Controlled Demolition Is All You Need for Grokking

Anonymous authors
Paper under double-blind review

Abstract

There are a great many conjectures about how grokking is reached and what its properties are; many of them are supported by experiments and yet sit poorly with one another. I introduce three new methods for producing grokking — artificial ricochets (controlled demolition: the learning rate is escalated until the network breaks, and the network is then left to recover), an attention learning-rate overdrive, and an overdrive of the FFN input matrix together with row-wise weight normalization. All three run on the same training loop, the same architecture and the same initialization, and are compared there with the classical method, weight decay; this makes the paths to grokking commensurable and separates the intrinsic properties of the phenomenon from artifacts of the method that produces it. I show that the network's work of generalizing can remain invisible to the usual metrics and is nonetheless brought out by applying another of these methods.

1. Introduction

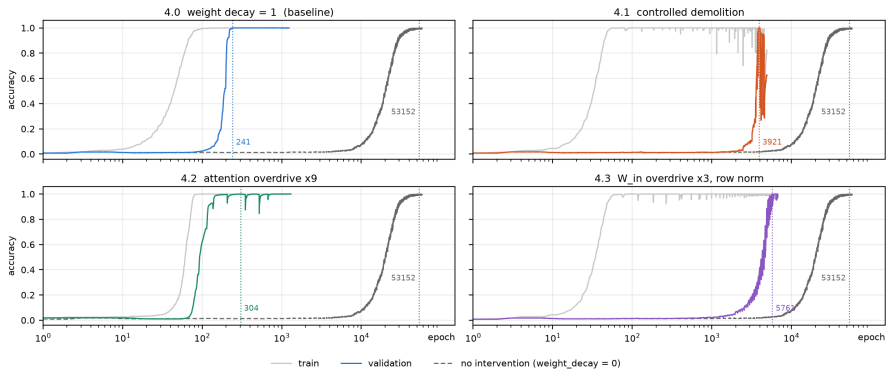


Figure 1. *Five paths to grokking on one testbed: one panel per method. Seed 53 is the median of the nine by the epoch of grokking; the curves are cleaned visually of downward outliers (at every point the maximum over the last 5 epochs is drawn). The fifth – the control, there for comparison in every panel – is simply weight_decay = 0.*

Grokking is delayed generalization (Power et al., 2022). It is most often demonstrated on small algorithmic tasks (§2, appendix K). Grokking has several persistent features that do not fit the intuition formed on underparametrized, low-dimensional problems:

- generalization arrives *after* overfitting has already set in, and the gap between the training and the validation results can be very large;
- successful generalization may require a substantially larger model, and a far greater degree of overfitting, than is usually held admissible in ML at all;
- the switch to a generalizing solution can be very fast — especially against the stretch of training on which, if one watches accuracy alone, nothing of importance appears to be happening (Nanda et al., 2023);
- generalization can reach 100 % on validation, which makes it look like an exact solution.

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.

The phenomenon has produced a large body of work on how to induce it, how to speed it up and how to explain it (for a systematic survey, §2). The aim of this work is not to accelerate grokking, nor to obtain it under some special conditions: the subject of the paper is an instrument that separates the *true* properties of the phenomenon from the particulars of any one method of reaching it. To that end I build a whole family of methods that reach grokking on one and the same fixed setup. All of them bring the network to the same transition, differing in time and in trajectory (figure 1). Comparing the methods turns the question of the quantities associated with grokking — the weight norm (Liu et al., 2022b), the feature rank (Fan et al., 2024), spectral characteristics (Truong et al., 2026b), sparsity (Merrill et al., 2023) — into a measurable one: for each of them a method can be looked for where grokking arrives and the quantity behaves otherwise. Where such a method is found, this does not refute the original observations, but shows that they describe one particular path to grokking rather than the phenomenon itself; quantities that behave alike under every method become, on the contrary, candidates for genuine properties of the phenomenon. It matters here that the algorithms proposed optimize none of the candidate quantities themselves (§2).

The need for such a separation is already visible in the literature: for instance, the falling weight norm and the rising sparsity that are often taken for the mechanism of grokking turn out, in Minegishi et al. (2025), not to be a cause in themselves but a companion. It is on that very example — the weight norm (§7.2) — that I show the instrument at work; a systematic pass over all the candidates lies outside the scope of this work.

I share the view that two processes coexist in grokking (Varma et al., 2023; Merrill et al., 2023; Nanda et al., 2023; Huang et al., 2024): memorization of the training set, and the learning of an efficient representation of the input data. I show that they do not replace each other as modes of training (cf. lazy → rich in Kumar et al., 2023) but run in parallel: the generalizing circuit forms long before the visible transition, inside a stretch that the usual metrics cannot tell from continued memorization (Nanda et al., 2023).

Contributions

- I introduce three new methods for producing grokking: **artificial ricochets / controlled demolition**, **attention LR overdrive**, `W_in` **overdrive + weight normalization** — §4.1–4.3; each brings the network to grokking one to two orders of magnitude sooner than training left alone, and none of them uses weight decay;
- I single out the **state of hidden generalization**: an efficient representation has already been acquired but is not put to productive use (§7.1);
- I release **GrokkingPaths** — the testbed code on the canonical setup of Power et al. (2022), with implementations of all three of the methods proposed in the paper.

2. Related Work

A survey of the methods for producing grokking

The literature has accumulated a great many separate levers, each of which triggers, accelerates or strengthens grokking. All of them are supported by experiments, and many of them contradict one another, which is somewhat disorienting. Below I collect the ones I have met in the literature and group them by the feature that matters for this work: where exactly the lever intervenes. The full list with attribution is in appendix L.

A comparison of the paths themselves has been posed in the literature, but along two other axes. Manning-Coe et al. (2025, 2502.01739) compare the features, the compressibility and the training dynamics of networks that arrived at the same quality with grokking and without it, with the data, the architecture and the regularization unchanged, and find that the same features are learned but encoded differently: their axis is whether the network grokked. A comparison of several methods of reaching grokking against each other is not known to me, and it is that comparison this work builds.

The levers divide by **where** they intervene, and that division matters more than the list: a method that changes the loss and a method that changes the gradient transformation can produce the same curve while optimizing different quantities.

- **into the loss** — weight decay (Power et al., 2022), dropout (Liu et al., 2022a) and later kinds of penalty: $\ell\infty$ (Mohamadi et al., 2024), the logit norm (Prieto et al., 2025), spectral entropy (DeMoss et al., 2024), the geometry of representations (Kazanskii, 2026);
- **into the optimizer and the gradient transformation** — the slingshots of adaptive methods (Thilak et al., 2022), amplification of slow gradients (Lee et al., 2024), a learned transformation (Zhou et al., 2025), an orthogonalized update (Tveit et al., 2025);
- **into the numerical stability of softmax** — stepping away from the collapse in softmax (Prieto et al., 2025) and related levers;
- **into the initialization** — the scale of the initial norm (Liu et al., 2022b) and a biased initialization (Miller et al., 2024); these levers require a particular starting point and are therefore not always applicable;
- **into transfer and warm start** — transfer of an embedding from a weaker model (Xu et al., 2025), subnetworks from the generalizing phase (Minegishi et al., 2025);
- **into the architecture and the representation** — the choice of the nonlinearity and of the input representation (Salah and Yevick, 2024);
- **into the dynamics and noise** — large noise, restarts and parameter swaps that push the network out of stagnation (Furuta et al., 2024).

