automata and generalises to longer strings, and diagonal ones have been compared on commutative and non-commutative automata (Terzić et al., 2024). Whether a diagonal state-space model implements a counter of even modulus depends on the arithmetic of its recurrence (Nowak et al., 2026). Transformers simulate automata in far fewer layers than steps, and trained ones find such shortcuts (Liu et al., 2023). These results fix what an architecture can compute. We train cells that can compute our task and ask which features training builds.

Features of more than one dimension. Some features of language models are irreducibly multi-dimensional, such as circles for the days of the week and the months of the year. The models

use these circles for modular arithmetic on days and months (Engels et al., 2025). A class of two conjugate characters is a feature of this kind: its part of the code places the states on an ellipse in a plane of the hidden space (section 5.2). Recurrent networks trained to recall a sequence after a delay can hold its items in superposition across time. The geometry of these features depends on the memory demand and the width (Sharma et al., 2026). In our task the candidate features are the classes, known from the group before training, and we read the code in their basis.

The order of emergence. Token embeddings trained from small initialisation align with low-order statistics of the data before high-order ones (Yao et al., 2026). In a recurrent network trained on streaming parity, the automaton read from the hidden state first fits the training strings as a tree. Its states then merge until the automaton is finite (van Rossem & Saxe, 2025). Our order is among the features of one group, where the multipliers can tie and a symmetry of the presentation limits which classes rise (Lemma 3).

D Limitations and open questions

Every result is on the shallow end of the Krohn-Rhodes and Barrington ladder, the abelian groups (Krohn & Rhodes, 1965; Barrington, 1989); the transformer side of the ladder has been measured (Liu et al., 2023), and the rungs above are left for later. We train three recurrent architectures, mostly at one width, one initial gain and one optimiser setting, Adam at one learning rate (section 4.1); appendix B.5 lists the runs trained otherwise, and the rest of the space is untested. The counts of section 4.2 compare two epochs within a run, the build's and the break's, so a run that builds late gives the symmetry longer to break; the counts say which came first. No result on the order in which features are learned, or on the dimension of the learned state, is proved for gated cells: the recurrent proofs are for linear recurrences (Cohen-Karlik et al., 2022b; Proca et al., 2025), and the nearest gated result describes the gates as preconditioners of the optimisation (Livi, 2025). Lemma 4 holds each class's plane fixed and moves only its amplitude and readout vector, by gradient flow (section 5.2); for networks that move every weight by Adam the keeping is measured. Two runs at $(4, 6)$ depart from the lemma's model: the first-built class declined where the model has it grow, though it stayed in the code (appendix B.4).

When the break comes depends on the architecture and on the label (section 4.4); why the columns part at all, and what sets the epoch, is open. Section 5.2's conjecture, that training completes a smallest generating set around the first-built class, fits the size of most codes, and at $(4, 4)$ and $(4, 6)$ most codes are larger. With weight decay the difference of Lemma 4 is no longer conserved, and in the lemma's model a needed class can shrink, reaching zero weight only after another built class separates every state it alone separated (appendix A.6); our runs, all trained without decay, do not test this. At one instance outside section 4, some runs built a class the symmetry excludes before the break (appendix B.2), so how near the columns must stay for the prediction of section 3.3 to hold is open.