GROKKING CAN BE STEERED: SELECTING DESIRABLE SOLUTIONS BY CHOICE OF LEARNING RULE

Anonymous authors
Paper under double-blind review

ABSTRACT

During grokking, a network transitions from memorization to generalization, yet what shapes the neural implementation (circuit) of the generalizing solution and specifically the number of participating neurons (its size) remains unclear. We find that the learning rule, the combination of credit assignment algorithm and optimizer, controls this: On modular addition, every learning rule we study that groks recovers the same Fourier-based algorithm, yet implemented by circuits that differ in size. Why? Experimentally, we find that equalizing the magnitude of credit assigned to each neuron is responsible, e.g., expanding the effective circuit size, measured from the outgoing weights, from 60% to 99% of the layer's neurons at similar training and test accuracy. We vary which parameters are normalized together, from the whole layer to individual neurons, and find that only normalization per neuron fully distributes the circuit. Credit magnitudes are usually set by the built-in normalization of standard optimizers: Nero, which normalizes per neuron, distributes the computation, whereas Adam and seven others, including RMSProp, Muon, and Shampoo, use other normalization schemes and retain small, concentrated circuits. Distributed circuits tolerate neuron deletion better, while concentrated circuits transfer about twice as fast to a task that reuses the same features. Whether a learning rule groks at all is visible after 2% of training: The rules that memorize retain larger weight components in directions that leave training outputs locally unchanged. Learning rule choice thus steers circuit size, trading robustness against transfer speed without changing the learned algorithm.

1 INTRODUCTION

A neural network can reach nearly perfect training accuracy while test accuracy remains near zero for many epochs before improving abruptly. This delayed generalization is called *grokking* (Power et al., 2022). One can distinguish two aspects of a grokked solution: the computational procedure used to compute the function, which is called its *algorithm*, and how that procedure is implemented across neurons, which is called its *circuit*. Modular addition is a well-understood example of grokking and a central task in mechanistic interpretability of small networks. Its algorithm and circuit can both be analyzed at the neuron level: the network represents each operand as an angle on a circle at a few frequencies and adds the angles, a periodic computation carried by the neurons tuned to those frequencies (Nanda et al., 2023; Gromov, 2023; Chughtai et al., 2023). When we say that two networks learn the same algorithm, we mean that they share this Fourier-based computation (Appendix B.3).

The concentrated (i.e., small) circuits reported for modular addition leave open whether the concentration is required by the task or produced by the training procedure. Chughtai et al. (2023) describe the circuits learned for group composition in terms of group representations, but find that which representations a network learns, and in which order, varies across runs. Optimization research offers a potential explanation: the *learning rule*, i.e., the combination of credit assignment algorithm and optimizer, *chooses* among the many parameter settings that fit the training data (Gunasekar et al., 2018; Woodworth et al., 2020). Yet the networks in these circuit studies have all been trained with similar learning rules based on backpropagation.

In this paper, we ask how the learning rule selects the circuit that implements a learned algorithm: In our experiments on modular addition, the same periodic computation appears in circuits ranging

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.

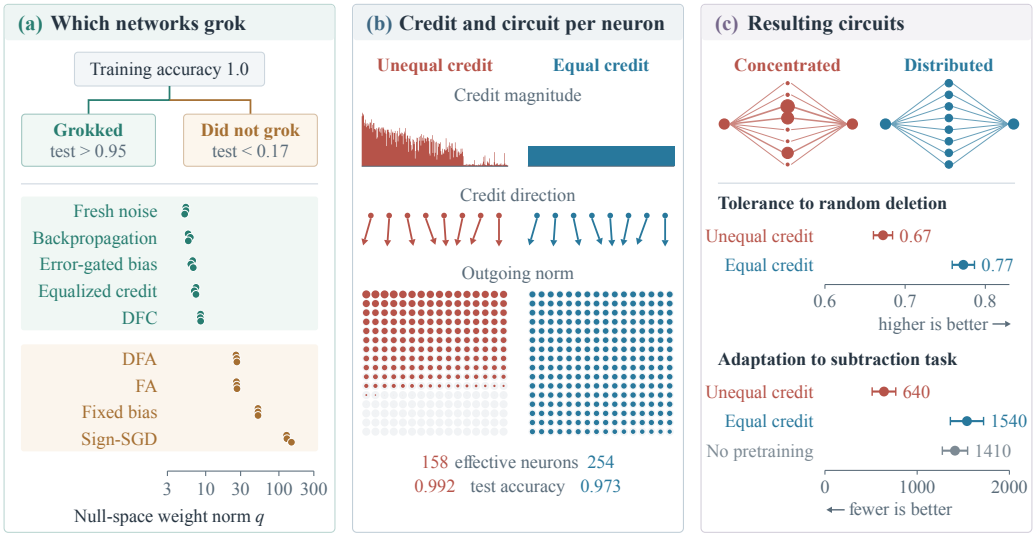


Figure 1: The learning rule selects whether a network groks, how many neurons implement the grokked algorithm, and the resulting robustness and adaptation. **(a)** All runs fit the training set, and only some then generalize to the held-out pairs. Each dot is one seed and shows the null-space weight norm q at the end of training (Section 4.3). **(b)** First hidden layer of two networks that share data and initialization. The credit of a neuron is the gradient of its incoming weights. Under backpropagation with Adam the credit is unequal (top row, relative to the layer mean and averaged over training), and neurons that receive little credit end with small outgoing weights. Credit equalization makes the magnitudes equal before the gradient enters the optimizer (Adam) and keeps every gradient's direction (middle row, schematic). Dots show the 256 neurons in the order of the top row, with area proportional to the outgoing norm. Effective counts are for this seed, and the three-seed means are 154 and 254. Nero, which normalizes credit per neuron, and DFC also produce distributed circuits (Section 4.2). **(c)** Tolerance is the normalized area under the curve of test accuracy against the fraction of randomly deleted neurons. Adaptation is the number of epochs to test accuracy 0.9 on subtraction, starting from the addition network or from scratch (without pretraining). The network drawings are schematic, $\mu \pm \sigma$ are over ten seeds (Table 16 and Appendix E.1).

from concentrated to nearly fully distributed. Effective circuit size varies across learning rules but little across random seeds. We investigate which property of the learning rule controls concentration and what the resulting differences imply for robustness and adaptability (Figure 1).

Our central hypothesis is that circuit concentration depends on how the learning rule distributes credit across neurons, where *credit* denotes the learning signal assigned to a neuron. Backpropagation with Adam assigns unequal credit, and the neurons that receive little of it end with small outgoing weights (Figure 1b). To isolate this effect, we modify backpropagation so that every neuron receives the same credit magnitude while its gradient direction is preserved, and we compare runs that share the data, initialization, loss, optimizer and hyperparameters. Equalizing credit increases the effective neuron count, computed from the outgoing weight norms, from approximately 154 to 254 of 256 neurons in the first hidden layer while retaining the periodic computation. Training loss remains the same within a few percent, and test accuracy is about two percentage points lower. Applied after grokking, the same intervention redistributes the outgoing weights of an already learned circuit at unchanged test accuracy.

Both parts of the learning rule shape the credit. The credit assignment algorithm computes each neuron's signal, and the optimizer rescales it before the update, so an equalization can be inserted between the two, as in our intervention, or be built into the optimizer, as in Nero. We then ask whether the effect depends on which parameters are normalized together: the weights into one neuron, a group of neurons, or an entire layer. Normalizing each neuron separately produces a nearly fully distributed circuit, while larger groups retain most of the concentration. A comparison of eight optimizers agrees with this distinction: Nero, which normalizes per neuron, produces a

distributed circuit, and the other seven keep the circuit concentrated (Section 4.2). Deep feedback control (DFC), which assigns credit through a feedback controller instead of a backpropagated loss gradient (Meulemans et al., 2021), provides a complementary example of a distributed circuit. To our knowledge, this is also the first demonstration of grokking in a neural network trained without a backpropagated loss gradient.

Our contributions are the following. (i) Credit equalization, a controlled intervention on the learning rule that changes circuit concentration while the algorithm and the training fit stay the same and test accuracy stays within about two percentage points, including when applied after grokking. (ii) A controlled comparison shows that normalizing each neuron separately produces nearly uniform outgoing-weight participation, whereas normalizing larger groups retains concentration. (iii) A trade-off: distributed circuits better tolerate neuron deletion, whereas concentrated circuits adapt about twice as fast from addition to subtraction, a task that shares the learned operand frequencies. The redistribution holds on modular subtraction and multiplication, permutation composition in S_5, sparse parity and a transformer with a wide MLP, and changes little where the reference circuit is already distributed (Appendix F). (iv) An early indicator of grokking: the rules that memorize keep weight in parameter directions along which the training outputs do not change locally, and this quantity separates them from the rules that later grok already after 2% of training, before test accuracy or weight norm does (Section 4.3).

2 Background and related work

Grokking and the assumed solution Grokking was first reported on modular arithmetic and other small symbolic tasks (Power et al., 2022) and later on image classification (Liu et al., 2023). Existing research explains the emergence of generalizing computations through circuit efficiency, that is, larger logits at a given parameter norm, which weight decay rewards (Varma et al., 2023), through the weight norm at initialization (Liu et al., 2023), and through a transition from nearly fixed initial features to learned task-specific features (Kumar et al., 2024). Nanda et al. (2023) track the Fourier circuit as it forms before test accuracy rises. A related theoretical analysis connects frequency selectivity with circuit size: after grokking, a one-hidden-layer network approaches solutions with single-frequency neurons and a computation distributed across many neurons (Morwani et al., 2024).

Other studies report multiple solutions, but these differ in their algorithms rather than only in the number of units used. Changing the architecture, hyperparameters or seed can produce a different algorithm (Zhong et al., 2023). A sparse generalizing subnetwork can also coexist with a dense memorizing subnetwork (Merrill et al., 2023). Smaller initialization scales can concentrate a two-layer network into fewer effective neurons (Luo et al., 2021), without identifying which circuit will form. Closest to our finding, Manning-Coe et al. (2025) compare networks that grok with networks that generalize without delay and find that both learn the same features but encode them with different efficiency. Our comparison instead isolates credit allocation and measures how it distributes the circuit that implements the same algorithm.

What the learning rule selects The implicit-bias literature studies how optimization selects a solution among the many parameter settings that fit the training data. Such selection has been characterized for steepest descent under different norms (Gunasekar et al., 2018) and for product parameterizations with small initialization (Woodworth et al., 2020). On grokking tasks, parameterization scale affects feature learning (Kumar et al., 2024), and the optimizer affects whether grokking occurs (Thilak et al., 2022). We connect this selection to circuit concentration through a controlled intervention on credit allocation.

Alternatives to backpropagation Several methods compute learning signals without using standard backpropagation. These include feedback alignment (Lillicrap et al., 2016; Nøkland, 2016), predictive coding (Whittington & Bogacz, 2017), equilibrium propagation (Scellier & Bengio, 2017), and deep feedback control (DFC) with its least control principle (Meulemans et al., 2021; 2022). They are usually evaluated by accuracy, which does not establish whether they learn the same computation or circuit as backpropagation. Refinetti et al. (2021) show that feedback alignment initially aligns with the gradient and later memorizes. Mallinar et al. (2025) demonstrate grokking without backpropagation in recursive feature machines, a kernel method without a neuron-level circuit.

3 EXPERIMENTAL SETUP

Task and network. We study modular arithmetic, a central task in grokking research (Section 2). The operands $a, b \in \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ are represented by concatenated one-hot vectors, and the target is $c = (a \circ b) \bmod p$ with $\circ \in \{+, -, \times\}$. Our primary task is addition modulo the prime $p = 31$. We use a fully connected ReLU network with two hidden layers of width 256 and a linear readout; multilayer perceptrons of this kind are standard in grokking studies of modular arithmetic (Gromov, 2023; Morwani et al., 2024). Unless stated otherwise, circuit measurements and neuron deletion concern the first hidden layer.

Training. Our reference learning rule is backpropagation with Adam (Kingma & Ba, 2015), cross-entropy loss and coupled weight decay 10^{-3}. Runs that we compare with each other share the data split and the initialization. We hold out 40% of the operand pairs, train full batch, and apply credit equalization to the first hidden layer; budgets are stated with each experiment (Appendix A). We classify a run as grokking when training accuracy reaches 1.0 and test accuracy first exceeds 0.95 at least 500 epochs later. Results are means over three seeds unless stated otherwise, with standard deviations in parentheses or as error bars.

3.1 LEARNING RULES

We distinguish two parts of training. A *credit assignment algorithm*, such as backpropagation, computes a learning signal for each parameter. We call the signal received by a hidden neuron its *credit*. An *optimizer*, such as SGD or Adam, converts these signals into weight updates. We use *learning rule* for the combination of the two and vary each part in turn (Table 1).

Credit equalization is the intervention we introduce. It rescales the magnitude of each neuron's incoming-weight gradient before the gradient enters Adam and leaves its direction unchanged (Section 4.1). The optimizer comparison keeps backpropagation and replaces Adam with LARS, RMSProp, LAMB, Adafactor, Lion, Muon, Shampoo or Nero. DFC replaces the backpropagated loss gradient. During training, a feedback controller drives the network output toward a target, its signal reaches each layer through the transposed Jacobian of the forward network, and each layer learns from the difference between its controlled and its feedforward activity (Meulemans et al., 2021). Section 4.3 asks which rules grok at all and adds six rules for that purpose. Feedback alignment and direct feedback alignment replace the transposed forward weights of the backward pass by fixed random matrices (Lillicrap et al., 2016; Nøkland, 2016), and sign-SGD keeps only the sign of each gradient coordinate (Bernstein et al., 2018). Three constructed rules add to the gradient a component orthogonal to it (Appendix H): a fresh random direction at every step (fresh noise), a random direction drawn once and held for the whole run (fixed bias), or that fixed direction scaled down as the training loss falls (error-gated bias). All six signals are passed to Adam like the reference gradient. The rules that memorize do not enter the circuit comparisons (Section 4.3).

4 RESULTS: CREDIT ALLOCATION CHANGES CIRCUIT CONCENTRATION

Measuring circuit concentration. We ask how a learned computation is spread across the neurons of a layer in the networks that grok. On modular addition, algorithm checks for all the tested learning rules that grok show the same periodic Fourier computation and not the alternative computation reported by Zhong et al. (2023) (Appendix B.3). The checks cover the input frequencies used, a trigonometric fit of the logits, and agreement across seeds and rules; for the reference and the equalized network, the fitted trigonometric function alone classifies every held-out pair correctly.