Changes to the task and the data themselves — the size and the fraction of the sample, the composition of the operation, the selection and the construction of the dataset, symmetry — are not counted as methods here: they change what the network has to learn and are therefore not comparable with levers applicable to a fixed task. The full catalogue of both kinds, with a column saying whether a lever requires a special initialization, is in appendix L.

3. Background / Preliminaries

3.1. Task and base configuration

Task. Modular arithmetic in the setting of the very first paper on grokking, Power et al. (2022): division `x1 / x2 (mod p) = y` at a prime `p = 97`, with 50 % of all admissible pairs used for training and the rest for validation. Equations are written as `␣x1␣␣op␣␣x2␣␣=␣␣y␣`, where every element is a token of its own carrying no information about its relations to the other tokens; loss and accuracy are computed on the answer part of the equation alone, the final token. Most of the operations present in the original paper are available in the accompanying code. Of them the measurements here use division: among the simple operations it is the one built least simply, and it groks no harder — a comparison with addition, for one, is in appendix M. Everything is taken from the original: the task, the modulus, the operation, the data fraction, the depth, the width, the number of heads in the attention block, the optimizer and its parameters, the batch size and the training budget. The experiments themselves were re-run, and divergences from the original, where they arise, are noted below. Grokking is taken to be validation accuracy reaching **0.999** (why that and not an errorless run is appendix N). This configuration is the **baseline** from here on.

Model. A decoder-only transformer with a causal mask: 2 layers, width 128, 4 attention heads (32 each), about $4{\cdot}10^5$ parameters excluding the embeddings. The FFN hidden layer is 512 (= 4·128): at 2 layers and width 128 this is the value consistent with the parameter count stated in the original paper. The K, V and Q projections are held in a single concatenated matrix, `W_QKV` below (`in_proj` in the code); Observation 1 and the method of §4.2 rest on it. The two FFN matrices I denote `W_in` (128 → 512) and `W_out` (512 → 128); in the code they are `up` and `down`. The original paper specifies the normalization order by reference — "a standard decoder-only transformer" (Vaswani et al., 2017) — that is, post-LN, `norm(x + inner(x))` after attention and after the FFN; that is what is done here. Dropout is not used. Power et al. leave the FFN activation and the initialization scheme unstated as well: the activation is ReLU, as in their published implementation and in Nanda et al. (2023); the initialization is PyTorch's default for the modules in question, with no additional scaling. A GELU variant is examined in appendix B.12.

Optimization. AdamW: learning rate 10^{-3}, weight decay 1, $\beta = (0.9, 0.98)$, a linear warm-up of the rate over the first 10 updates, a budget of 10^5 gradient steps. The batch size is 512, as in the original. Weight decay is applied to every parameter, the embeddings included; a variant that exempts the embeddings is examined in the ablations (§8).

4. Method / Approach

4.0. The classical method (reference / baseline)

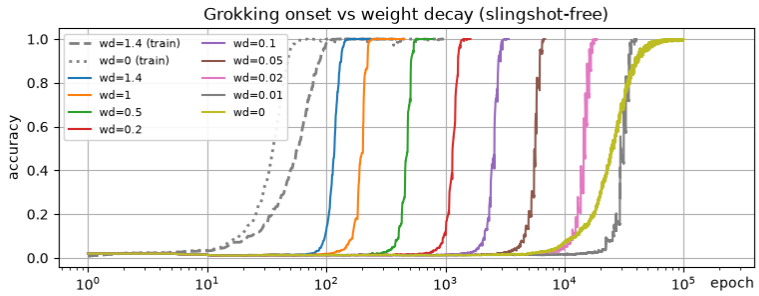


Figure 3. *Grokking on the base configuration of §3.1 at eight values of weight_decay; the epoch axis is logarithmic and the curves are cleaned of downward spikes (each point is the maximum over the last 5 epochs). The coloured curves are validation accuracy, the grey ones the train curves of the two extreme runs, shown up to 10^3 epochs.*

The reference algorithm for comparison (the baseline) is grokking as Power et al. (2022) described it: the same task, the same configuration, the same weight regularization.

As the original work shows, fast grokking needs nothing beyond a choice of hyperparameters — a large weight_decay (within reason) shortens the delay sharply. The inverse dependence of the delay on the regularization is known: Liu et al. (2022b) derive $t \propto 1/wd$ from their mechanism and confirm it empirically; Truong et al. (2026a) test the same law directly and qualify it, finding that it holds only approximately. At wd = 1.4, half of what is visible as grokking falls in a period when the training set has not yet been learned to the end: validation accuracy passes 50 % at epoch 118, and train reaches 100 % only at epoch 121. At the upper edge of the weight_decay range, that is, the baseline's delay between memorization and generalization is compressed until the two processes overlap.

4.1. Artificial ricochets — controlled demolition

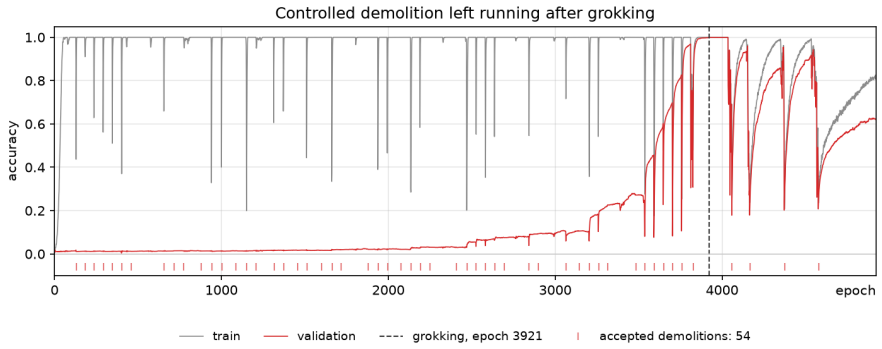


Figure 4.1-a. *Training with artificial demolitions and no further limiters: after grokking is reached, the demolitions continue. weight_decay = 0, seed 53; the marks under the axis are 54 accepted demolitions, 4 of them after grokking. After the very first of those, validation accuracy never returns to 0.999.*

The method the paper is named for. At certain moments the network is demolished deliberately — under control, no harder than a set limit, and in a chosen direction — and is then left to recover. Weight regularization is switched off entirely (weight_decay = 0): the demolitions bring grokking

about reasonably quickly on their own. Brief collapses of a network, which in deep learning happen of their own accord and are usually fought, are here provoked instead, at the moment they are wanted.

The scheme of an artificial demolition.