We measure circuit concentration from the norms of the weights that connect each hidden neuron to the next layer, since, at a given activation, a neuron with small outgoing weights has only a small effect on the output, whatever it computes. For the n neurons in the measured layer, let $a_j = \|\boldsymbol{W}^{\text{out}}_{:,j}\|_2$ be the norm of the outgoing weights of neuron j. Their *participation ratio* is

$$\text{PR} \;=\; \frac{\left(\sum_j a_j\right)^2}{n \sum_j a_j^2} \;\in\; [1/n, 1], \tag{1}$$

Table 1: Learning rules under test, grouped by how they transform the learning signal. Per-parameter and matrix transformations can change a neuron's update direction and are not equivalent to normalizing a group of neurons. Feedback alignment, direct feedback alignment, sign-SGD and the three constructed rules of Section 3.1 enter only the comparison of Section 4.3 and are not listed. Settings are given in Appendix A.

Learning rule	Transformation of the learning signal
Backpropagation with Adam (reference)	per-parameter adaptive scaling (Kingma & Ba, 2015)
Credit equalization, then Adam	one positive factor per neuron, applied before Adam
LARS	one factor per layer (You et al., 2017)
LAMB	per-parameter scaling and a layerwise trust ratio (You et al., 2020)
RMSProp	per-parameter second-moment scaling (Tieleman & Hinton, 2012)
Lion	elementwise sign of a momentum signal (Chen et al., 2023)
Adafactor	factored second-moment scaling (Shazeer & Stern, 2018)
Muon	matrix orthogonalization (Jordan et al., 2024)
Shampoo	matrix preconditioning (Gupta et al., 2018)
Nero	one factor per neuron and weight constraints (Liu et al., 2021)
DFC	controller-based credit, no gradient rescaling (Meulemans et al., 2021)

for a nonzero vector of norms. If all norms are equal, $\mathrm{PR} = 1$, and if only one is nonzero, $\mathrm{PR} = 1/n$ (Proposition 1, and all propositions are stated and proved in Appendices G and H). Applied to the outgoing norms, as here, we call it the *circuit participation*, and $n \cdot \mathrm{PR}$ the *effective neuron count* (a layer with effective count 154 has the participation ratio of a layer in which 154 neurons carry equal outgoing norms and the rest none). Both are computed from the weights, and Section 4.1 confirms them with two measures tied to the computation. Applied to the norms of the learning signal that the neurons receive (Section 4.2), we call it the *credit participation*.

Backpropagation with Adam produces a concentrated circuit with approximately 154 effective neurons out of 256 in the first hidden layer. In contrast, DFC, which does not use the backpropagated loss gradient, groks with a delay of 520 to 610 epochs (Appendix A) and produces a distributed circuit with approximately 235 effective neurons at nearly perfect test accuracy.

4.1 Equalizing credit across neurons

Backpropagation and DFC differ in several respects, so we modify backpropagation itself and keep everything else fixed.

Let $\boldsymbol{g}_j$ be the gradient of neuron j's incoming weights. Its norm $\|\boldsymbol{g}_j\|_2$ is the neuron's credit magnitude. The mean credit magnitude over the n neurons of the layer is $\bar{g} = \frac{1}{n}\sum_{k=1}^{n} \|\boldsymbol{g}_k\|_2$. The intervention moves each magnitude toward the mean with strength $\beta \in [0, 1]$,

$$\widetilde{\boldsymbol{g}}_j \;=\; \frac{\boldsymbol{g}_j}{\|\boldsymbol{g}_j\|_2}\Big((1-\beta)\,\|\boldsymbol{g}_j\|_2 + \beta\,\bar{g}\Big) \quad \text{for } \|\boldsymbol{g}_j\|_2 > 0, \tag{2}$$

and leaves zero gradients at zero. At $\beta = 0$ the gradient is unchanged and training is ordinary backpropagation with Adam. At $\beta = 1$ all nonzero incoming-weight gradients have the same norm. We call this *full credit equalization*. The direction of every neuron's incoming-weight gradient is preserved, so before it enters Adam the rescaled gradient is still a descent direction for the loss (Proposition 2, Appendix G). The rescaling acts only on the incoming-weight gradients. Circuit participation is measured from the outgoing weights, which are trained without modification.

Increasing β from 0 to 1 raises circuit participation from 0.60 to 0.99, that is, from approximately 154 to 254 effective neurons (Figure 2a). Training loss, margin (the mean difference between the correct logit and the largest incorrect logit) and parameter norm stay within a few percent of the reference values, and mean test accuracy decreases from 0.998 to 0.976. Changing the credit magnitudes is sufficient to redistribute the circuit under otherwise identical training conditions. The gap widens with width: at width 2048 the reference uses 332 effective neurons and the equalized network 2046 (Appendix F.2).

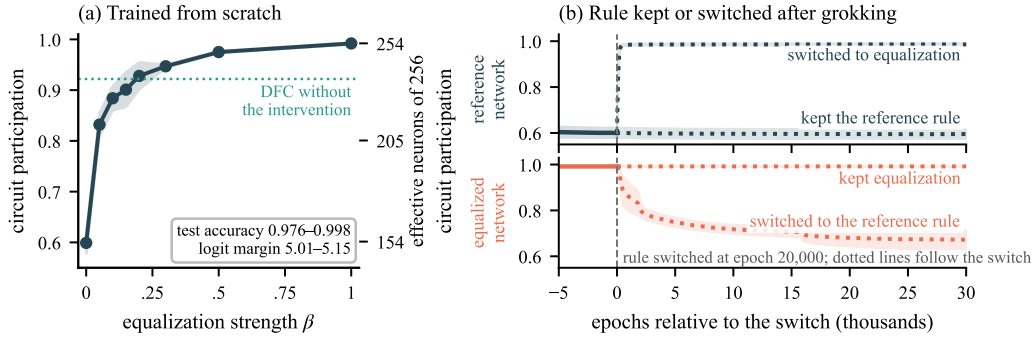


Figure 2: Credit allocation sets the circuit and reorganizes it after grokking (modular addition, $p = 31$, three seeds, bands one standard deviation). **(a)** Circuit participation and effective neuron count of networks trained from scratch at equalization strength β. Training cross-entropy (0.040), margin and parameter norm stay close to the reference values. **(b)** A network grokked with the reference rule and one grokked with credit equalization, each copied at epoch 20,000 and continued for 30,000 epochs with its own rule or with the other rule. Test accuracy remains above 0.95 throughout.

Two measures that use each neuron's activity or its effect on the output also increase. Over eight seeds, participation computed from outgoing norms weighted by mean absolute activation rises from 0.34 to 0.81. Participation computed from the absolute mean change in the correct-class logit after removing each neuron rises from 0.42 to 0.80 (Appendix B.1). The activation-weighted measure is invariant to compensating rescalings of a ReLU neuron's weights.

We next ask whether the intervention can redistribute a circuit that has already been learned. We let the reference network grok, then switch to full credit equalization and continue training on the same task. Outgoing-weight participation rises from 0.60 to above 0.95 within 300 epochs and then to 0.99, while test accuracy remains above 0.98 (Figure 2b). The effect of the neurons on the output spreads as well, from ablation participation 0.40 to 0.72 over five seeds (Appendix E.3). A control that continues with the reference rule stays at 0.60, and in separate runs both rules hold their participation over 200,000 epochs (Appendix C.4).

Controls in Appendix C exclude alternative explanations. Constraining the incoming weight norms instead of the credit gives participation 0.66. A grid over learning rate and weight decay stays between 0.53 and 0.63. SGD with momentum in place of Adam gives the same rise from 0.53 to 0.99 (Appendix C.3). Dropout also distributes the circuit and tolerates neuron deletion about as well, but it changes the objective and raises margin and norm (Appendix C.3). Proposition 3 (Appendix G) shows why in a simplified linear model. Without equalization each unit grows in proportion to its size, so a unit that starts twice as large stays twice as large. With equalization every unit grows by the same amount, so initial differences fade. The model rescales both weights of a unit, whereas our intervention rescales only the incoming gradient, so it gives the mechanism, not the numbers.

4.2 Which parameters are normalized together?

Full credit equalization is itself a normalization of the gradient with one factor per neuron. Optimizers normalize the learning signal at other scales: Adam normalizes each weight separately, LARS rescales a whole layer by one factor, and Nero normalizes the gradient of the weights into each neuron. Before comparing optimizers, we ask in a controlled setting whether normalizing several neurons together has the same effect as normalizing each one alone. We divide the n neurons of the first hidden layer into groups of m neurons and rescale all gradients in a group by one common factor, chosen so that every group has the same mean credit,

$$\widetilde{\boldsymbol{g}}_j = \boldsymbol{g}_j \frac{\bar{g}}{\bar{g}_{B(j)}}, \qquad \bar{g}_{B(j)} = \frac{1}{m} \sum_{k \in B(j)} \|\boldsymbol{g}_k\|_2, \tag{3}$$

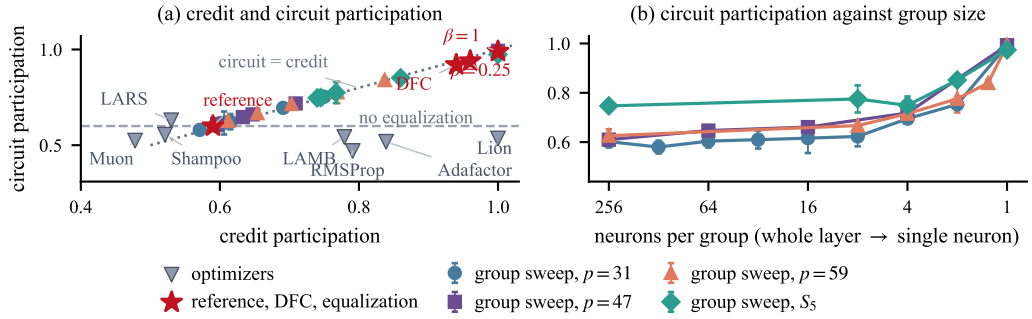


Figure 3: Circuit participation under different transformations of the learning signal. **(a)** Circuit participation against credit participation for the reference, DFC, credit equalization at $\beta = 0.25$ and $\beta = 1$, the group-size sweep on four tasks and seven optimizers. Credit participation is measured on the signal listed in Table 3: the learning signal before the optimizer for the reference, DFC, the equalization conditions and LARS, and the applied update for the other optimizers. **(b)** Circuit participation against the number of neurons that share one normalization factor, on modular addition at three moduli and on S_5. Error bars show one standard deviation across three seeds.

where $B(j)$ is the group containing neuron j and $\bar{g}$ is the layer mean from Equation 2. With $m = 1$ every neuron is rescaled separately and the rule is full credit equalization ($\beta = 1$). With $m = n$ the factor equals one and the gradient is unchanged.

For groups of eight or more neurons, circuit participation stays within about 0.02 of the value 0.60 of the reference, backpropagation with Adam. It rises to 0.70 for groups of four, 0.75 for pairs and 0.99 for single neurons (Figure 3b). Most of the redistribution occurs only when each neuron receives its own factor. A common factor multiplies all gradients in a group by the same number. This equalizes the mean credit between groups but keeps the ratios inside a group: a neuron that received twice the credit of its neighbor still does after the rescaling. In a simplified model, a linear network in which every unit has one incoming and one outgoing scalar weight, these within-group differences persist throughout training and keep the circuit concentrated. Only single-neuron groups leave none to preserve (Proposition 4 in Appendix G, sweep details in Appendix D.1).

Comparison with optimizers. If per-neuron normalization is what distributes the circuit, optimizers should sort accordingly. We train with the eight optimizers of Table 1 that replace Adam. LARS, RMSProp, LAMB, Adafactor, Lion, Muon and Shampoo normalize groups other than a single neuron or change update directions, and all seven produce concentrated circuits, with participation between 0.470 and 0.613. Nero, at its default learning rate of 10^{-2}, produces a distributed circuit with participation 0.986 at test accuracy 0.985 (Table 11). Nero, which normalizes credit per neuron, and DFC, whose credit is nearly equal (credit participation 0.94), are cases of equal credit that arise without our intervention, and both produce distributed circuits.

For the reference, DFC and the equalization sweeps, credit participation lies close to circuit participation (Figure 3a). Equal final update norms, however, are not sufficient to distribute the circuit. Lion gives every neuron an update of the same norm, yet its circuit remains concentrated. These equal norms arise from the sign of each coordinate, which also changes the update direction, whereas credit equalization preserves it. Per-parameter scaling, as in Adam, is not a finer version of per-neuron scaling: it rotates each neuron's update and leaves the norms across neurons unequal (participation 0.66 for Adam's step, Appendix C.3).

4.3 Which learning rules grok

We say that a learning rule groks or memorizes when the networks trained with it do. On modular addition, backpropagation with the optimizers in Table 1, credit equalization, DFC and the two constructed rules with a fresh or a fading orthogonal component grok. Feedback alignment, direct feedback alignment, sign-SGD and the rule with a fixed orthogonal component fit the training set

Table 2: Deletion tolerance and adaptation of the concentrated and the distributed circuit on modular addition ($p = 31$). Deletion rows are means over three seeds (Appendix E.1). Adaptation rows are means over ten seeds for subtraction, including from scratch, and over five seeds for multiplication, with standard deviations in parentheses (Table 16).

	Concentrated ($\beta = 0$)	Distributed ($\beta = 1$)	From scratch
Random deletion tolerance	0.674	0.770	—
Norm-ranked deletion tolerance	0.332	0.581	—
Greedy deletion, neurons removed	23	61	—
Adaptation to subtraction, epochs	640 (128)	1540 (180)	1410 (137)
Adaptation to multiplication, epochs	1800 (141)	1940 (136)	1200 (63)

and stay near zero test accuracy over the 15,000-epoch budget. According to the weight-decay explanation of grokking (Nanda et al., 2023; Varma et al., 2023), once the training data are fitted, weight decay shrinks the weights along the directions in which small parameter changes leave the training outputs unchanged. Weight along these directions is invisible to the training loss to first order at the current parameters, but it can still change the outputs on held-out inputs, and a learning signal that is not a loss gradient can hold weight there against the decay. Let J be the Jacobian of the training logits with respect to the parameters θ and P the orthogonal projector onto its null space. Then $q = \|P\theta\|_2$ is the norm of the weight component along these unconstrained directions (Appendix H). Across nine configurations with three seeds each, q separates the two outcomes without overlap (Figure 1a). At the end of training, the measured means lie between 5.3 and 8.5 for the rules that grok and between 26 and 135 for the rules that memorize. The separation is present long before generalization: after 2% of training, when every run has perfect training accuracy and zero test accuracy, the largest q among the runs that later grok (22.0) lies below the smallest among those that memorize (27.7), whereas the parameter norm does not yet separate the two groups. For every rule we tested, the outcome is the one the weight-decay account predicts: in the rules that grok this weight shrinks during training, and in the rules that memorize it persists or grows.

5 Relevance: Distributed circuits tolerate neuron deletion, concentrated circuits adapt faster

Throughout this section the concentrated circuit is the reference ($\beta = 0$) and the distributed circuit is the fully equalized network ($\beta = 1$), both trained on modular addition at $p = 31$.

5.1 Tolerance to neuron deletion

A computation spread over many neurons should depend less on any single one of them (Georgopoulos et al., 1986), and networks that generalize well are known to rely less on single units (Morcos et al., 2018). In an additive model, the relative fluctuation of the output under random deletion falls with the effective neuron count (Proposition 5, Appendix G). We test this with three deletion procedures, all applied to the first hidden layer of the trained network without retraining. Random deletion removes a random subset of the neurons. Norm-ranked deletion removes neurons in decreasing order of outgoing norm, so the neurons a concentrated circuit relies on go first. Greedy deletion removes, at each step, the neuron whose removal lowers test accuracy the most. For random and norm-ranked deletion we report the *neuron-deletion tolerance*, or *deletion tolerance* for short, the normalized area under the curve of test accuracy against the deleted fraction from 0 to 0.8. For greedy deletion we report the number of neurons removed before test accuracy falls below 0.5 (Appendix E.1).

The distributed circuit is more tolerant under all three procedures (Table 2). The advantage is largest under norm-ranked deletion, as expected if the concentrated circuit rests on few high-norm neurons, and it persists under greedy deletion. In this network, the tolerance follows the distribution of outgoing norms rather than the rule that produced it: an outgoing-norm constraint that holds participation at 0.99 without changing the gradients gives the same random-deletion tolerance as equalization (Appendix E.2). Nero reaches similar deletion tolerances without the intervention (Appendix D.2).

5.2 Adaptation to a new operation

Does circuit concentration affect how quickly the network learns a related task? We take networks that have grokked addition at each β and fine-tune the whole network on a new operation with backpropagation with Adam. The *adaptation time* is the number of fine-tuning epochs until test accuracy on the new operation reaches 0.9. We compare it with training a new network from scratch.

We first transfer to subtraction, which uses the same Fourier frequencies as addition and differs in how they are combined. Training from scratch takes 1410 epochs. Starting from the concentrated addition circuit takes 640 epochs, so the pretrained circuit helps. Starting from the distributed circuit takes 1540 epochs, no faster than from scratch, and the adaptation time increases monotonically with β in between (Appendix E.3). The concentrated circuit adapts about 2.4 times as fast as the distributed one. The ordering holds at one third and at three times the fine-tuning learning rate, in the reverse transfer from subtraction to addition, when an outgoing-norm constraint sets the participation of the source network, and when both networks are fine-tuned with credit equalization instead of the reference rule (Appendices E.2 and E.3).

The transfer difference also appears when both conditions begin from the same grokked checkpoint. After matched continuation budgets, the equalized branches require 1052 epochs to learn subtraction, compared with 608 for reference continuations, a paired difference of $444\,(131)$ epochs, and adapt more slowly in all five seed pairs.

We repeat the experiment with multiplication, which does not share the Fourier frequency representation of addition. Every pretrained network takes 1800 to 2020 epochs, compared with 1200 from scratch, so pretraining on addition does not help at any β.

6 Discussion and conclusions

Circuit concentration is a consequence of the training dynamics and not of the task alone. The concentrated circuits found under backpropagation with Adam are one of several ways a grokked solution can be implemented, and among the optimizers we tested the concentration follows whether the optimizer rescales each neuron's credit by one factor of its own.

For mechanistic interpretability this means that the concentration of a circuit found in a trained network reflects the learning rule as well as the task. A sparse circuit read out of a network trained with Adam is a property of that training run, and comparing learning rules separates what the task requires from what the optimizer selected. The learning rule also decides whether grokking happens at all: we find that the rules that memorize keep weight in parameter directions along which the training outputs do not change locally, weight the training loss does not see to first order, and this tells the two outcomes apart after 2% of training. For the nine rules tested, a fraction of a run was enough to tell which of them would generalize. DFC groks, which shows that delayed generalization does not depend on a backpropagated loss gradient.

The trade-off between the two circuits has practical consequences. Tolerance to neuron deletion is what a hardware fault would test, and a distributed circuit loses less when units are removed. Fast adaptation to a related task is what fine-tuning relies on, and there the concentrated circuit wins. On modular addition, the choice of learning rule trades robustness to neuron loss against speed of adaptation, without changing the learned algorithm.

Limitations. The theory isolates the distribution mechanism in a simplified linear model and covers rescalings that preserve update directions. The results on nonlinear networks and on optimizers are empirical. The algorithm checks cover modular addition only. The adaptation results use one pair of related operations and one unrelated pair. The robustness advantage concerns the deletion of neurons.

AI use statement

We used generative AI tools to assist with experiment and analysis code, to edit text written by the authors, to check related-work coverage, and to check the derivations and proofs in the appendix. The checks of the derivations and proofs did not lead to changes. The authors developed the re-

search ideas, experimental design, derivations, interpretations, and conclusions. We reviewed all AI-assisted work, read the code and tested it against its results, and checked suggested references and formulations. We take responsibility for the final text, claims and artifacts, including material developed with generative AI assistance.

Reproducibility statement

Section 3 and Appendix A specify the tasks, networks, learning rules and training settings. The code for all experiments, tables and figures will be published on GitHub after acceptance.

References

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022. arXiv:2207.08799.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning (ICML)*, pp. 560–569, 2018. arXiv:1802.04434.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 49205–49233, 2023. arXiv:2302.06675.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning (ICML)*, pp. 6243–6267, 2023. arXiv:2302.03025.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632:768–774, 2024. doi: 10.1038/s41586-024-07711-7.

Apostolos P. Georgopoulos, Andrew B. Schwartz, and Ronald E. Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

Suriya Gunasekar, Jason D. Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning (ICML)*, pp. 1832–1841, 2018. arXiv:1802.08246.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning (ICML)*, pp. 1842–1850, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.06110.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. NorMuon: Making Muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7 (1):13276, 2016.

Yang Liu, Jeremy Bernstein, Markus Meister, and Yisong Yue. Learning by turning: Neural architecture aware optimisation. In *International Conference on Machine Learning (ICML)*, pp. 6748–6758, 2021.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.01117.

Tao Luo, Zhi-Qin John Xu, Zheng Ma, and Yaoyu Zhang. Phase diagram for two-layer ReLU neural networks at infinite-width limit. *Journal of Machine Learning Research*, 22(71):1–47, 2021.

Neil Rohit Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: Grokking modular arithmetic via average gradient outer product. In *International Conference on Machine Learning (ICML)*, pp. 42834–42856, 2025. arXiv:2407.20199.

Dmitry Manning-Coe, Jacopo Gliozzi, Alexander G. Stapleton, Edward Hirst, Giuseppe De Tomasi, Barry Bradlyn, and David S. Berman. Grokking vs. learning: Same features, different encodings. *arXiv preprint arXiv:2502.01739*, 2025.

William Merrill, Nikolaos Tsilivis, and Aman Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks. *arXiv preprint arXiv:2303.11873*, 2023.

Alexander Meulemans, Matilde Tristany Farinha, Javier García Ordoñez, Pau Vilimelis Aceituno, João Sacramento, and Benjamin F. Grewe. Credit assignment in neural networks through deep feedback control. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 4674–4687, 2021.

Alexander Meulemans, Nicolas Zucchet, Seijin Kobayashi, Johannes von Oswald, and João Sacramento. The least-control principle for local learning at equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 33603–33617, 2022.

Ari S. Morcos, David G. T. Barrett, Neil C. Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR)*, 2018.

Depen Morwani, Benjamin L. Edelman, Costin-Andrei Oncescu, Rosie Zhao, and Sham Kakade. Feature emergence via margin maximization: Case studies in algebraic tasks. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2311.07568.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Maria Refinetti, Stéphane D'Ascoli, Ruben Ohana, and Sebastian Goldt. Align, then memorise: the dynamics of learning with feedback alignment. In *International Conference on Machine Learning (ICML)*, pp. 8925–8935, 2021.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, 2016.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning (ICML)*, pp. 4596–4604, 2018.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32145–32168, 2023.

Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. *arXiv preprint arXiv:2206.04817*, 2022.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. Lecture slides, COURSERA: Neural Networks for Machine Learning, 2012.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023.

James C. R. Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local Hebbian synaptic plasticity. *Neural Computation*, 29(5):1229–1262, 2017.

Blake Woodworth, Suriya Gunasekar, Jason D. Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory (COLT)*, pp. 3635–3673, 2020. arXiv:2002.09277.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1904.00962.

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)*, volume 36, pp. 27223–27250, 2023. arXiv:2306.17844.

A Experimental details

Statistical reporting. Results are means across three random seeds, with standard deviations in parentheses, unless stated otherwise. Conditions that are compared with each other use the same seeds and data splits. Independent experiments have their own reference runs, so reference values vary slightly between tables. For comparisons between credit and circuit participation we take the absolute difference within each seed and then average.

Networks and training. Training is full batch. Hidden weights use Kaiming normal initialization (He et al., 2015). Adam uses learning rate 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and coupled weight decay, in which the decay term is added to the gradient before the moment estimates; biases are trained and decayed like the weights. For the learning signals passed to Adam, the gradient norm is clipped to 1.0 after any rescaling and before the optimizer step, and Adam adds the decay term to the clipped gradient. The optimizers compared in Section 4.2 apply no clipping. In the multilayer perceptrons, credit equalization is applied to the incoming-weight gradients of the first hidden layer (bias gradients are not rescaled), and circuit participation and neuron deletion are measured on the outgoing weights of the same neurons. The transformer experiments apply it to every linear map (Appendix F.5). Seeds start at 0, so three-seed experiments use seeds 0 to 2. Each run uses one GPU. Runs continue past grokking until participation has stabilized, which takes $12{,}000$ to $70{,}000$ epochs depending on task, modulus and architecture; compared conditions use the same budget, and the experiments with other budgets state them.

Modular-arithmetic data. We hold out 40% of the p^2 operand pairs at random and train on the rest, with the same split procedure for all three operations. Multiplication includes the pairs with a zero operand, identically across rules. We test $p \in \{31, 47, 59\}$ and use $p = 31$ unless stated otherwise.

Optimizer settings. The comparison of Section 4.2 runs every optimizer at the reference learning rate 10^{-3} and weight decay 10^{-3} for 20,000 epochs. LARS uses fifty times this learning rate. Nero uses its default learning rate 10^{-2} and no weight decay, since its norm constraint fixes the weight scale; it normalizes each neuron's gradient by a running estimate of its norm and holds each neuron's incoming-weight norm fixed after its first update. Lion adds weight decay to the gradient before the signed momentum update. Muon uses five cubic Newton–Schulz iterations, ordinary momentum and coupled weight decay on matrices. LARS uses a layerwise norm ratio without momentum. LAMB uses uncorrected moments and caps the trust ratio at 10. Adafactor uses fixed-rate factored second moments without update clipping, Shampoo uses exponential second moments, and RMSProp uses second-moment decay 0.99 without momentum. Evaluated every 50 epochs over three seeds, every optimizer fits the training set first and exceeds test accuracy 0.95 at least 500 epochs later, so all of them grok under the criterion of Section 3: the delays are 500 to 650 epochs for LARS, 650 to 700 for Lion, 1200 to 1350 for RMSProp and Adafactor, 2150 to 2400 for LAMB, 7450 to 7800 for Muon, 12,200 to 13,100 for Shampoo and 4450 to 7000 for Nero, against 1300 to 1550 for the reference.

Deep feedback control. DFC uses a controller during training that moves the network output toward a target. The output $\hat{\boldsymbol{y}}$ is the linear readout, that is, the logits, and the label $\boldsymbol{y}$ is one-hot. The output error is $\boldsymbol{e} = \boldsymbol{y}^* - \hat{\boldsymbol{y}}$, where the target $\boldsymbol{y}^* = \hat{\boldsymbol{y}} + 2\kappa(\boldsymbol{y} - \hat{\boldsymbol{y}})$ lies between the current output and the label at a distance set by the control strength κ; it is one step on the squared error to the label, as in the reference implementation. A leaky proportional-integral controller turns the error into a control signal $\boldsymbol{u}$, which is fed back into every layer i through the transpose of the Jacobian $\boldsymbol{J}_i$ from that layer to the output:

$$\boldsymbol{u}_{\mathrm{int}} \leftarrow \boldsymbol{u}_{\mathrm{int}} + \mathrm{dt}\,(\boldsymbol{e} - \alpha \boldsymbol{u}), \qquad \boldsymbol{u} = \boldsymbol{u}_{\mathrm{int}} + k_p \boldsymbol{e}, \tag{4}$$

$$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \tau(\boldsymbol{W}_i \boldsymbol{r}_{i-1} + \boldsymbol{b}_i + \boldsymbol{J}_i^\top \boldsymbol{u} - \boldsymbol{v}_i), \qquad \boldsymbol{r}_i = \phi(\boldsymbol{v}_i). \tag{5}$$

We use $\mathrm{dt} = 0.02$, $\tau = \mathrm{dt}/0.2 = 0.1$, where 0.2 is the ratio of the neuronal to the controller time constant, $k_p = 2.0$ and $\alpha = 1.7 \times 10^{-3}$, allow at most 500 controller iterations, and stop when the change in the control signal falls below 10^{-4}. The control strength is $\kappa = 0.3$ for modular arithmetic and 0.5 for parity. The modular-arithmetic setting was selected on a grid over weight decay $\{10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$ and $\kappa \in \{0.1, 0.2, 0.3\}$. At equilibrium each layer applies the local update

$$\Delta \boldsymbol{W}_i \;\propto\; (\boldsymbol{r}_i - \boldsymbol{r}_i^{\mathrm{ff}})\, \boldsymbol{r}_{i-1}^\top, \tag{6}$$

the difference between the controlled activity $\boldsymbol{r}_i$ and the feedforward activity $\boldsymbol{r}_i^{\mathrm{ff}}$, multiplied by the presynaptic activity. Its negative, $-2\,(\boldsymbol{r}_i - \boldsymbol{r}_i^{\mathrm{ff}})\, \boldsymbol{r}_{i-1}^\top$ averaged over the batch, is written into the gradient slot, so that Adam's descent step moves the weights along $\Delta \boldsymbol{W}_i$; it is clipped and passed to Adam like the other learning signals in this paper. Feedback through the transpose Jacobian follows the reference implementation of Meulemans et al. (2021); the feedback pathway is therefore computed from the forward network. DFC shows the same delayed generalization as the reference (Figure 4). Evaluated every 10 epochs, training accuracy reaches 1.0 at epochs 90 to 130 for DFC and at epoch 100 for the reference, and test accuracy first exceeds 0.95 at epochs 650 to 710 for DFC and 1220 to 1450 for the reference; the grokking delays are 520, 600 and 610 epochs for DFC and 1120 to 1350 for the reference over the three seeds, all above the 500-epoch criterion of Section 3.