- 1.1. Save a checkpoint, the optimizer's state included. The order in which batches are served is NOT fixed — returning to the checkpoint (step 1.4) undoes the damage alone (appendix B.1).
- 1.2. Escalate the learning rate — **LR escalation** below: x2 of the base lr on the current step; x2.5 on the next, then x3, x3.5 and so on.
- 1.3. …until the network shows a train accuracy worse than the threshold D1 on some batch; then restore the original rate and remember the number of that step.
- 1.4. Then a relaxation period: over R steps, watch the train accuracy within the batch. If it turns out worse than a second threshold D2 (D2 < D1: this threshold is the stricter one), return to the checkpoint and escalate the LR as before but one step less far. This is in case the escalation has damaged the network and it goes on breaking at the normal rate.
- 1.5. Repeat until a pass leaves train accuracy better than D2 for the whole of the relaxation period. Better a demolition that turns out weak than one that ruins the network irreversibly.
- 1.6. Then leave the network to recover.

Limiters. A demolition begins only at an epoch boundary, and only when three conditions hold at once: train accuracy has reached the threshold TA1 = 0.99, train loss has stopped improving, and at least 50 epochs have passed since it first did. The exact definitions, and the rules by which the conditions are reset after a demolition, are in appendix B.6.

If the demolitions are not switched off once grokking is reached, they break the network fairly quickly — see figure 4.1-a; what cures that is in appendix B.11.

The phenomenon: a ricochet

What the method provokes on purpose, a network trained with regularization goes through on its own. Anyone who has ever looked at training curves has seen these spikes: the training loss jumps by several orders of magnitude and then slides back down. They are usually taken for **slingshots** (Thilak et al., 2022, 2206.04817); I call the phenomenon a **ricochet** here and tell it apart from a slingshot, as shown below, by three signs. The angular metric — the cosine of the angle between the gradients on neighbouring batches (appendix S) — rises several epochs before the event and goes far past 90°: neighbouring gradients point towards one another, that is, the network is thrashing between the walls of the landscape, and an angle past the right angle is naturally read as a rate too large for the stretch of landscape the network has found itself on. Immediately after the demolition the metric falls to about 40°, to a confident directed descent (appendix O.1). The norm of the last layer's weights, meanwhile, changes little and smoothly. And besides, moving the computation to fp64 stops the classical slingshots of Thilak altogether, or pushes them far into the future, while it has almost no effect on ricochets (appendix B.10).

That is where the scheme of the previous section comes from. Escalating the learning rate (step 1.2) is not an arbitrary choice: it is precisely that state — a rate too large for the stretch of landscape occupied — that the angular metric shows in the steps leading up to a collapse. Escalating the rate brings the network to a similar state, and in that sense steps 1.2-1.5 are not a method of breaking a network by any means available, but a method of reproducing the events that weight_decay produces. A network can also be collapsed by an arbitrary movement: a random punch at the same thresholds does bring generalization about, and sooner at that, but on none of the nine seeds does it reach 99.9 %, where the escalation reaches it on eight (appendix O.2).

A ricochet is not a slingshot (Thilak et al., 2022, 2206.04817), though both are a temporary destruction of the network with a quick recovery. What separates them above all is the norm of the last layer's weights — the sign by which Thilak et al. identify a slingshot — and several further signs besides; the full comparison by phase is appendix O.3.

4.2. Attention LR overdrive

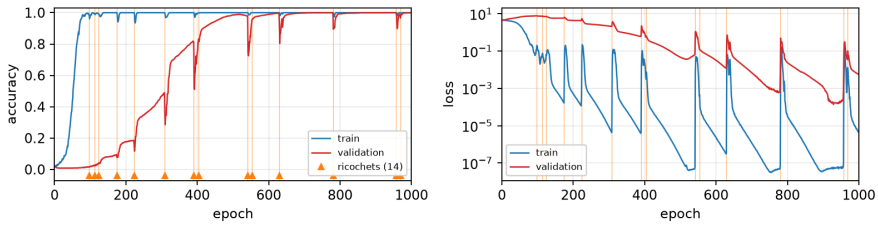


Figure 4.2-b. *Attention overdrive on the baseline of §3.1 with no regularization:* `W_QKV` *is given nine times the learning rate, everything else untouched. Orange marks the natural ricochets (§4.1) that the method produces on its own. Memorization comes at epoch 138, grokking at 719; every step up in validation accuracy sits on a ricochet. Why this particular run was chosen is in appendix B.2.*

The second new method rests on two observations.

Observation 1 (localizing weight_decay). If weight_decay is applied not to every matrix of the network but only to some, it turns out that they are not equally necessary for grokking. An honourable second place, by reliability and by how easily grokking is obtained, goes to the embedding matrices at the very beginning of the network — which some works, by an established habit, do the opposite with and exempt from weight_decay, which is amusing. First place, by a narrow margin, goes to the concatenated attention matrices `W_QKV`. Even acting with weight_decay on their **V** component alone produces grokking in most cases. On the strength of these observations one may try to get the result by working with the `W_QKV` matrices only.

The full hierarchy of that test — from "every parameter" down to one third of one block's `W_QKV`, 261 runs over nine seeds — is taken apart in appendix R.

Observation 2 (the path under regularization is near-constant). The path the network travels under the regularization alone before grokking barely changes while weight_decay, and with it the epoch at which grokking arrives, changes by two orders of magnitude (appendix B.9).

The method: attention overdrive. From these observations the second method follows: the attention matrix should be given more opportunity to learn than the other layers. That is what I call **attention LR overdrive**: the optimized parameters are split into param_groups and the `W_QKV` matrix is set aside in a group of its own that receives 9 times the learning rate. The multiplier is chosen from the map in appendix B.2: the optimum is flat, and x9 is the only step at which grokking arrives in all nine runs. The same multiplier is used for column 4.2 in the invariants table of §8.1.

Giving different matrices sharply different rates is not in itself new: the record runs of the modded-nanogpt competition are built that way (Jordan et al., 2024-2026) — different matrices are assigned different rates there, and within a single optimizer the effective rates differ by anything from a factor of two to two orders of magnitude. The justifications offered for this vary widely, from theoretical arguments based on the sizes of the matrices and the batches (You et al., 2017, 1708.03888; You et al., 2020, 1904.00962) to dynamic selection of the multiplier from elaborate and largely empirical diagnostics (Atallah, 2026, 2607.09266). What is new here is the crystalline simplicity of the experiment, which follows from the observation.

On the testbed of §3.1, with no regularization whatever, the method brings the network to grokking where the untouched baseline takes 53152 epochs by the median of nine seeds (appendix B.7): at a multiplier of x9, grokking arrives in all nine runs with a median of 931 epochs — almost sixty times faster. The plateau stays real: memorization comes by the hundredth epoch, generalization seven hundred-odd epochs later, so every classical sign of grokking is preserved.

Reliability. The method does not always work: of 117 runs at multipliers from x2 to x20, grokking arrived in 87; the working range is from x3 to x20, with a flat optimum at x9-x10 and a gradual loss of reliability in both directions. Both kinds of failure, the efficacy of each lever and the limits of the method are in appendices B.2-B.4.

What the failure is good for. The kind of failure in which the network memorizes the training set confidently and stays in semi-grokking is lifted not by changing the rate but by raising the variance of the gradient at the same rate — the same move as in §4.1. Semi-grokking, then, is not the absence of a generalizing solution but memorization in a state that is easy to destroy: under destabilization the generalization comes out. This is one more piece of evidence for the hidden accumulation (§7.1), obtained from another side — not by measuring checkpoints, but by the fact that a network standing still by its metrics yields to a push.

What makes the method valuable. This method is particularly good in that it satisfies every classical criterion of grokking while doing no damage to the network along the way — which makes it especially useful for an ablation whose purpose is to separate the properties of grokking itself from the properties of the methods that bring it about.

4.3. `W_in` overdrive with weight normalization

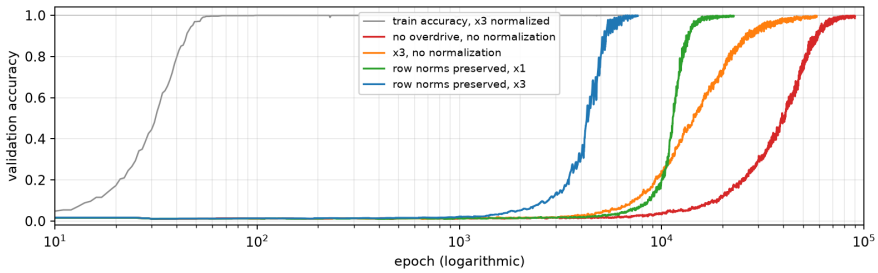


Figure 4.3-a. *What preserving the row norms of* `W_in` *buys, with the overdrive and without it. All four runs are on the baseline of §3.1 at weight_decay = 0 and at the same data and model seeds, 71; the red line is untouched training, grokking at epoch 89089; the orange one is the x3 overdrive alone, without normalization: 57344; the green one is normalization alone, without the overdrive: 21702; the blue one is the two together: 6616. The grey line is the training accuracy of the blue run: memorization is over by the hundredth epoch, and everything after it happens on the plateau. Normalization on its own shortens the path a great deal; the overdrive without it also does, though less, and destabilizes training noticeably; together they give a manyfold speed-up. The gains at other seeds, and the details, are in appendix B.5.*

The same hierarchical ablation of weight_decay over individual matrices (appendix R, table R-a) points at other candidates for a raised learning rate as well — at the **input matrices of the FFN blocks,** `W_in` (128 → 512). This is of interest also under the conjecture that grokking calls for moving the processing of the signal to layers as early as possible. Giving them any appreciable overdrive, however, does not work: at a large multiplier training destabilizes. The first natural move — raising `betas[0]` — has too little effect, so what is used here is **weight normalization** (Salimans and Kingma, 2016): the vector of input weights of every output neuron is brought back, after every optimizer step, to the norm that row received at initialization, so that training changes only the direction of the vector and the starting point shared by all the methods (§3.1) is left intact. How the device works formally, and how it differs from normalizing the whole network in Liu et al. (2022b), is in appendix B.5.

The method of the previous section is easy to misread: since an overdrive of `W_QKV` produces grokking, it is easy to conclude that attention is what matters, and that it is the only thing that does. That is the motive of this section: to reach grokking by overdrive without touching attention at all. It is reachable, and from that follows the conclusion the method was introduced for: what matters is not the attention mechanism — many networks that grok perfectly well have none — but the accelerated training of the early matrices, that is, getting an effective representation of the data to form as fast as possible, in whichever matrix it takes shape.

What normalization gives and what the overdrive gives. The method was measured on nine seeds at eight multipliers, 72 runs in all (§8.5; the full table is in appendix B.5). What they add up to is a picture in which the main work is done by the normalization and not by the overdrive.

Normalization alone, with no overdrive whatever, brings the network to grokking in eight runs of nine — three times faster than untouched training. An overdrive on top of it speeds the transition up by a further factor of two and a half: at a multiplier of x3, grokking arrives in all nine runs and eight times earlier than in the baseline without intervention.

The overdrive **without** normalization gives almost nothing (appendix B.5). The curves show why: without normalization a run passes through hundreds of natural ricochets (§4.1) and rises gently, whereas with it the same seed covers the path in one short steep climb and through a handful of collapses. That is, normalization does not open access to high learning rates, as one might have thought — it **orders** the process, keeping the overdrive from tearing the network apart faster than it can generalize.

Combining the methods. All the methods introduced in this paper can be combined with one another; at small overdrive values the source of the perturbation can be artificial ricochets (§4.1). With a long enough search over hyperparameters one can even arrange for the transition from complete overfitting to complete grokking to happen after a single ricochet.

6. Experimental Setup

Every measurement was made on nine arbitrarily chosen and always identical seeds — the primes 37, 41, 43, 47, 53, 59, 61, 67, 71. The seed fixes both the split of the data and the initialization of the model, so a difference between two runs on one seed is a difference of the intervention and not of the sample. The spread across seeds is wide: with no intervention at all, grokking arrives anywhere between epoch 30757 and epoch 103491 — which is why a claim tested on one seed counts as untested here, and why, wherever a single number is given over nine seeds, it is the median with the range beside it. All marks and budgets are in epochs (at a batch of 512 an epoch is ten optimizer steps). The remaining details of the setup are in appendix C.

7. Observations

The methods of §4 give what none of them gives alone: several independent paths to one and the same transition of the state of the network, on one and the same testbed. That turns the question "what happens during grokking" into a question that can be answered by comparison — a quantity that behaves alike on every path belongs to the phenomenon, and one that diverges describes the method that produced it.

The paper has two observations of this kind. They are given as an **example of what is possible, not as a list of it**: any quantity the testbed already records, or that you can add to it, can be compared, and on any set of paths — of which §4 has four, plus untouched training as a control.

7.1. The hidden accumulation of an efficient representation of the data

The measurement is set up as follows: the network trains in the ordinary way, and at a chosen point along the way the controlled demolitions (§4.1) are added to the devices already at work — the methods compose with one another without interference — and from there the count is of how many epochs they take to bring validation accuracy to its maximum. The runs are the baseline of §3.1, nine seeds. There are two scales of measurement, and they were taken on different regularization families, which the weight_decay row of the table notes: the numbers of one scale are not directly comparable with the numbers of the other.

The points of measurement are set **relatively** rather than by epoch number, and that is forced: the key events float widely from seed to seed. A settled 10 % is reached between epoch 6579 and 11856, a settled 32 % between 7992 and 14307; at the moment the fastest seed already holds 32 % of the validation set, the slowest has not reached 10 %. Averaging such runs by epoch number is meaningless, so the stretch from memorization to a settled 10 % is cut into quarters and the measurement goes in fractions of the way.

Both scales in one table. On the left the point is the fraction of the way from memorization (0) to a settled 10 % (100); on the right the validation accuracy itself, at whose level the checkpoint is taken; the head of the table names the family.

	wd = 0.01					wd = 0					
point, %	0	25	50	75	100	2	4	8	16	24	32
val acc	0.013	0.018	0.021	0.043	0.108	0.02	0.04	0.08	0.16	0.24	0.32
gap, epochs	3087	1909	1302	978	754	4066	3041	2113	1291	1056	948

A network that answers correctly on a tenth of the held-out set looks helpless. But more than half of the work is done by then, and it shows in how much labour is left to the controlled demolitions: from that point they bring the network to its maximum in 754 epochs by the median of nine seeds, whereas starting from the very beginning the demolitions need four times as many — 3087.