Measuring credit participation. Credit participation applies Equation 1 to the row norms of a neuron-level learning signal for the first hidden layer, one row per neuron, computed at every epoch of the last 800 epochs and then averaged; it is the mean of instantaneous participation ratios. Which signal is logged depends on the condition (Table 3). For the rules that hand a gradient-like signal to Adam it is that signal before clipping; for the adaptive optimizers, whose rescaling cannot be separated from their running state, it is the update applied to the first-layer weights. Circuit participation is measured from the outgoing weights of the trained network, on different parameters, so the agreement between credit and circuit participation is an empirical result and not an identity. Table 8 also reports participation measured from Adam's effective step $m/\sqrt{v}$.

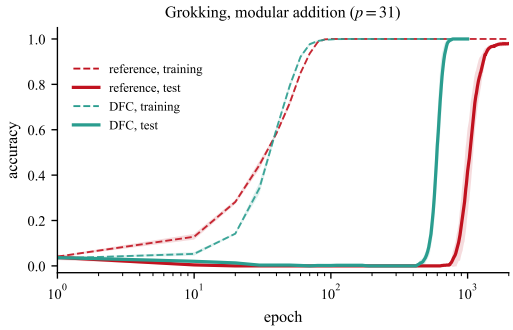


Figure 4: Training (dashed) and test (solid) accuracy for the reference and DFC on modular addition ($p = 31$), three seeds with shared initializations, evaluated every 10 epochs; bands show one standard deviation across seeds, and the DFC runs end at epoch 1000. Both fit the training set first and generalize hundreds of epochs later.

Table 3: Signal on which credit participation is measured, per condition. All signals are for the first hidden layer, with one row per neuron.

Condition	Logged signal
Reference; LARS	raw loss gradient, before clipping and without the weight-decay term; for LARS this differs from the applied update only by the common layer factor and the weight-decay term
Credit equalization; group-size sweeps; imposed profiles	rescaled gradient of Equation 2 or 3, before clipping and before Adam
DFC	local controller update of Equation 6 for the first layer, before Adam
RMSProp, LAMB, Adafactor, Lion, Muon, Shampoo	update applied to the first-layer weights: gradient plus weight-decay term, transformed with the optimizer's running moments (Lion: sign of the momentum-interpolated gradient; LAMB: including the trust ratio)

B Circuit measurements and algorithm checks

B.1 Do outgoing norms reflect the distribution of computation?

We use the participation ratio of outgoing norms rather than a Fourier-based sparsity measure because it needs no Fourier basis, applies unchanged to every task we study, and is invariant to a uniform rescaling of the outgoing weights. It is, however, a structural measure. The network's function is unchanged when a ReLU neuron's incoming weights and bias are multiplied by a positive constant and its outgoing weights are divided by the same constant, so outgoing norms could in principle change without any change in the computation. We check the redistribution with two measures that are tied to the computation.

The first weights each neuron's outgoing norm by its mean absolute activation, $\mathbb{E}_x|h_j(x)|\,\|\boldsymbol{W}^{\text{out}}_{:,j}\|_2$, where $h_j(x)$ is the activation of neuron j on input x and the mean runs over the test inputs; this product is invariant to the compensating rescaling above. The second removes each neuron j in turn and records $|\mathbb{E}_x[z_y(x) - z_y^{(-j)}(x)]|$, the absolute value of the mean change of the correct-class logit z_y over the same inputs, where $z_y^{(-j)}$ is the logit with neuron j removed; we apply Equation 1 to these ablation effects. Over eight seeds, activation-weighted participation increases from 0.343 under the reference to 0.813 under full credit equalization, and ablation participation from 0.422 to 0.801. The mean absolute change of the logit, $\mathbb{E}_x|z_y(x) - z_y^{(-j)}(x)|$, and of the margin give the same separation: 0.34 to 0.49 under the reference against 0.76 to 0.98 under equalization for the logit, and 0.38 to 0.48 against 0.85 to 0.97 for the margin (three seeds).

Low-norm neurons account for only part of the difference. We recompute participation among the neurons whose outgoing norm is at least one tenth of the layer mean. The reference has 84 neurons

below this threshold and participation 0.889 (0.006) among the rest; full equalization has no neuron below the threshold and participation 0.991 (0.001). Under equalization no neuron falls below the threshold, and the norms above it are also more uniform. It does not make the neurons less selective: the mean fraction of a neuron's input-weight Fourier power at its strongest frequency rises from 0.667 under the reference to 0.884 at $\beta = 1$ (five seeds, the experiment of Table 14).

B.2 Training fit and output scale

Participation could change because the training fit or the output scale changes. We therefore report the parameter norm $\|\theta\|$ and the logit margin $\gamma = \mathbb{E}_x[z_y(x) - \max_{c \neq y} z_c(x)]$, where $z_c(x)$ is the logit of class c for input x, y is the correct label of x and the mean runs over the training set, for the reference, full credit equalization and DFC (Table 4). The reference and the equalized network have nearly the same norm and margin despite their different participation, so the redistribution is not accompanied by a change in fit or output scale. DFC has a much smaller norm and margin: its outputs live on a different scale, which is one reason we use the credit intervention rather than DFC for the matched comparison.

Table 4: Parameter norm and logit margin at matched weight decay (modular addition, $p = 31$, four seeds; means and standard deviations). All runs have test accuracy ≥ 0.97. Credit equalization leaves both quantities close to the reference values while producing a circuit at least as distributed as DFC's.

Rule	$\|\theta\|$	margin γ
Reference (backpropagation with Adam)	26.10 ± 0.15	5.02 ± 0.02
Credit equalization (β=1)	26.65 ± 0.10	5.15 ± 0.03
DFC	13.09 ± 0.02	0.54 ± 0.00

B.3 Checks of the modular-addition computation

Prior work derives the computation that grokked networks use for modular addition: the first layer represents each operand by a few frequencies k, hidden neurons combine them into $\cos$ and $\sin$ of $k(a + b)$, and the logit for candidate c sums $\cos(2\pi k(a + b - c)/p)$ over frequencies, which peaks at $c = a + b$ (Nanda et al., 2023; Gromov, 2023; Chughtai et al., 2023; Morwani et al., 2024). We check whether the reference, DFC and full credit equalization all implement this computation, and then repeat the checks for the remaining grokking rules.

Fourier measurements. Let $\boldsymbol{u}_j, \boldsymbol{v}_j \in \mathbb{R}^p$ be the halves of neuron j's incoming weights that act on the two operands, and let $\hat{u}_j, \hat{v}_j$ be their discrete Fourier transforms. The power of neuron j at frequency $k = 1, \ldots, \lfloor p/2 \rfloor$ is

$$s_j(k) \;=\; |\hat{u}_j(k)|^2 + |\hat{v}_j(k)|^2, \qquad \hat{u}_j(k) = \textstyle\sum_{a=0}^{p-1} u_j[a]\, e^{-2\pi i k a/p}. \tag{7}$$

The aggregate spectrum of a layer sums $s_j(k)$ over neurons.

Frequency ablation. We keep only the strongest input frequencies, ranked by aggregate power, and count how many are needed to recover 95% of the full test accuracy: 6.7 for backpropagation with Adam, our example of a concentrated circuit, and 9.3 for DFC, our example of a distributed circuit. Both networks rely on a small set of input frequencies. A comparison between the reference and full credit equalization, with a random control, follows.

Frequency retention and removal under credit equalization. We repeat the frequency ablation for the reference and full credit equalization over five seeds and add removal and a random control. Frequencies are ranked once per network by the summed power of its first-layer weights, so the ranking uses no data. We keep or remove the top k frequencies for every k from 0 to 15, always keep the constant component, and filter both operand blocks alike. The control keeps or removes k frequencies drawn at random, averaged over 20 draws per network. For both rules, keeping the top frequencies preserves more test accuracy than keeping random ones, and removing them costs more

than removing random ones, at every k from 2 to 14 in every seed (Table 5, Figure 5). The equalized network depends on its leading frequencies as the reference does, and spreads that dependence over more of them. Fitted on the training pairs only, the trigonometric function alone classifies every held-out pair correctly in every seed for both rules, with a mean margin between 6.1 and 6.8.

Table 5: Test accuracy after keeping or removing k input frequencies, ranked by first-layer weight power (top) or drawn at random (20 draws per network), modular addition at $p = 31$. Means over five seeds; the complete curves are in Figure 5.

		Frequencies kept or removed, k					
		2	4	6	8	10	12
Reference	Keep top k	0.23	0.57	0.86	0.97	1.00	1.00
	Keep random k	0.10	0.21	0.40	0.58	0.76	0.90
	Remove top k	0.83	0.44	0.19	0.06	0.02	0.02
	Remove random k	0.94	0.83	0.62	0.44	0.27	0.13
Equalized	Keep top k	0.14	0.42	0.69	0.86	0.93	0.96
	Keep random k	0.10	0.23	0.43	0.64	0.80	0.92
	Remove top k	0.91	0.73	0.46	0.23	0.10	0.03
	Remove random k	0.94	0.87	0.71	0.50	0.32	0.16

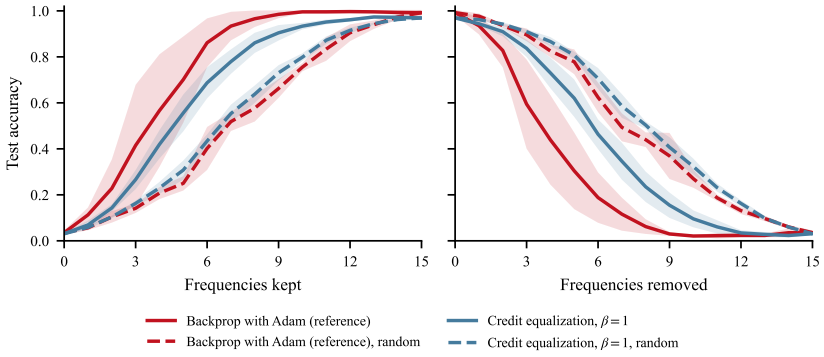


Figure 5: Test accuracy after keeping (left) or removing (right) k input frequencies, ranked by first-layer weight power (solid) or drawn at random (dashed). Means over five seeds, bands one standard deviation across seeds.

Trigonometric fit. We regress the full logit tensor onto the $(a + b - c)$ trigonometric basis of the Fourier algorithm. The fit explains 0.57 of the logit variance for the reference, 0.42 for full credit equalization and 0.90 for DFC. The residual of the reference agrees across initializations with a correlation of only 0.235, so there is little evidence of a second computation shared across seeds.

Agreement across runs. With a fixed data split, seeds of the same rule differ only in their initialization, and their agreement is the yardstick for comparisons across rules. Logit tensors of reference seeds correlate at 0.446; correlations across rules lie between 0.427 and 0.467. Linear centered kernel alignment between the first-hidden-layer activations of the reference and of full equalization is 0.69, compared with 0.57 between reference seeds and 0.74 between equalized seeds. Agreement across rules is thus comparable to agreement across seeds of one rule.

The published alternative. The pizza algorithm of Zhong et al. (2023) multiplies the clock logit by an envelope that depends on the difference of the operands, $|\cos(\pi k(a-b)/p)| \cdot \cos(2\pi k(a+b-c)/p)$. Its basis overlaps with the trigonometric one: on a synthetic pizza tensor the trigonometric basis explains 0.81 of the variance, and on a synthetic clock tensor the pizza basis explains 0.81, so explained variance alone does not tell the two apart. Two statistics do. Adding the pizza basis to the trigonometric basis raises the explained variance by at most 0.0012 in every seed of every rule,

against 0.19 for a synthetic pizza tensor and 0.001 for noise. The distance irrelevance of Zhong et al. (2023), the variability of the correct logit at a fixed operand difference relative to its total variability, lies between 0.976 and 0.990 for every seed of every rule; they report 0.85 for a clock network and 0.17 for a pizza network, and classify networks below 0.6 as pizza. None of the rules implements this alternative.

The remaining grokking rules. We repeat the checks for the other learning rules that grok: the eight optimizers of Table 11, trained as in Appendix A, and the two constructed rules with a fresh or a fading orthogonal component, trained as in Appendix H, with three seeds each. As above, the seeds share one data split and differ in their initialization, and a reference trained in the same way provides the yardstick (Table 6). In every rule the trigonometric basis explains between 0.42 and 0.68 of the logit variance per seed, the distance irrelevance lies above 0.97, and the logit correlation and the linear centered kernel alignment of first-hidden-layer activations with the reference across initializations lie at the level of the reference's agreement with itself. Together, the checks show the same periodic computation across all tested grokking rules and distinguish it from the published alternative.

Table 6: Algorithm checks for the remaining grokking rules (modular addition, $p = 31$, three seeds, one shared data split). Trig: fraction of logit variance explained by the $(a + b - c)$ basis. Distance irrelevance: the statistic of Zhong et al. (2023), above 0.6 for clock networks and below 0.6 for pizza networks. Correlation and CKA: agreement with the reference trained in the same way, across initializations; the reference rows give its agreement with itself. The optimizers are trained for $20{,}000$ epochs and the constructed rules for $15{,}000$, each with its own reference. The seeds share one data split, so the values differ slightly from Table 11, whose seeds use different splits.