Both scales give one and the same regularity, and it is monotone: the further the network has gone, the less work is left to the demolitions, and the gap shrinks between the end points by roughly a factor of four — from 3087 epochs to 754 on the first scale and from 4066 to 948 on the second. There are no exceptions on either scale: even demolitions switched on right after memorization need more epochs than those switched on a quarter of the way in. Medians over nine seeds; the full tables by run are in appendix P.

7.2. Weight norms at grokking

A great body of work ascribes a considerable role in grokking to the change of the weight norm. On our testbed that connection turns out to be ambiguous, to the point of being reversed: under regularization the norm falls — the theories of a "Goldilocks zone" are built on this — under controlled demolitions it rises sharply at the moment of grokking, and under both overdrives the norm of the embedding matrices does not change at all, although on the baseline it makes the largest contribution to the fall and, as the ablation shows (appendix R), it alone suffices to bring grokking about. The measurements across all the methods, broken down by groups of matrices, are in appendix Q.

9. Discussion

An alternative interpretation: metastability. Ersoy and Wiesner (2026) describe delayed generalization as the network being trapped in a metastable state, which it leaves when SGD noise carries it across an energy barrier. Part of what they observe agrees with what I see, but I do not share the interpretation: with no perturbation at all the network on this testbed reaches generalization on its own — a median of 53152 epochs over nine seeds (appendix B.7) — and runs that have not grokked mostly do not stand still but go on rising slowly. Their work, and what of it is visible on this testbed, is set out in appendix T.

A practical consequence. Leaving the state of hidden generalization calls for a mechanism that displaces it: by Nanda et al. (2023) slingshots are not necessary when weight decay is present, but in its absence it is they that play the part of an implicit regularizer. In the ordinary practice of training large networks, spikes of that kind are suppressed deliberately — by gradient clipping, by lowering the learning rate, by constraining norms — or else they do not arise at the chosen hyperparameters. Hence a conjecture: in some such networks hidden generalization may be present and never show itself, and those training the model do not know of it. Testing the conjecture is outside the scope of this work.

10. Limitations

- No systematic test of the candidates for invariants of grokking was carried out: one quantity, the weight norm (§7.2), is given as an example. The other candidates — the feature rank, spectral characteristics, sparsity — are outside the scope of this work.

11. Conclusion / Future Work

The obvious continuation is to run, on this testbed, every hypothesis about the properties of grokking mentioned in the paper, and those that can be gathered from the literature.

Reproducibility Statement

The setting, the model and the hyperparameters are in §3.1; the details of the setup, the seeds and the bitwise reproducibility of the runs are in appendix C; the qualifier of the original that is not reproduced here, on the batch size, is in appendix D; the definition of the grokking threshold is in appendix N. The full tables over all the runs the claims of the paper rest on are in appendices B, O, P, Q and R. The testbed code, with all three of the proposed methods, is attached to the submission; it also contains the scripts that build every figure of the paper.

AI Use Statement

Generative language models were used in preparing this work, and this is where.

- **Translation.** The text of the paper was translated into English with the help of a language model. Every translated block was checked against the source for numbers, references, quotations and names, and divergences were corrected.
- **Editing.** Shortening sections to the page limit, removing repetitions, moving material into the appendices and rewording were done with the help of a model on the author's instructions. Every wording is the author's own.
- **Work with the literature.** Searching for sources, checking quotations against the originals, and verifying that a citation does not strengthen what the cited work claims were done with the help of a model; the conclusions were checked against the texts of the works themselves.
- **Code and typesetting.** The auxiliary scripts — figure rendering, the typesetting build, the page count, the preparation of the attached code archive — were written and revised with the help of a model.

The statement of the problem, the design of the three proposed methods, the construction of the experiments and the interpretation of the results are the author's. The author is responsible for the whole content of the work, including those parts in whose preparation language models were used.

References

- Kesten, 1958. "Accelerated Stochastic Approximation". The Annals of Mathematical Statistics 29(1), 41–59. DOI:10.1214/aoms/1177706705.
- Mirzakhmedov F., Uryasev S. P., 1983. "Адаптивное регулирование шага для алгоритма стохастической оптимизации". Zh. Vychisl. Mat. Mat. Fiz. 23(6), 1314–1325; English translation: USSR Comput. Math. Math. Phys. 23(6), 20–27. DOI:10.1016/S0041-5553(83)80071-8. (text not obtained; cited after Pflug, 1988)
- Pflug G. Ch., 1983. "On the Determination of the Step Size in Stochastic Quasigradient Methods". IIASA CP-83-025.
- Chan L. W., Fallside F., 1987. "An adaptive training algorithm for back propagation networks". Computer Speech & Language 2, 205–218. DOI:10.1016/0885-2308(87)90009-X. (text not obtained; cited after Hassoun, 1995)
- Pflug G. Ch., 1988. "Stepsize Rules, Stopping Times and their Implementation in Stochastic Quasigradient Algorithms". In: Ermoliev, Wets (eds.), Numerical Techniques for Stochastic Optimization, Springer, 353–372. DOI:10.1007/978-3-642-61370-8_17.
- Hassoun M. H., 1995. "Fundamentals of Artificial Neural Networks". MIT Press, sec. 5.2.
- Salimans and Kingma, 2016. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". arXiv:1602.07868 (NeurIPS 2016).
- Chee J., Toulis P., 2018. "Convergence diagnostics for stochastic gradient descent with constant learning rate". arXiv:1710.06382 (AISTATS 2018).
- You, Gitman and Ginsburg, 2017. "Large Batch Training of Convolutional Networks". arXiv:1708.03888.
- You et al., 2020. "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes". arXiv:1904.00962 (ICLR 2020).

- Power et al., 2022. “Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets”. arXiv:2201.02177.
- Liu et al., 2022a. “Towards Understanding Grokking: An Effective Theory of Representation Learning”. arXiv:2205.10343.
- Thilak et al., 2022. “The Slingshot Mechanism: An Empirical Study of Adaptive Optimizers and the Grokking Phenomenon”. arXiv:2206.04817.
- Liu et al., 2022b. “Omnigrok: Grokking Beyond Algorithmic Data”. arXiv:2210.01117.
- Gromov, 2023. “Grokking modular arithmetic”. arXiv:2301.02679.
- Nanda et al., 2023. “Progress Measures for Grokking via Mechanistic Interpretability”. arXiv:2301.05217 (ICLR 2023).
- Merrill et al., 2023. “A Tale of Two Circuits: Grokking as Competition of Sparse and Dense Subnetworks”. arXiv:2303.11873.
- Notsawo et al., 2023. “Predicting Grokking Long Before it Happens: A look into the loss landscape of models which grok”. arXiv:2306.13253.
- Varma et al., 2023. “Explaining grokking through circuit efficiency”. arXiv:2309.02390.
- Rubin et al., 2024. “Grokking as a First Order Phase Transition in Two Layer Networks”. arXiv:2310.03789.
- Kumar et al., 2023. “Grokking as the Transition from Lazy to Rich Training Dynamics”. arXiv:2310.06110.
- Miller et al., 2024. “Grokking Beyond Neural Networks: An Empirical Exploration with Model Complexity”. arXiv:2310.17247.
- Minegishi et al., 2025. “Bridging Lottery Ticket and Grokking: Understanding Grokking from Inner Structure of Networks”. arXiv:2310.19470.
- Tan et al., 2023. “Understanding Grokking Through A Robustness Viewpoint”. arXiv:2311.06597.
- Lyu et al., 2024. “Dichotomy of Early and Late Phase Implicit Biases Can Provably Induce Grokking”. arXiv:2311.18817 (ICLR 2024).
- Huang et al., 2024. “Unified View of Grokking, Double Descent and Emergent Abilities: A Perspective from Circuits Competition”. arXiv:2402.15175.
- Furuta et al., 2024. “Towards Empirical Interpretation of Internal Circuits and Properties in Grokked Transformers on Modular Polynomials”. arXiv:2402.16726.
- Bhaskar et al., 2024. “The Heuristic Core: Understanding Subnetwork Generalization in Pretrained Language Models”. arXiv:2403.03942.
- Wang et al., 2024. “Grokked Transformers are Implicit Reasoners: A Mechanistic Journey to the Edge of Generalization”. arXiv:2405.15071.
- Fan et al., 2024. “Deep Grokking: Would Deep Neural Networks Generalize Better?”. arXiv:2405.19454.
- Lee et al., 2024. “Grokfast: Accelerated Grokking by Amplifying Slow Gradients”. arXiv:2405.20233.
- Mohamadi et al., 2024. “Why Do You Grok? A Theoretical Analysis of Grokking Modular Addition”. arXiv:2407.12332.
- Clauw et al., 2024. “Information-Theoretic Progress Measures reveal Grokking is an Emergent Phase Transition”. arXiv:2408.08944.
- Beck et al., 2024. “Grokking at the Edge of Linear Separability”. arXiv:2410.04489.
- Wen et al., 2024. “Understanding Warmup-Stable-Decay Learning Rates: A River Valley Loss Landscape Perspective”. arXiv:2410.05192.
- Salah et al., 2024. “Controlling Grokking with Nonlinearity and Data Symmetry”. arXiv:2411.05353.
- DeMoss et al., 2024. “The Complexity Dynamics of Grokking”. arXiv:2412.09810 (Physica D, 2025).
- Prieto et al., 2025. “Grokking at the Edge of Numerical Stability”. arXiv:2501.04697.
- Manning-Coe et al., 2025. “Grokking vs. Learning: Same features, different encodings”. arXiv:2502.01739.
- Gu et al., 2025. “Beyond Progress Measures: Theoretical Insights into the Mechanism of Grokking”. arXiv:2504.03162.
- Xu et al., 2025. “Let Me Grok For You: Accelerating Grokking via Embedding Transfer from a Weaker Model”. arXiv:2504.13292.
- Tveit et al., 2025. “Muon Optimizer Accelerates Grokking”. arXiv:2504.16041.

- Zhou et al., 2025. “NeuralGrok: Accelerate Grokking by Neural Gradient Transformation”. arXiv:2504.17243.
- Abramov et al., 2025. “Grokking in the Wild: Data Augmentation for Real-World Multi-Hop Reasoning with Transformers”. arXiv:2504.20752 (ICML 2025).
- Prakash et al., 2025. “Grokking and Generalization Collapse: Insights from HTSR theory”. arXiv:2506.04434.
- Notsawo et al., 2025. “Grokking Beyond the Euclidean Norm of Model Parameters”. arXiv:2506.05718.
- Walker et al., 2025. “GrokAlign: Geometric Characterisation and Acceleration of Grokking”. arXiv:2506.12284.
- Li et al., 2025. “Grokking in LLM Pretraining? Monitor Memorization-to-Generalization without Test”. arXiv:2506.21551.
- Salah et al., 2025. “Tracing the Path to Grokking: Embeddings, Dropout, and Network Activation”. arXiv:2507.11645.
- Han et al., 2025. “Flatness is Necessary, Neural Collapse is Not: Rethinking Generalization via Grokking”. arXiv:2509.17738.
- Tian et al., 2025. “Provable Scaling Laws of Feature Emergence from Learning Dynamics of Grokking”. arXiv:2509.21519.
- Saheb Pasand et al., 2026. “Egalitarian Gradient Descent: A Simple Approach to Accelerated Grokking”. arXiv:2510.04930.
- Hutchison and Yevick, 2025. “Grokking in the Ising Model”. arXiv:2510.25966.
- Musat, 2025. “The Geometry of Grokking: Norm Minimization on the Zero-Loss Manifold”. arXiv:2511.01938.
- Singh et al., 2025. “When Data Falls Short: Grokking Below the Critical Threshold”. arXiv:2511.04760.
- Jiang et al., 2026. “On the Convergence Behavior of Preconditioned Gradient Descent Toward the Rich Learning Regime”. arXiv:2601.03162.
- He et al., 2026. “Is Grokking Worthwhile? Functional Analysis and Transferability of Generalization Circuits in Transformers”. arXiv:2601.09049.
- Prakash and Martin, 2026. “Late-Stage Generalization Collapse in Grokking: Detecting anti-grokking with WeightWatcher”. arXiv:2602.02859.
- Beck et al., 2026. “The Implicit Bias of Logit Regularization”. arXiv:2602.12039.
- Hwang et al., 2026. “Intrinsic Task Symmetry Drives Generalization in Algorithmic Tasks”. arXiv:2603.01968.
- Yıldırım, 2026. “The Geometric Inductive Bias of Grokking: Bypassing Phase Transitions via Architectural Topology”. arXiv:2603.05228.
- Truong et al., 2026a. “The Norm-Separation Delay Law of Grokking: A First-Principles Theory of Delayed Generalization”. arXiv:2603.13331.
- Acharya et al., 2026. “Grokking as a Variance-Limited Phase Transition”. arXiv:2603.15492.
- Manir and Rupa, 2026. “A Systematic Empirical Study of Grokking: Depth, Architecture, Activation, and Regularization”. arXiv:2603.25009.
- Zhang et al., 2026. “Grokking: From Abstraction to Intelligence”. arXiv:2603.29262.
- Tomàs et al., 2026. “Breaking Data Symmetry is Needed For Generalization in Feature Learning Kernels”. arXiv:2604.00316.
- Truong et al., 2026b. “Spectral Entropy Collapse as a Phase Transition in Delayed Generalisation: An Interventional and Predictive Framework for Grokking”. arXiv:2604.13123.
- Liu et al., 2026. “Grokking or Glitching? How Low-Precision Drives Slingshot Loss Spikes”. arXiv:2605.06152.
- Ersoy and Wiesner, 2026. “Noise-Driven Escape from Metastable Phases explains Grokking in Deep Neural Networks”. arXiv:2606.17120.
- Tiwari et al., 2026. “Radial Suppression Accelerates Algorithmic Generalization: A Geometric Analysis of Delayed Generalization”. arXiv:2606.32000.
- Atallah, 2026. “LionVote: Per-Layer Learning Rate Adaptation for Lion”. arXiv:2607.09266.
- Kazanskii, 2026. “How to Tame Grokking: Representation Geometry as a Control Signal”. arXiv:2607.11666.
- Pandey, 2026. “Thermodynamic Weight Decay: Exploring Grokking Acceleration via Attention Specific Heat”. arXiv:2607.20552.