Rule	Test	Circuit PR	Trig	Dist. irrel.	Correlation	CKA
Optimizers, 20,000 *epochs*						
Reference	0.999	0.558	0.535	0.983	0.407	0.590
LARS	0.975	0.604	0.550	0.986	0.419	0.592
RMSProp	0.995	0.496	0.517	0.984	0.456	0.598
LAMB	0.998	0.566	0.463	0.984	0.396	0.573
Adafactor	0.998	0.489	0.553	0.983	0.432	0.580
Lion	0.998	0.479	0.558	0.983	0.461	0.585
Muon	0.993	0.494	0.498	0.980	0.407	0.554
Shampoo	0.992	0.529	0.525	0.980	0.339	0.518
Nero	0.988	0.986	0.455	0.984	0.435	0.659
Constructed rules, 15,000 *epochs*						
Reference	0.998	0.579	0.532	0.983	0.399	0.594
Fresh noise	0.993	0.571	0.531	0.984	0.467	0.621
Error-gated bias	0.984	0.688	0.458	0.980	0.415	0.603

C Controls on credit equalization

The main experiment changes the credit magnitudes and observes a change in the outgoing weights. This section tests the alternative explanations: that the effect comes from the incoming weight norms rather than from the credit, that it depends on which neurons receive the credit, that Adam or the hyperparameters produce it, and that the concentrated circuit is only a transient state.

C.1 Constraining incoming weights

Full credit equalization gives every neuron the same gradient norm, and Nero, which also distributes the circuit, additionally constrains the weight norms. Could equal incoming weight norms alone explain the redistribution? We test two constraints on the weights that leave the gradients unchanged. The first renormalizes each neuron's incoming weights to a fixed norm after every update. Circuit participation is $0.661\ (0.015)$, compared with $0.595\ (0.023)$ for the reference, and deletion tolerance decreases (Table 7, upper block). The second is weight normalization (Salimans & Kingma, 2016), which parameterizes each incoming weight vector as a direction times a learned scale; with weight

decay on the scales it groks to test accuracy 0.993 with participation $0.466\,(0.026)$. Neither constraint approaches the participation of 0.99 reached by credit equalization. Equal incoming weight norms do not reproduce the redistribution.

C.2 Assigning credit independently of neuron identity

In the first hidden layer, a neuron's incoming-weight gradient is the input multiplied by the backpropagated error, and that error depends linearly on the neuron's outgoing weights. Credit and outgoing norms are therefore coupled at every step, and their agreement could reflect this coupling rather than an effect of the credit on the circuit. We break the coupling by assigning credit magnitudes that carry no information about the neurons.

A *credit profile* is a vector of n nonnegative target magnitudes with a chosen participation ratio. At every step we replace the incoming-gradient norms by this profile, scaled so that the total norm is preserved, and keep every gradient's direction. A random permutation, drawn once per run, decides which neuron receives which magnitude, so the assignment is independent of the neurons' roles. Circuit participation follows the imposed value, from 0.637 at profile participation 0.60 to 0.990 at 1.00, deletion tolerance increases with it, and test accuracy stays above 0.96 (Table 7, lower block). An allocation chosen independently of the neurons' current roles therefore sets the final concentration.

A second control keeps the reference's credit magnitudes at each step but randomly reassigns them to the neurons at every step. Each permutation preserves that step's participation and changes only which neurons receive the largest signals. Final circuit participation is 0.992, compared with 0.613 for the reference. Measured over the last 800 epochs, credit participation is 0.979 rather than the reference's 0.604: once the assignment no longer favors the same neurons, the trained network itself develops more uniform gradients. This suggests that concentration requires the credit to stay concentrated on the same neurons over time.

The effect persists across the tested initialization scales. Across scales from 0.5 to 4, reference participation is 0.586 to 0.613 and full equalization gives 0.990 to 0.998, with a mean absolute difference of 0.008 between credit and circuit participation. The effect is also local to the layer in which credit is equalized: equalizing only the second hidden layer gives participation 1.000 there while the first layer stays at $0.593\,(0.004)$ with 86 low-norm neurons, and equalizing both layers distributes both, at test accuracy of at least 0.98.

Table 7: Controls on the weights and on the credit profile ($p = 31$, three seeds; standard deviations in parentheses). Upper block: constraining the incoming weight norms without rescaling the gradients leaves the circuit concentrated. Lower block: credit profiles of a chosen participation, assigned to neurons by a random permutation, produce circuit participation within 0.018 of the imposed value on average; the final row reassigns the reference's credit magnitudes to random neurons at every step. Each block has its own reference run. Random del and norm-ranked del are deletion tolerances (Appendix E.1).

Condition	Test	Credit PR	Circuit PR	Random del	Norm-ranked del
Constraining the incoming weights					
Reference	0.998	0.586	0.595 (0.023)	0.672	0.311
Fixed incoming norm	0.954	0.611	0.661 (0.015)	0.607	0.256
Imposing a credit profile					
Reference	0.997	0.604	0.613 (0.016)	0.673	0.298
Imposed 0.60	0.980	0.600	0.637 (0.003)	0.607	0.240
Imposed 0.70	0.981	0.700	0.727 (0.001)	0.663	0.319
Imposed 0.80	0.972	0.800	0.815 (0.005)	0.693	0.420
Imposed 0.90	0.969	0.900	0.902 (0.005)	0.723	0.533
Imposed 1.00	0.975	1.000	0.990 (0.003)	0.775	0.599
Reshuffled each step	0.974	0.979	0.992 (0.002)	0.775	0.601

C.3 Optimizer and hyperparameter controls

Adam normalizes each parameter separately, so one might suspect that Adam either creates the redistribution or removes it. Neither is the case. With SGD with momentum in place of Adam, circuit participation is 0.53 at $\beta = 0$ and 0.99 at $\beta = 1$, with deletion tolerance 0.60 and 0.76 (Table 8). With Lion in place of Adam, full credit equalization gives participation $0.984\,(0.003)$ at test accuracy $0.974\,(0.010)$, whereas Lion alone gives about 0.5. In the Adam runs, participation measured from Adam's effective step $m/\sqrt{v}$ is 0.66 at $\beta = 0$ and 0.99 at $\beta = 1$, so the difference between neurons survives Adam's transformation.

Ordinary tuning of the reference does not distribute the circuit either. Across three learning rates and two weight-decay values, every setting groks with circuit participation between 0.530 and 0.626 (Table 9). Credit participation ranges from 0.523 to 0.613 and differs from circuit participation by 0.009 on average.

Dropout is the natural comparison for deletion tolerance, since it trains the network under random deletion. It distributes the circuit as well, to participation 0.95 at rate 0.1 and 0.99 at rate 0.5, and its random-deletion tolerance, 0.79 and 0.82, is close to the 0.77 of full credit equalization, which never sees a deleted neuron during training. Two unrelated interventions thus reach the same participation and nearly the same tolerance, which supports the reading of Appendix E.2 that, in this network, deletion tolerance follows the distribution of outgoing norms however that distribution is produced. The interventions differ in what else they change. Dropout modifies the objective and raises the margin and the parameter norm (Table 8), whereas credit equalization changes only the learning rule and leaves loss, margin and norm within a few percent of the reference. Intermediate equalization strengths lie between the endpoints: at $\beta = 0.5$, circuit participation is 0.98 and test accuracy 0.985; at $\beta = 0.75$, they are 0.99 and 0.977.

Table 8: Optimizer and regularizer controls ($p = 31$, three seeds). Upper block: circuit participation and random-deletion tolerance under other optimizers and under dropout; margin and norm are given where measured. Lower block: participation ratio of the measured learning signal in the Adam runs, from the raw gradient and from Adam's effective step $m/\sqrt{v}$.

Setting	Circuit PR	Random del	Margin	Norm
Circuit participation				
SGD with momentum, β=0	0.53	0.60	—	—
SGD with momentum, β=1	0.99	0.76	—	—
Lion, β=1	0.984 (0.003)	—	—	—
Dropout 0.1	0.95	0.79	5.3	26.8
Dropout 0.5	0.99	0.82	6.8	30.2
Credit equalization, β=1 (Adam)	0.99	0.77	5.2	26.6
Participation of the measured learning signal (Adam runs)				
Raw gradient, β=0	0.59	—	—	—
Effective step $m/\sqrt{v}$, β=0	0.66	—	—	—
Effective step $m/\sqrt{v}$, β=1	0.99	—	—	—

Table 9: Learning-rate and weight-decay grid for the reference ($p = 31$, three seeds; standard deviations in parentheses). All settings grok and retain concentrated circuits, with credit participation close to circuit participation.

Learning rate	Weight decay	Test	Credit PR	Circuit PR
3×10^{-4}	10^{-3}	0.998	0.613	0.626 (0.012)
3×10^{-4}	3×10^{-3}	0.983	0.560	0.571 (0.027)
10^{-3}	10^{-3}	0.997	0.604	0.613 (0.016)
10^{-3}	3×10^{-3}	0.993	0.523	0.530 (0.008)
3×10^{-3}	10^{-3}	0.993	0.589	0.594 (0.024)
3×10^{-3}	3×10^{-3}	0.988	0.546	0.553 (0.022)

C.4 Longer training and switching the learning rule

Is the concentrated reference circuit a stable solution, or a stage on the way to a distributed one? We extend training from 20,000 to 200,000 epochs. Reference participation stabilizes by approximately 40,000 epochs and ends at $0.605\,(0.012)$; full equalization ends at $0.990\,(0.003)$ (Table 10). Longer training does not distribute the reference circuit.

We then train for 20,000 epochs with one rule and continue for 30,000 epochs with the other, with same-rule continuations as controls. Switching from the reference to full equalization raises participation from $0.613\,(0.016)$ to $0.991\,(0.000)$ within 5000 epochs, at test accuracy 0.997; continuing with the reference leaves it at $0.610\,(0.012)$. The reverse switch lowers participation from $0.990\,(0.003)$ to $0.722\,(0.016)$ and is still decreasing when the run ends, while its control stays at $0.990\,(0.003)$. Task and data are unchanged in all four runs; only the credit assignment changes. A fixed rule thus preserves its circuit over the tested horizon, and changing the credit allocation reorganizes the outgoing weights of an already learned circuit in either direction. Figure 2b shows a separate run of the same protocol at 100-epoch resolution: after the switch to equalization, participation exceeds 0.95 within 300 epochs, and after the reverse switch it reaches 0.67 after 30,000 epochs and is still decreasing.

Table 10: Circuit stability under a fixed rule and reorganization after switching ($p = 31$, three seeds; standard deviations in parentheses). Fixed-rule runs show participation after 20,000 and 200,000 epochs. Switched runs use 20,000 epochs in the first phase and 30,000 in the second and show participation after each phase; the reverse switch has not stabilized by the end. Same-rule continuations control for the additional training.

Condition	Circuit PR before	Circuit PR after	Test
Rule held, 20,000 *to* 200,000 *epochs*			
Reference (β=0)	0.60	$0.605\,(0.012)$	0.997
Credit equalization (β=1)	0.99	$0.990\,(0.003)$	0.975
Rule switched after phase one			
β=0 $\rightarrow$ β=1	$0.613\,(0.016)$	$0.991\,(0.000)$	0.997
β=1 $\rightarrow$ β=0	$0.990\,(0.003)$	$0.722\,(0.016)$	0.990
β=0 $\rightarrow$ β=0 (control)	$0.613\,(0.016)$	$0.610\,(0.012)$	0.997
β=1 $\rightarrow$ β=1 (control)	$0.990\,(0.003)$	$0.990\,(0.003)$	0.975

D Normalization groups and optimizer comparison

D.1 Normalizing groups of neurons

Section 4.2 shows that the number of neurons sharing one normalization factor decides how far the circuit is redistributed. This section gives the details of that sweep. We divide the 256 neurons of the first hidden layer into g equal groups of $m = n/g$ neurons, fixed at the start of the run, and at every step multiply all gradients in a group by the common factor of Equation 3, so that the mean credit is equal across groups while the relative magnitudes within a group are unchanged; a group whose gradients are all zero is left unchanged. Group size $m = 256$ corresponds to no rescaling and reproduces the reference: $0.602\,(0.023)$ against $0.60\,(0.02)$ in the β sweep. Group size $m = 1$ rescales each neuron separately and reproduces full credit equalization: 0.992 against 0.99 at $\beta = 1$. Across the per-neuron settings, credit and circuit participation differ by 0.0074 on average within seeds, which is the vertical distance from the identity line in Figure 3a.

Most of the change happens at the smallest groups. Going from pairs to single neurons raises circuit participation from 0.753 to 0.992, about 61% of the total increase from the whole-layer value, and norm-ranked deletion tolerance from 0.327 to 0.576. Groups of four and two give intermediate values. Per-neuron rescaling of the credit is thus the setting most likely to distribute a circuit, and coarser groups keep most of the concentration.

On S_5 the sweep gives 0.749 for groups of four, 0.852 for pairs and 0.974 for single neurons, the same pattern as on modular addition.

D.2 Comparison across optimizers

The controlled sweep varies only the group size and preserves the direction of every neuron's gradient before Adam. Optimizers combine a group structure with other transformations, so we ask whether they nevertheless sort by their normalization group. Table 11 reports test accuracy, credit participation and circuit participation for the eight optimizers of Table 1 with the settings of Appendix A. LARS (one factor per layer), RMSProp and Lion (per parameter), LAMB (per parameter with a layerwise trust ratio), Adafactor (factored per-parameter scaling), and Muon and Shampoo (matrix transformations) all produce concentrated circuits, with participation between 0.470 and 0.613, although their credit participation ranges from 0.478 to 1.000. Nero normalizes per neuron and produces a distributed circuit. At its default learning rate 10^{-2} and at 3×10^{-3}, it generalizes with test accuracy 0.985 and 0.978 and participation $0.986\,(0.002)$ and $0.983\,(0.001)$ (lower block of Table 11). It also tolerates neuron deletion better than the reference: random-deletion tolerance is 0.748 and norm-ranked tolerance 0.626 (0.728 and 0.541 at the smaller learning rate), against 0.672 and 0.311 for the reference in the same experiment, close to the values of the equalized network. A per-neuron optimizer thus provides the same robustness as our intervention. Because Nero holds each neuron's incoming-weight norm fixed, its outgoing-norm participation cannot arise from scale moving between a neuron's incoming and outgoing weights, the rescaling symmetry discussed in Appendix B.1.

Proposition 4 covers positive rescalings that preserve update directions, whereas per-parameter and matrix transformations rotate the update, so the optimizer comparison extends the controlled sweep empirically. NorMuon (Li et al., 2025) adds a per-neuron normalization after Muon's orthogonalization, motivated by the unequal neuron update norms that Muon produces, the same inequality that concentrates the circuit here.

Table 11: Optimizer comparison on modular addition ($p = 31$, three seeds, 20,000 epochs). The upper block runs every optimizer at the learning rate 10^{-3} and weight decay 10^{-3} of the reference (LARS at fifty times the learning rate); the reference and full credit equalization are included as comparison points. The lower block reports Nero, which uses no weight decay, at its default learning rate 10^{-2} and at 3×10^{-3}. Credit participation is measured on the raw gradient for the reference and LARS, on the rescaled gradient for credit equalization, and on the applied update for the other optimizers (Appendix A). Optimizers: LARS (You et al., 2017), RMSProp (Tieleman & Hinton, 2012), LAMB (You et al., 2020), Adafactor (Shazeer & Stern, 2018), Lion (Chen et al., 2023), Muon (Jordan et al., 2024), Shampoo (Gupta et al., 2018), Nero (Liu et al., 2021).