- Jordan and the competition participants, 2024–2026. "modded-nanogpt: the NanoGPT speedrun". GitHub repository, commit `ecbb586` of 9 August 2026; the record log cited is the FP8DownProjection record of 2026-07-17 under `records/track_1_short/` (the rate multipliers are in the table `param_table` of that log). `https://github.com/KellerJordan/modded-nanogpt/tree/ecbb586296d3dac36fd206211f25d63bad4a6b35` (a software artefact rather than a publication: cited as of the commit named)

Appendix

B. Further experiments and ablations

B.1. Why the order of the batches must not be restored for the escalation

Step 1.1 of the scheme of §4.1 requires that, for the escalation, the return to the checkpoint should undo the damage but not return training to the same data — whereas for the random punch, on the contrary, it returns everything. The requirement looks arbitrary, and asymmetrical besides, so here are the measurements it follows from, and a check of why the second method has no such requirement. All the numbers were taken on the baseline of §3.1 at weight_decay = 0, D1 = 0.25, D2 = 0.1, and a relaxation window of 20 optimizer steps.

What happens inside an escalation. The scheme measures damage by train accuracy, and on this baseline such a measure is almost blind. Below is a control batch of 512 examples, measured after every step of the fourth escalation of a run (epoch 281):

step	LR multiplier	train loss	accuracy	p(correct token), median	p(predicted), median
36	×20.0	$8.0\cdot10^{-6}$	1.0000	1.00	1.0000
39	×21.5	$1.0\cdot10^{-5}$	1.0000	1.00	1.0000
40	×21.5	$1.3\cdot10^{-2}$	1.0000	1.00	1.0000
41	×22.0	$1.58\cdot10^{1}$	0.0208	$2.1\cdot10^{-7}$	0.9787

For forty steps in a row the escalation does not spoil the answers at all: accuracy holds exactly at one. On the forty-first the network loses everything at once — and loses it not into noise but into another confident state: 71 % of the examples are given one and the same token, and the confidence in it stays as it was (median 0.979). The reason it all happens at once is that the shift of the weights at a step is common to every example: while it is smaller than the gap between the correct logit and the rest, the argmax changes for no example at all, and the moment the gap is covered, it is covered for all of them at once.

The escalation first deepens the minimum. Look at the train loss in the same table, and at the summary over the first thirteen demolitions: before the break the loss does not rise but falls, sometimes orders of magnitude below the one the escalation started from (demolition 6: $2.9\cdot10^{-4}$ at the start, $6.2\cdot10^{-8}$ at the last step but one, at a multiplier of ×307). That is, a learning rate set too high first drives the network deeper into the same minimum and only at the edge of stability throws it out — in a single step, from a loss of 10^{-8} to a loss of 42.

The size of that ejection has no analogue in ordinary training. Over 4896 steps taken outside demolitions at train accuracy = 1.0, the change in $\log_{10}$(loss) per step is -0.040 ± 0.124 (that is, a step usually changes the loss by a factor of 1.33), while the largest deviation observed is +0.655, or 5.6 σ. The last steps of the escalations give from +0.50 (4.4 σ) at the first demolition to +6.92 (56 σ) at the sixth: the longer the network has been training, the sharper the break.

Hence the requirement on the rollback. Restoring the order of the batches makes the repeated pass deterministic, and that is not a merit but a breakage of the scheme. If the return to the checkpoint restores the order of the batches as well, the repeated pass "one step shorter" travels exactly the same

path over the same data and stops exactly short of the break. It is guaranteed not to damage the network, the monitor accepts it, and the demolition is counted without having taken place. In the run with the order restored there turned out to be 90 such empty demolitions out of 147; the mean gain in validation accuracy over the 100 epochs after them is +0.003 against +0.105 after the ones that did take place.

The price is the time to grokking. At identical thresholds and one and the same algorithm:

rollback policy	grokking	demolitions before it	of them, ones that took place
the order of the batches is restored	epoch 10825	138	29 (one per 303 epochs)
the order of the batches is not restored	epoch 3890	53	34 (one per 57 epochs)

Demolitions that take place come five times as densely, and generalization arrives 2.8 times earlier. The difference lies entirely in whether the repeated pass meets new data.

Why this does not happen with the punch. The same policy, applied to the random punch, changes nothing. On the paper's nine seeds, with the order of the batches restored and without it, the outcome coincides on every seed — semi-grokking everywhere, not one run reaches 99.9 % — the medians of the marks diverge by 2–6 % (50 % at epoch 1234 against 1294), and the sign test on that mark gives p = 1.00: four seeds faster, five slower. There is not a single empty demolition: all 1376 accepted punches lie within the damage corridor, and every demolition gives the same +0.40 of validation accuracy over the hundred epochs after it as it does without restoration.

The reason is that the punch has no such breaking mechanism as the one in the paragraph above. Its repeated pass does not shorten the path by a step but halves the **amplitude**, so that it is not a deterministic neighbour of the rejected attempt and does not skirt the break. And the direction of the punch is drawn by a generator of its own, seeded from the model seed and the number of the demolition, which does not read the global RNG state at all — that is, restoring the order of the batches does not affect it even indirectly. Hence the asymmetry of step 1.1: reproducibility of the data is precious where it comes free, and forbidden where it costs a factor of 2.8.

A remark on the measure of damage. On this baseline the threshold D1 on train accuracy distinguishes only two states — "every answer correct" and "every answer wrong" — between which, at the late demolitions, there lies a single step. Train loss, by contrast, changes continuously and shows the break coming a step before it (in the table above: a rise of three orders of magnitude at unchanged accuracy). For implementations of the scheme on tasks where such blindness is likely, thresholds on the loss are preferable — see the remark at the end of §4.1.

An exception to the rollback: grokking inside a demolition. The rollback returns not only the weights and the state of the optimizer but the record of the run as well: the snapshot that is rolled back to contains the whole series of measurements. Usually this is a matter of indifference, because a demolition lies either entirely before the 0.999 mark or entirely after it. But a demolition takes tens of epochs, and if validation accuracy crosses the threshold precisely inside one, and the relaxation then turns out to be violated, the rollback erases the crossing itself: the network returns to the state before it, and in the record of the run there is no grokking — although there was. The testbed therefore makes one exception: a demolition that began before the first crossing and discovered it by the time it was resolved is counted rather than repeated. The rule fires no more than once per run — after it, grokking is already known on entry to every subsequent demolition — does not touch demolitions that began after the mark, and acts in the same way for both methods of this section. The price is that the network stays damaged to the end of the run, which in the scheme of §4.1 stops 10^3 epochs after the mark anyway; whoever needs an undamaged network takes the checkpoint in which there is one.