Optimizer	Normalizes	Test	Credit PR	Circuit PR
Reference (backpropagation, Adam)	per parameter	0.998	0.586	0.595
LARS	per layer	0.967	0.520	0.613
RMSProp	per parameter	0.992	0.791	0.470
LAMB	per parameter and per layer	0.995	0.779	0.543
Adafactor	factored, per parameter	0.991	0.839	0.518
Lion	per parameter (sign)	0.985	1.000	0.535
Muon	per matrix	0.999	0.478	0.524
Shampoo	per matrix	0.986	0.521	0.555
Credit equalization (β=1)	per neuron	0.976	1.000	0.992
Nero, no weight decay				
Nero, learning rate 10^{-2}	per neuron	0.985	—	0.986 (0.002)
Nero, learning rate 3×10^{-3}	per neuron	0.978	—	0.983 (0.001)

E Neuron deletion and adaptation

E.1 Neuron deletion

Deleting a first-hidden-layer neuron sets its outgoing weights to zero; the network is then evaluated on the test set without retraining. Random deletion removes fractions $0, 0.1, \ldots, 0.8$ of the layer, with 20 random masks per fraction (30 in the ten-seed experiment of Figure 1c). Deletion toler-

ance is the area under the curve of test accuracy against the deleted fraction at these nine fractions, computed by the trapezoidal rule and divided by the range 0.8, so that it lies between 0 and 1. Norm-ranked deletion removes neurons in decreasing order of outgoing norm at the same fractions. Greedy deletion evaluates every remaining neuron, removes the one whose removal lowers test accuracy the most, and repeats until accuracy falls below 0.5; we report the number of neurons removed. Greedy deletion selects neurons by their effect on the test set rather than by their weights.

Table 12 compares the reference, full credit equalization and DFC under random and norm-ranked deletion in one matched three-seed experiment (Figure 6). Equalization improves tolerance under both procedures, and the improvement is larger under norm-ranked deletion, which removes exactly the neurons a concentrated circuit relies on. DFC lies between the two. Table 13 reports greedy deletion for the reference and three equalization strengths: the number of neurons that must be removed grows with participation, from 23 at $\beta = 0$ to 61 at $\beta = 1$. The deletion advantage of the distributed circuit thus does not depend on selecting neurons by their norm. Over ten seeds, random-deletion tolerance is $0.672\ (0.012)$ for the reference and $0.773\ (0.014)$ under full equalization (Figure 1c).

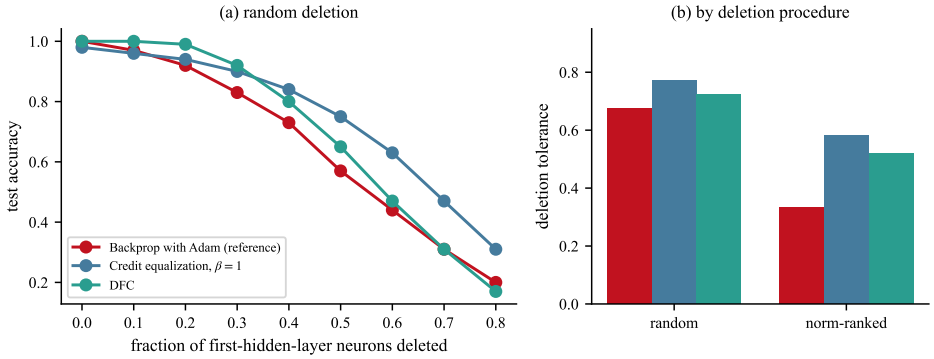


Figure 6: Neuron deletion for the reference, full credit equalization ($\beta = 1$) and DFC on modular addition ($p = 31$, three seeds). **(a)** Test accuracy against the fraction of randomly deleted first-hidden-layer neurons. **(b)** Deletion tolerance, the normalized area under the accuracy curve over the deletion fractions, under random and norm-ranked deletion.

Table 12: Deletion tolerance under random and norm-ranked deletion (modular addition, $p = 31$, three seeds). Norm-ranked deletion removes neurons in decreasing order of outgoing norm.

Deletion procedure	Reference	Credit equalization (β=1)	DFC
Random	0.674	**0.770**	0.725
Norm-ranked	0.332	**0.581**	0.518

Table 13: Greedy neuron deletion on modular addition ($p = 31$, three seeds). At each step the procedure removes the neuron whose removal lowers test accuracy the most. The last column gives the mean number of neurons removed before test accuracy falls below 0.5. Standard deviations of circuit participation are in parentheses.

Condition	Test	Circuit PR	Neurons removed
Reference	0.998	0.595 (0.023)	23.3
Credit equalization, β=0.25	0.997	0.940 (0.017)	43.0
Credit equalization, β=0.5	0.986	0.975 (0.002)	59.0
Credit equalization, β=1	0.976	0.992 (0.002)	60.7

E.2 Matching participation through outgoing-weight constraints

Does the deletion advantage depend on how the distributed circuit was trained, or only on the resulting distribution of outgoing norms? We compare two ways of reaching the same circuit participation. Credit equalization changes the gradients. An *outgoing-norm constraint* leaves the gradients unchanged and instead rescales the outgoing weights of the first hidden layer after every update to a prescribed norm profile with participation 0.60, 0.80 or 0.99; the profile has two norm levels, each shared by half of the neurons. All conditions use five seeds (Table 14).

Deletion. At participation 0.99, credit equalization and the outgoing-norm constraint give the same random-deletion tolerance, 0.777; at participation 0.60, the constrained network and the reference differ by 0.02. Greedy deletion removes $66.7\,(11.1)$ neurons under equalization and $68.7\,(10.8)$ under the constraint before accuracy falls below 0.5. Deletion tolerance thus follows the distribution of outgoing norms in both ways of producing it.

Adaptation. We also fine-tune these networks from addition to subtraction and measure the epochs to test accuracy 0.9, evaluated every 20 epochs. Within the constrained networks, adaptation takes 740, 1228 and 1480 epochs at participation 0.60, 0.80 and 0.99; the equalized network takes 1608 epochs. Adaptation slows with participation under both interventions. At participation 0.60, the reference takes 464 epochs and the constrained network 740: the reference has 87 neurons with an outgoing norm below one tenth of the layer mean, which the new task can recruit, whereas the constraint holds every neuron at one of two norms and leaves none. The difference points to the low-norm neurons, which a related task can recruit.

Table 14: Credit equalization compared with an outgoing-norm constraint of matched participation (modular addition, $p = 31$, five seeds; standard deviations in parentheses). Random del is the random-deletion tolerance; Adapt is the number of fine-tuning epochs to test accuracy 0.9 on subtraction.

Condition	Test	Circuit PR	Random del	Adapt
Reference	$0.995\,(0.007)$	$0.597\,(0.020)$	0.669	464
Credit equalization (β=1)	$0.976\,(0.015)$	$0.993\,(0.002)$	0.777	1608
Outgoing-norm constraint, 0.60	$0.987\,(0.016)$	$0.600\,(0.000)$	0.647	740
Outgoing-norm constraint, 0.80	$0.977\,(0.008)$	$0.800\,(0.000)$	0.675	1228
Outgoing-norm constraint, 0.99	$0.976\,(0.012)$	$0.990\,(0.000)$	0.777	1480

E.3 Adaptation to a new operation

Networks first grok addition at $p = 31$ at each β. We then fine-tune the whole network on the new operation with the reference rule, backpropagation with Adam at learning rate 10^{-3} and a fresh optimizer state, in every condition, on the new operation's own split with 40% of the pairs held out, and record the number of epochs until test accuracy on the new operation reaches 0.9, evaluated every 100 epochs (Table 16).

Transfer to subtraction. Adaptation time increases with the participation of the source circuit at every step, from 640 epochs at $\beta = 0$ to 1540 at $\beta = 1$; training from scratch takes 1410 epochs. The seed ranges at the two endpoints do not overlap: 400 to 800 epochs at $\beta = 0$ against 1200 to 1800 at $\beta = 1$. An independent ten-seed run, evaluated every 20 epochs, gives $690\,(164)$ and $1550\,(191)$ epochs. Pretraining on addition helps from the concentrated circuit and not from the distributed one.

The role of low-norm neurons. We count the neurons whose outgoing norm is below one tenth of the layer mean. The reference has about 85 such neurons. Our hypothesis is that subtraction can recruit them with little disturbance to the addition circuit. Already at $\beta = 0.25$ the count is zero, and adaptation time rises from 640 to 940 epochs. It keeps rising to 1540 epochs at $\beta = 1$ as the circuit spreads further, which is consistent with the new operation having to modify a larger share

of a distributed representation. In continual learning, inactive units signal a loss of plasticity that resetting them restores (Sokar et al., 2023; Dohare et al., 2024); here the low-norm neurons of the reference circuit coincide with faster adaptation, which suggests that unused capacity can serve a related task when the rest of the circuit stays intact.

Transfer after switching the learning rule. We train reference networks on addition for $20{,}000$ epochs, then copy each into two branches that continue for $30{,}000$ epochs with fresh Adam states, using either the reference rule or full equalization. Both branches are then fine-tuned identically on subtraction, with evaluation every 20 epochs. Across five seeds, equalized branches reach 0.9 test accuracy in $1052\,(273)$ epochs versus $608\,(160)$ for reference continuations; the mean paired difference is $444\,(131)$ epochs. Source accuracies are 0.987 and 0.996, respectively. Equalized versus reference continuations have outgoing-weight participation of 0.99 versus 0.60 and ablation participation of 0.72 versus 0.40. The participation measures show little further change between $10{,}000$ and $30{,}000$ continuation epochs. Parentheses give standard deviations.

Fine-tuning with the source rule. The transfer runs above fine-tune with the reference rule. To check that the ordering does not depend on the fine-tuning rule, we fine-tune each source network with both rules, evaluated every 20 epochs (Table 15). Over five seeds, the concentrated network reaches 0.9 on subtraction after $580\,(151)$ epochs with the reference rule and $1000\,(196)$ with full equalization; the distributed network takes $1496\,(195)$ and $1856\,(95)$. The distributed source is slower under both fine-tuning rules and in every seed, by a factor of 2.6 and 1.9. The slower adaptation of the distributed circuit does not depend on the fine-tuning rule.

Table 15: Fine-tuning from addition to subtraction with the reference rule or with full credit equalization, for a concentrated and a distributed source network (modular addition, $p = 31$, five seeds; standard deviations in parentheses). Adaptation time is the number of epochs to test accuracy 0.9, evaluated every 20 epochs.

Source network	Fine-tuning rule	Source test	Epochs to 0.9	Circuit PR after
Concentrated (β=0)	reference	0.992	$580\,(151)$	0.62
Concentrated (β=0)	equalization	0.992	$1000\,(196)$	0.99
Distributed (β=1)	reference	0.969	$1496\,(195)$	0.60
Distributed (β=1)	equalization	0.969	$1856\,(95)$	0.99

Transfer to multiplication. The same pretrained networks take 1800 to 2020 epochs to reach the threshold on multiplication, against 1200 from scratch. Multiplication does not share the operand frequencies of addition, and pretraining slows adaptation at every β. In our experiments the advantage of the concentrated circuit appears only for the target task that shares the operand frequencies of the source task.

Fine-tuning learning rate. At one third and at three times the reference learning rate, adaptation time still increases with β, and the fully equalized start takes 1.9 to 2.5 times as long as the concentrated start (Table 17). The reverse transfer, from subtraction to addition, shows the same ordering: 740 epochs at $\beta = 0$ and 1600 at $\beta = 1$.

F Other tasks and architectures

The main experiments use modular addition and a multilayer perceptron. Here we repeat the comparison between the reference and full credit equalization on other operations, tasks and architectures (Table 18). The intervention raises participation wherever the reference circuit is concentrated, and it changes little where the reference circuit is already distributed.

F.1 Subtraction and multiplication

Table 19 repeats the comparison on subtraction and multiplication at $p = 31$, with DFC for comparison. Equalization raises participation to 0.99 on both operations while test accuracy stays at

Table 16: Fine-tuning from addition to a new operation, with standard deviations in parentheses. Subtraction results for $\beta = 0, 0.25, 1$ and from scratch use ten seeds; the intermediate settings and all multiplication results use five. Adaptation time is the number of epochs to test accuracy 0.9. Low-norm neurons have an outgoing norm below one tenth of the layer mean.

Condition	Circuit PR	Low-norm neurons	Source test	Epochs to 0.9 on	
				subtraction	multiplication
$\beta = 0$ (reference)	0.593 (0.034)	85	0.995	640 (128)	1800 (141)
$\beta = 0.25$	0.933 (0.020)	0	0.989	940 (191)	1940 (233)
$\beta = 0.5$	0.972 (0.006)	0	0.990	1200 (89)	1980 (248)
$\beta = 0.75$	0.983 (0.003)	0	0.981	1300 (141)	2020 (133)
$\beta = 1$	0.991 (0.002)	0	0.972	1540 (180)	1940 (136)
From scratch	—	—	—	1410 (137)	1200 (63)

Table 17: Adaptation time from addition to subtraction across an order of magnitude in the fine-tuning learning rate, in epochs to test accuracy 0.9 (standard deviations in parentheses). The outer columns use three seeds; the middle column repeats Table 16 (ten seeds for the endpoints, five for $\beta = 0.5$).

Start	3×10^{-4}	10^{-3}	3×10^{-3}
$\beta = 0$ (reference)	1933 (525)	640 (128)	267 (47)
$\beta = 0.5$	3067 (205)	1200 (89)	533 (47)
$\beta = 1$	3733 (262)	1540 (180)	667 (94)
ratio β=1 to β=0	1.9	2.4	2.5

Table 18: Circuit participation of the reference (β=0) and under full credit equalization (β=1) across the tested settings. Test accuracies and seed counts are given in the cited subsections. Sparse parity and MNIST show no grokking delay. The last two settings start from a distributed circuit and are unchanged.

Setting	Circuit measured	β=0	β=1	Details
Addition, subtraction, multiplication	first hidden layer	0.43–0.60	0.99	App. F.1
Sparse parity	first hidden layer	0.15	0.96	App. F.3
S_5 composition	first hidden layer	0.70	0.97	App. F.4
Transformer, $p = 31$, wide MLP	MLP block	0.61	0.99	App. F.5
Transformer, $p = 113$, canonical	MLP block	0.96	0.96	App. F.5
MNIST	first hidden layer	0.99	0.99	App. F.6

or above 0.994, and random-deletion tolerance increases as on addition. The largest change is on multiplication, where the reference circuit is the most concentrated of the three, with participation 0.43.

F.2 Network width

Does the difference between the two circuits depend on how many neurons are available? We vary the hidden width from 32 to 2048 with the task, the budget and the comparison between $\beta = 0$ and $\beta = 1$ fixed (Table 20). At width 32 the effective counts are 30 for the reference and 31 under equalization. At width 2048 they are 332 and 2046. The reference uses a decreasing fraction of the layer as the width grows, from participation 0.937 to 0.162, while equalization stays near one. Test accuracy is at least 0.96 throughout. Normalized participation and effective count answer different questions and can move in opposite directions: from width 128 to 512 the reference's participation falls from 0.791 to 0.400 while its effective count grows from 101 to 205. A wider reference network uses more neurons in absolute terms and a smaller share of the layer.

Table 19: Circuit participation and random-deletion tolerance on the three modular operations ($p = 31$, three seeds; standard deviations in parentheses). All three operations use the same random split with 40% of the pairs held out.

Operation	Rule	Test	Circuit PR	Random del
Addition	reference	0.998 (0.001)	0.60 (0.02)	0.671 (0.016)
	equalization (β=1)	0.976 (0.011)	0.99 (0.00)	0.771 (0.019)
	DFC	1.000 (0.000)	0.92 (0.00)	0.717 (0.014)
Subtraction	reference	0.999 (0.001)	0.59 (0.04)	0.680 (0.026)
	equalization (β=1)	0.994 (0.005)	0.99 (0.00)	0.785 (0.014)
	DFC	1.000 (0.000)	0.93 (0.01)	0.735 (0.010)
Multiplication	reference	0.999 (0.001)	0.43 (0.07)	0.666 (0.035)
	equalization (β=1)	0.999 (0.001)	0.99 (0.00)	0.867 (0.012)
	DFC	1.000 (0.000)	0.86 (0.02)	0.764 (0.010)

Table 20: Width dependence of first-hidden-layer circuit participation and effective neuron count $n \cdot \mathrm{PR}$ at $\beta = 0$ and $\beta = 1$ (modular addition, $p = 31$, three seeds, 15,000 epochs).

	Circuit PR		$n \cdot \mathrm{PR}$	
Width	$\beta = 0$	$\beta = 1$	$\beta = 0$	$\beta = 1$
32	0.937	0.980	30	31
64	0.915	0.987	59	63
128	0.791	0.991	101	127
512	0.400	0.994	205	509
1024	0.262	0.994	268	1018
2048	0.162	0.999	332	2046

F.3 Sparse parity

We test credit equalization on sparse parity, a task used to study how networks find a few relevant features after a long plateau (Barak et al., 2022). The input is a vector of 20 bits, each 0 or 1; the label is 1 if an odd number of three fixed bits are set and 0 otherwise, so the remaining 17 bits are distractors that the network has to ignore. We draw 4000 or 600 inputs at random from the 2^{20} possibilities and assign each to the training or the test set with probability one half, which gives about 2000 and 300 training examples. The runs generalize without delay under full-batch training, so this experiment tests the redistribution outside the grokking regime. Equalization raises circuit participation from 0.15 to 0.96 on the larger dataset and from 0.08 to 0.89 on the smaller one. Test accuracy is 1.000 for both conditions on the larger dataset and changes from 1.000 to 0.988 on the smaller one, and random-deletion tolerance improves at both sizes (Table 21). The redistribution is thus not restricted to modular arithmetic or to delayed generalization.

Table 21: Sparse parity with approximately 2000 and 300 training examples (three seeds; standard deviations in parentheses).

Train size	Rule	Test	Circuit PR	Random del
~ 2000	reference	1.000 (0.000)	0.15 (0.06)	0.865 (0.051)
	equalization (β=1)	1.000 (0.000)	0.96 (0.03)	0.955 (0.002)
~ 300	reference	1.000 (0.000)	0.08 (0.02)	0.825 (0.017)
	equalization (β=1)	0.988 (0.017)	0.89 (0.02)	0.908 (0.060)

F.4 PERMUTATION COMPOSITION

We test whether credit equalization redistributes circuits on a noncommutative operation, the composition of permutations, which Chughtai et al. (2023) use to study the circuits of grokked networks. The group S_5 contains the 120 permutations of five objects. The input is a pair (g, h) of permutations, and the target is their composition $g \circ h$, the permutation obtained by applying h first and then g. We use one-hot inputs and train for 30,000 epochs on half of the 120×120 input pairs, with the rest held out. Both the reference and the equalized network grok under the same definition as for modular arithmetic; the delay grows when fewer training pairs are available. Full equalization raises circuit participation from $0.699\,(0.078)$ to $0.973\,(0.005)$ while both conditions reach test accuracy 1.000. Random-deletion tolerance increases from 0.485 to 0.633 and norm-ranked tolerance from 0.149 to 0.468 (Table 22). Reducing the training fraction to 40% reproduces the pattern. Credit and circuit participation again agree closely.

Table 22: Composition in S_5. The upper block uses ten seeds with half the pairs held out, the lower block five seeds with 60% held out; parentheses show standard deviations. Delay is the number of epochs between reaching training accuracy 1.0 and test accuracy 0.95.

Rule	Test	Credit PR	Circuit PR	Random del	Norm-ranked del	Delay
Half the pairs held out, ten seeds						
Reference	1.000	0.689	0.699 (0.078)	0.485	0.149	500
Equalization, β=0.5	1.000	0.970	0.942 (0.010)	0.585	0.322	550
Equalization, β=1	1.000	1.000	0.973 (0.005)	0.633	0.468	900
60% *held out, five seeds*						
Reference	1.000	0.657	0.662 (0.038)	0.409	0.129	1350
Equalization, β=0.5	1.000	0.951	0.907 (0.016)	0.525	0.260	1650
Equalization, β=1	1.000	1.000	0.970 (0.002)	0.597	0.410	1750

F.5 TRANSFORMERS

We test whether credit equalization changes the participation of a transformer's MLP block. We follow the modular-addition transformer of Nanda et al. (2023): inputs have the form "$a\ b\ =$", and the canonical configuration has one block, $p = 113$, $d_{\text{model}} = 128$, one attention head per 32 channels, a ReLU MLP of width $4d_{\text{model}}$ and weight decay 1.0, trained with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.98$) at learning rate 10^{-3} on 30% of the operand pairs for $40{,}000$ epochs. Credit equalization is applied to every linear map.

In the canonical model the reference MLP is already distributed: its participation is 0.955, and full equalization leaves it at 0.955 with test accuracy 1.000 in both conditions (Table 23). A second modulus and a doubled model width give the same picture, with MLP participation changing by at most 0.015. Applying the intervention only to the MLP changes participation by less than the variation across seeds. The intervention thus does not increase MLP participation where the reference is already distributed.

A wider MLP gives a concentrated reference. With 4096 MLP neurons at $p = 31$, reference participation is $0.610\,(0.048)$, and full equalization raises it to $0.993\,(0.003)$ over five seeds, with the seed ranges not overlapping. Norm-ranked deletion tolerance rises from 0.580 to 0.941, test accuracy stays at 1.000 and the margin is unchanged to two decimal places.

Credit equalization thus has a large effect in the transformer whose reference MLP is concentrated and a small one in the configurations whose reference is already distributed.

F.6 MNIST

We use the setting in which image-classification grokking was reported (Liu et al., 2023): 1000 training images and a network with two hidden layers of width 200, at the standard and at a larger initialization scale. Training and test accuracy rise together, so these runs show no grokking delay. The reference already has circuit participation $0.992\,(0.002)$, and full credit equalization changes it

Table 23: Modular-addition transformer results: test accuracy and MLP circuit participation (standard deviations in parentheses). Credit equalization is applied to every linear map; β=0 is the reference. The wide-MLP result is described in the text.

β	Test	MLP circuit PR
Canonical: one block, $p = 113$*,* $d_{model} = 128$*, six seeds*		
0.0	1.000	0.955 (0.01)
0.5	1.000	0.953 (0.01)
1.0	1.000	0.955 (0.01)
Second modulus: $p = 97$*, three seeds,* 70,000 *epochs*		
0.0	1.000	0.959 (0.01)
1.0	1.000	0.951 (0.02)
Twice the width: $d_{model} = 256$*, three seeds*		
0.0	1.000	0.902 (0.04)
1.0	1.000	0.917 (0.01)

by less than the variation across seeds. As in the canonical transformer, the reference leaves little room for a further increase in participation.

G THEORY OF CREDIT EQUALIZATION

We use a diagonal linear network to examine how equalizing gradient magnitudes changes the distribution of contributions across units. The model allows an exact comparison between unmodified gradient flow, per-unit equalization and groupwise equalization, and it explains a mechanism for the redistribution; the ReLU-network experiments test the effect itself. We first record two elementary properties of the participation ratio and of the intervention.

Proposition 1 (Participation ratio bounds)**.** *For $a \in \mathbb{R}^n_{\geq 0}$, $a \neq 0$, the participation ratio of Equation 1 satisfies $1/n \leq \mathrm{PR}(a) \leq 1$. The upper bound is attained exactly when all a_j are equal, and the lower bound exactly when a single a_j is nonzero. Consequently $n \cdot \mathrm{PR}(a)$ interpolates between 1 and n and acts as an effective count of contributing units.*

Proof. By Cauchy–Schwarz, $(\sum_j a_j)^2 \leq n \sum_j a_j^2$ with equality exactly for constant a, which gives $\mathrm{PR} \leq 1$. Since the a_j are nonnegative, $(\sum_j a_j)^2 \geq \sum_j a_j^2$ with equality exactly when at most one entry is nonzero, which gives $\mathrm{PR} \geq 1/n$. □

Proposition 2 (Credit rescaling preserves a descent direction)**.** *Let $g \neq 0$ be the loss gradient of the weight matrix to which Equation 2 is applied, with one row g_j per neuron of the layer it feeds, and let $\tilde{g}$ be the rescaled signal for any $\beta \in [0, 1]$. Then $\langle \tilde{g}, g \rangle > 0$, so a sufficiently small step along $-\tilde{g}$ decreases the loss when the other parameters are held fixed.*

Proof. Each nonzero row is rescaled by the positive scalar $s_j = \big((1-\beta)\|g_j\| + \beta \bar{g}\big)/\|g_j\|$, and zero rows remain zero, so $\langle \tilde{g}, g \rangle = \sum_{j:\, \|g_j\|>0} s_j \|g_j\|^2 > 0$ whenever $g \neq 0$. The directional derivative of the loss along $-\tilde{g}$ is therefore negative. □

This statement concerns the rescaled loss gradient before Adam and weight decay.

Proposition 3 (Unmodified and equalized flow select different minima)**.** *Consider a diagonal linear network with n units. Unit i has incoming weight u_i, outgoing weight v_i and contribution $a_i = u_i v_i$. Initialize $u_i = v_i = \sqrt{\epsilon_i} > 0$ with $\sum_i \epsilon_i < 1$, and train by gradient flow on the loss*

$$L \;=\; \tfrac{1}{2}\Big(\sum\nolimits_i a_i - 1\Big)^2, \tag{8}$$

whose minima form the set $\{a : \sum_i a_i = 1\}$. In this model, equalization rescales the two-coordinate gradient $(\partial_{u_i} L, \partial_{v_i} L)$ of each unit by one common factor, so both weights of a unit are updated symmetrically and stay balanced, $u_i = v_i$ and $a_i = u_i^2$. Participation is computed from the contributions a_i. Unless the ϵ_i are all equal, unmodified and equalized gradient flow converge to different minima:

(i) Unmodified gradient flow keeps every ratio a_i/a_j at its initial value, so it converges to the rescaled initialization $a_i = \epsilon_i / \sum_j \epsilon_j$, whose participation ratio equals that of the initialization.

(ii) The symmetric per-unit equalized flow keeps every difference $u_i - u_j$ at its initial value, so it converges to $u_i = \sqrt{\epsilon_i} + C$ with C set by $\sum_i (\sqrt{\epsilon_i} + C)^2 = 1$.

(iii) $\mathrm{PR}\big((\sqrt{\epsilon_i} + C)^2\big)$ is nondecreasing in $C \geq 0$, strictly increasing unless the ϵ_i are equal. Whenever the initialization undershoots the constraint, $\sum_i \epsilon_i < 1$ and $C > 0$, the equalized solution has strictly higher participation than the initialization, and it reaches 1 as the initialization scale tends to zero. Overshooting initializations are outside this statement because some coordinates can cross zero under equalization.

Proof. Write $r = \sum_j a_j - 1$ for the residual, a single scalar shared by every unit. The two flows differ in one line,

$$\text{unmodified flow: } \dot{u}_i = -\partial_{u_i} L = -r\, u_i, \qquad \text{equalized flow: } \dot{u}_i = -r\, \bar{u}, \quad \bar{u} = \tfrac{1}{n} \textstyle\sum_j u_j, \tag{9}$$

where the equalized form holds because unit i's two-coordinate gradient (ru_i, ru_i) has norm $|r|\sqrt{2}\, u_i$, and equalization rescales it to the mean norm $|r|\sqrt{2}\, \bar{u}$. For (i), $\dot{a}_i = 2u_i \dot{u}_i = -2ra_i$, so $\frac{d}{dt} \log(a_i/a_j) = 0$ because r is common to all units. The invariance is a property of this model, since in a network whose units see different residuals the ratios do move. For (ii), every u_i takes the same step, so $\sqrt{a_i(t)} = \sqrt{\epsilon_i} + C(t)$ with $C(t) = -\int_0^t r\bar{u}\, ds$, and fitting the target fixes C. For (iii), write $b_i = \sqrt{\epsilon_i} + C$ and $S_k = \sum_i b_i^k$, so that $\mathrm{PR} = S_2^2/(nS_4)$ and

$$\frac{d}{dC} \mathrm{PR} \;=\; \frac{4\, S_2 \left(S_1 S_4 - S_2 S_3\right)}{n\, S_4^2}, \qquad S_1 S_4 - S_2 S_3 \;=\; \tfrac{1}{2} \sum_{i,j} b_i b_j\, (b_i + b_j)(b_i - b_j)^2 \;\geq\; 0, \tag{10}$$

where the identity follows by symmetrizing the double sum $\sum_{i,j} b_i b_j^4 - b_i^2 b_j^3$ over i and j, and it is zero exactly when all nonzero b_i coincide. Participation is nondecreasing along the path from $C = 0$, where it equals $\mathrm{PR}(\epsilon)$, to the fitted $C > 0$, and strictly increasing unless the ϵ_i are equal. As the initialization scale tends to zero, $C \to 1/\sqrt{n}$ and $a_i \to 1/n$, whose participation ratio is 1. □

Under unmodified gradient flow every contribution grows by the same factor, so their ratios stay fixed. Under symmetric per-unit equalization every u_i receives the same increment; the initial differences then become small relative to the shared growth, and participation increases. Both flows optimize the same loss but select different members of its set of minima. The model differs from the ReLU experiment in two ways: it rescales both weights of a unit so that they stay balanced, whereas the experiment rescales only the incoming weights, and it measures the products $u_i v_i$ rather than the outgoing norms. It provides a mechanism for the observed redistribution, not an exact prediction for the trained nonlinear network. The same model describes groupwise equalization, which preserves the ratios within each group while equalizing the group totals.

Proposition 4 (Group equalization removes only between-group inequality)**.** *Partition the n units into g groups of $m = n/g$, fixed throughout training. In the model of Proposition 3, rescale each unit's two-coordinate gradient by the factor, common within its group, that equalizes the total credit across groups. Write*

$$p_i \;=\; \frac{\sqrt{\epsilon_i}}{\sum_{j \in \mathrm{group}(i)} \sqrt{\epsilon_j}} \tag{11}$$

for unit i's share of its own group's initialization, measured in the $\sqrt{a}$ coordinate in which the flow is linear, and let k run over groups. As the initialization scale tends to zero, the fitted solution has participation ratio

$$\mathrm{PR}(m) \;=\; \frac{S^2}{m\, Q}, \qquad S = \frac{1}{g} \sum_k \sum_{i \in k} p_i^2, \qquad Q = \frac{1}{g} \sum_k \sum_{i \in k} p_i^4, \tag{12}$$

which equals the unmodified-flow value of Proposition 3 at $g = 1$ and equals 1 at $m = 1$.

Proof. The rescaling factor is shared within a group, so inside group k every $\log u_i$ has the same time derivative. The within-group shares p_i freeze at their initial values, as the ratios freeze under unmodified flow. The group totals $U_k = \sum_{i \in k} u_i$ obey

$$\dot{U}_k \;=\; -\,r\,c_k U_k \;=\; -\,r\,\bar{U}, \qquad c_k = \bar{U}/U_k, \qquad \bar{U} = \tfrac{1}{g}\textstyle\sum_k U_k, \tag{13}$$

a common increment, so as the initialization tends to zero all group totals equalize to a common U. The solution is then $a_i = p_i^2 U^2$ with $U^2 = 1/(gS)$ forced by $\sum_i a_i = 1$, and substituting into Equation 1 gives Equation 12. At $m = 1$ every share is 1, so $S = Q = 1$ and $\mathrm{PR} = 1$, recovering Proposition 3. At $g = 1$ the rescaling is the identity, and Equation 12 reduces to the participation of the normalized initialization, the unmodified-flow value. □

Groupwise equalization changes the totals assigned to the groups while preserving the relative weights inside each group. To compare Equation 12 with the neural-network sweep, we assume lognormally distributed initial contributions with log standard deviation σ. For many units the unmodified model then has participation $e^{-\sigma^2}$; we set this equal to the measured whole-layer value 0.602, which fixes σ from that one point. The model then predicts participation 0.674, 0.737, 0.834 and 1.000 for groups of eight, four, two and one, against measured values of 0.623, 0.695, 0.753 and 0.992. It reproduces the shape of the curve: flat for large groups and steep at the smallest.

Proposition 5 (Participation and score fluctuations under random deletion)**.** *Model the score of the correct class as a sum of unit contributions, $y = \sum_j c_j$ with $c_j \geq 0$ over n units, and delete units independently with probability ρ, giving the surviving score $y_\rho = \sum_j b_j c_j$ with $b_j \sim \mathrm{Bernoulli}(1-\rho)$ marking survival. Then, assuming $\sum_j c_j > 0$ and $0 \leq \rho < 1$, for every $\varepsilon \in (0,1)$,*

$$\frac{\sqrt{\mathrm{Var}[y_\rho]}}{\mathbb{E}[y_\rho]} = \sqrt{\frac{\rho}{1-\rho}} \cdot \frac{1}{\sqrt{n \cdot \mathrm{PR}(c)}}, \qquad \mathbb{P}\big(y_\rho < (1-\varepsilon)\,\mathbb{E}[y_\rho]\big) \;\leq\; \frac{\rho}{(1-\rho)\,\varepsilon^2\, n \cdot \mathrm{PR}(c)}, \tag{14}$$

where $\mathrm{PR}(c)$ is Equation 1 evaluated on the vector of contributions.

Proof. $\mathbb{E}[y_\rho] = (1-\rho)\sum_j c_j$ and, by independence, $\mathrm{Var}[y_\rho] = \rho(1-\rho)\sum_j c_j^2$. Dividing and using Equation 1 gives the first identity, and Chebyshev's inequality gives the tail bound. □

In this additive model, a larger effective count reduces the relative fluctuation of the score under independent random deletion. The proposition gives the direction of the effect; the deletion experiments establish the accuracy differences in the trained networks.

H Supplementary analysis of generalization

Section 4.3 reports that a weight-space measurement, q, separates the runs that generalize from those that only fit the training set. This appendix gives the definition, the compared configurations, the full results and the limits of the measurement.

A local measure on the weight vector. A well-studied account of grokking explains generalization through weight decay: a generalizing circuit fits the training data with smaller weights than a memorizing one, so regularization gradually favors it (Nanda et al., 2023; Varma et al., 2023; Liu et al., 2023). Once the training data are fitted, the network can keep moving among parameter settings that preserve the fit, and weight decay pushes it toward the smallest of them, unless some other component of the learning signal opposes it. To measure this, let $J(\theta)$ be the Jacobian of all training logits with respect to the parameters. A small parameter change $\boldsymbol{v}$ changes the logits by $J\boldsymbol{v}$ to first order, so the directions with $J\boldsymbol{v} = 0$, the null space of J, are the directions in which weights can be reduced without changing the training outputs to first order. Let P be the orthogonal projector onto this null space. Then $P\theta$ is the component of the current weight vector along these directions, and we measure its norm,

$$q(\theta) \;=\; \|P\theta\|_2. \tag{15}$$

Both J and P depend on the current parameters, so q is a local measurement that we recompute at every checkpoint; it is not a count of unnecessary parameters or neurons.

For continuous dynamics $\dot{\theta} = -\nabla\ell - \lambda\theta - \boldsymbol{b}$, where ℓ is the training loss, $\lambda > 0$ the weight-decay strength and $\boldsymbol{b}$ the learning signal's departure from the loss gradient, the loss gradient has no component in the null space and

$$P\dot{\theta} = -\lambda P\theta - P\boldsymbol{b}.$$

Weight decay shrinks the projected weights while an additional signal can oppose it. If the Jacobian is held fixed and $P\boldsymbol{b} = 0$, the projected weights decay exponentially and q tends to zero; a constant nonzero $P\boldsymbol{b}$ instead leaves the stationary value $P\theta = -P\boldsymbol{b}/\lambda$ (Proposition 6). This calculation motivates measuring q during training. Since the projector changes and Adam transforms the signal, we measure q for every rule, including backpropagation.

Compared configurations. We measure nine configurations on modular addition at $p = 31$ with three seeds each and a $15{,}000$-epoch budget: the reference, full credit equalization, DFC, feedback alignment (FA) and direct feedback alignment (DFA) (Lillicrap et al., 2016; Nøkland, 2016), sign-SGD (Bernstein et al., 2018), and three constructed controls. Every configuration writes its learning signal into the gradient slot, and the signal is then clipped and passed to Adam at the shared learning rate 10^{-3} and weight decay 10^{-3}. Sign-SGD also memorizes when applied directly, at learning rates 10^{-4} and 10^{-5} and with weight decay up to 10^{-2} (three seeds, test accuracy 0.000). Each control replaces the update by the unit vector $\cos\varphi\,\hat{\boldsymbol{g}} + \sin\varphi\,\hat{\boldsymbol{v}}_\perp$ rescaled to the gradient norm, where $\hat{\boldsymbol{g}}$ is the normalized cross-entropy gradient, $\hat{\boldsymbol{v}}_\perp$ is a random vector projected orthogonally to the current gradient and normalized, and $\cos\varphi = 0.4$. *Fresh noise* draws the random vector anew at every step. *Fixed bias* draws it once per run, so the added component keeps its direction up to the projection. *Error-gated bias* uses the same fixed vector but multiplies the orthogonal component by the current loss relative to its initial value before the renormalization, so its cosine to the gradient is 0.4 at the start and rises toward one as the loss falls. Fresh noise and error-gated bias grok, fixed bias memorizes. The recorded checkpoints, at epochs 75, 150, 300, 450, 750 and 1500, show the delay in every grokking run: training accuracy is 1.0 by epoch 150 (by epoch 300 for fresh noise), and test accuracy first exceeds 0.95 between epochs 750 and 1500 for the reference and error-gated bias, after epoch 1500 for credit equalization and fresh noise, and between epochs 450 and 750 for DFC, whose delays at 10-epoch resolution are 520 to 610 epochs (Appendix A).

Results. At epoch 300, 2% of training, all 27 runs have training accuracy 1.0 and test accuracy 0.0. The estimated q nevertheless already separates the eventual outcomes: the largest value among the runs that later grok is 22.02, below the smallest value among the runs that memorize, 27.71 (Table 24). The separation is also present at the endpoints, where every grokking run has a smaller q than every memorizing run (Table 25). Total parameter norm does not separate the runs at epoch 300: the largest norm among the runs that later grok, 42.2, exceeds the smallest norm among the memorizing runs, 41.3. At the endpoints the norm separates the outcomes as well (grokking runs between 13.1 and 26.7, memorizing runs between 35.6 and 150), so the advantage of q over the norm is its early separation. These observations agree with the weight-decay account.

Table 24: Estimated null-space weight norm q early in training, as means (standard deviations) over three seeds. Every run fits the training set and has zero test accuracy at epoch 300. The outcome column refers to the completed runs.

Configuration	Epoch 300 (2%)	Epoch 750 (5%)	Outcome
Reference (backpropagation with Adam)	14.32 (0.09)	7.63 (0.14)	groks
Credit equalization (β=1)	14.46 (0.08)	9.11 (0.13)	groks
DFC	11.69 (0.14)	6.48 (0.08)	groks
Error-gated bias	21.93 (0.09)	9.53 (0.04)	groks
Fresh noise	15.37 (0.26)	9.32 (0.13)	groks
DFA	27.80 (0.10)	26.05 (0.36)	memorizes
FA	28.45 (0.21)	27.63 (0.31)	memorizes
Fixed bias	48.94 (0.08)	49.71 (0.05)	memorizes
Sign-SGD	49.13 (0.02)	72.33 (0.48)	memorizes

Numerical estimation. We estimate the projection as

$$\widehat{P\theta} = \theta - J^\top (JJ^\top + \epsilon I)^{-1} J\theta,$$

Table 25: Estimated null-space weight norm q at the end of training (15,000 epochs of modular addition at $p = 31$; means over three seeds). Every grokking configuration has a smaller q than every memorizing one. At initialization, $q = 29.9$.

Configuration	$q(\theta)$	Outcome	Configuration	$q(\theta)$	Outcome
Fresh noise	5.30	groks	DFA	26.34	memorizes
Reference	5.90	groks	FA	26.55	memorizes
Error-gated bias	6.55	groks	Fixed bias	51.80	memorizes
Credit equalization (β=1)	7.24	groks	Sign-SGD	134.72	memorizes
DFC	8.49	groks			

where the subtracted term approximates the component of θ in the span of the training-logit gradients and the damping $\epsilon > 0$ makes the remainder an approximation rather than an exact projection. The linear system is solved with conjugate gradients using Jacobian-vector products, without forming J. The reported measurements use relative damping 10^{-6} and 400 conjugate-gradient iterations. On small networks the estimate agrees with an explicit singular-value decomposition to a relative error below 10^{-4}. A longer solve at one trained width-256 reference checkpoint (10,000 preconditioned iterations) gives $q = 6.04$ with a relative residual of 10^{-3} and an idempotence error of 0.010 relative to the parameter norm. At epoch 300, retraining the nine configurations and re-estimating q with relative damping from 10^{-10} to 10^{-4}, up to 10,000 iterations and in double precision changes every value by at most 0.01, and the largest value among the runs that later grok stays below the smallest among those that memorize under every setting.

Proposition 6 (Projected weight dynamics with a fixed local Jacobian)**.** *Let ℓ be a differentiable loss of the training logits $z(\theta)$, let $J = \partial z/\partial\theta$, and let P project orthogonally onto* $\mathrm{null}(J)$*. Under $\dot\theta = -\nabla_\theta \ell - \lambda\theta - \boldsymbol{b}$ with $\lambda > 0$,*

$$P\dot\theta \;=\; -\lambda P\theta - P\boldsymbol{b}. \tag{16}$$

At a stationary point, $P\theta = -P\boldsymbol{b}/\lambda$. In the local model in which J is held fixed, write $\boldsymbol{v} = P\theta$. If $P\boldsymbol{b} = \boldsymbol{b}_0$ is constant, then

$$\boldsymbol{v}(t) \;=\; e^{-\lambda t}\boldsymbol{v}(0) - \frac{1 - e^{-\lambda t}}{\lambda}\,\boldsymbol{b}_0,$$

so $\|\boldsymbol{v}(t)\|_2$ tends to $\|\boldsymbol{b}_0\|_2/\lambda$. If $P\boldsymbol{b}(t)$ is locally integrable and tends to zero, then $\boldsymbol{v}(t)$ tends to zero.

Proof. The chain rule gives $\nabla_\theta \ell = J^\top \nabla_z \ell$, so $P\nabla_\theta \ell = 0$. Projecting the dynamics gives Equation 16 and the stationary-point condition. For fixed J, the integrating-factor solution is

$$\boldsymbol{v}(t) \;=\; e^{-\lambda t}\boldsymbol{v}(0) - \int_0^t e^{-\lambda(t-s)} P\boldsymbol{b}(s)\, ds.$$

Constant forcing gives the displayed expression. If the forcing tends to zero, splitting the integral into a fixed initial interval and the remaining tail shows that both contributions tend to zero. □

The proposition concerns a continuous-time model with a fixed Jacobian and the signal applied directly.