The case really is rare: of the 565 demolitions of the nine seeds of appendix B.6 exactly one covers the mark. But when it does occur, without this exception the run loses precisely what it was made for. In the testbed's control run where this happened, the record breaks off at epoch 3496 with a maximum validation accuracy of 0.998711 — the threshold is nowhere in it — and yet the run was

stopped, because the stopping watches its own latch and not the record. The same run with this exception goes epoch for epoch along the same path up to the firing of the rule, and the crossing of 0.999 at epoch 3296 stays in the record.

B.2. Attention LR overdrive: the full map over multipliers

The baseline of §3.1 with no regularization (weight_decay = 0), nine seeds — the primes from 37 to 71 — a budget of $2 \cdot 10^4$ epochs, stopping 10^3 epochs after grokking. 117 runs in all. "Memorization" is the first epoch with train accuracy = 1, "grokking" the first epoch with validation accuracy above 0.999 (§3.1), the "plateau" the distance between them. Medians over the runs that grokked, with the range in brackets.

multiplier	grokked	grokking	range	memorization	plateau	failures
×2	0/9	—	—	109	—	9 semi-grokking
×3	7/9	3459	2042–8370	187	3259	2 semi-grokking
×4	8/9	2130	1391–6208	181	1961	1 semi-grokking
×5	6/9	1679	676–5074	124	1542	3 semi-grokking
×6	6/9	2812	1426–9354	137	2665	3 semi-grokking
×7	7/9	2670	2071–7745	118	2371	2 semi-grokking
×8	8/9	1513	467–5496	109	1401	1 semi-grokking
×9	**9/9**	**931**	304–9670	94	704	—
×10	8/9	962	326–1744	106	826	1 non-convergence
×11	7/9	1775	103–5419	99	1652	1 semi-grokking, 1 non-convergence
×12	6/9	2480	273–7836	128	2320	2 semi-grokking, 1 non-convergence
×15	8/9	1076	159–8190	124	924	1 non-convergence
×20	7/9	4952	460–7281	158	4816	2 semi-grokking

For comparison: the same baseline without overdrive and without regularization passes 0.999 at a median epoch of 53152 over the nine seeds (appendix B.7), so the best step is 57 times faster. Success depends both on the multiplier and on the seed, and the two dependencies do not reduce to one another: the seed that is fastest at one multiplier may turn out the worst at the one next to it.

The choice of the run for figure 4.2-b. Seeds on one step diverge widely: on the working step ×9, grokking takes from 304 to 9670 epochs. To show any one of them "by eye" is to show a random point of the distribution, so the run for the figure was chosen formally — by the sum of the relative deviations from the step's medians across three quantities at once: the epoch of grokking, the length of the plateau and the number of ricochets.

seed	grokking	plateau	ricochets	total deviation
59	719	581	19	**0.402**
67	931	704	31	0.632

seed	grokking	plateau	ricochets	total deviation
43	1103	1014	14	0.888
61	524	438	11	1.236
41	453	333	8	1.619
53	304	210	8	1.954
37	2469	2354	29	4.522
47	3506	3425	51	8.315
71	9670	9588	148	28.795

The step's medians: grokking 931, plateau 704, ricochets 19. Seed 59 was taken. Seed 67 hits two of the three medians exactly — grokking 931 and plateau 704 — but loses on the third: 31 ricochets against the median 19; the criterion adds all three deviations, and by it 59 wins. The counts of ricochets here and below follow the definition in `c4_experiments/task017/plot_run.py` (a threefold rise of the loss above the minimum of the previous twenty epochs, with the rise continuing); the same one is used in drawing the figure as well.

The failures and their cure. Of the 117 runs, 87 grokked straight away and 30 did not. The failures fall into two types:

type of failure	how many	what happens	where it occurs
semi-grokking	26	train accuracy = 1 is reached, validation stops below 0.999	×2–×8, ×11, ×12, ×20
non-convergence	4	train accuracy does not reach 1 within the whole budget	×10, ×11, ×12, ×15 — all on seed 53

The two kinds of failure lie on opposite sides of the optimum. Beyond the upper boundary the network stops converging at all: train accuracy does not reach one, and from the very first epochs training goes into a state it does not get out of within the whole budget — the rate is simply large for this configuration. Beyond the lower one the picture is the reverse, and far more interesting: the network memorizes the training set confidently, at every multiplier by the first or second hundred epochs — but generalization does not come. The threshold of non-convergence depends both on the multiplier and on the seed: in this series all the non-convergence fell on seed 53, the least stable of the nine, and on raising the multiplier further it is to be expected on the others as well.

To the semi-grokkings the levers that raise the variance of the gradient at an unchanged learning rate were applied; to the non-convergence they were not, since there is nothing there to destroy: the network has not memorized.

lever	rescued out of 26	median grokking
batch 512 → 256	18	981
`betas[0]` 0.9 → 0.8	4	2668
both at once	1	1398
did not yield	3	—

The levers were tried in this order, and the order is not a matter of indifference: reducing the batch rescues more often and twice as fast, whereas lowering `betas[0]` on one of the runs (×8, seed 37) switched generalization off — validation accuracy stayed at 0.016 at train accuracy = 1, whereas without the lever the same configuration reached 0.997. Applying both levers together turned out, against expectation, to be weaker than either of them separately: one rescue out of four attempts. On individual runs reducing the batch turns a stuck semi-grokking into the fastest grokking of all those observed: 240 epochs where the step's median is 1513.

Three runs yielded to nothing: ×2 on seeds 59 and 67, ×12 on seed 37.

B.3. Lowering the overall rate under a large multiplier: the full table

A remedy that was tested and does not work. For multipliers beyond the upper boundary (appendix B.2) a remedy suggested itself: lower the overall learning rate of the whole network, which makes the overdrive of one matrix relatively milder. It changes the baseline rather than the method, and therefore called for a series of its own, laid out so that pairs with the same **effective** rate on `W_QKV` could be read against one another. The baseline of §3.1 at weight_decay = 0, a budget of $2 \cdot 10^4$ epochs, stopping 10^3 epochs after grokking — the settings of §4.2 unchanged; only the base learning rate and the `W_QKV` multiplier vary. Nine seeds — the primes from 37 to 71, as throughout the work; 45 runs.

base rate	multiplier	effective on `W_QKV`	seed	memorization	grokking
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	37	111	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	41	94	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	43	95	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	47	99	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	53	107	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	59	108	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	61	96	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	67	114	—
$5 \cdot 10^{-4}$	×1 (control)	$5 \cdot 10^{-4}$	71	97	—
10^{-3}	×20	0.02	37	7597	—
10^{-3}	×20	0.02	41	1685	—
10^{-3}	×20	0.02	43	107	460
10^{-3}	×20	0.02	47	136	4952
10^{-3}	×20	0.02	53	154	7140
10^{-3}	×20	0.02	59	201	6104
10^{-3}	×20	0.02	61	173	7281
10^{-3}	×20	0.02	67	158	1900
10^{-3}	×20	0.02	71	121	3473
$5 \cdot 10^{-4}$	×20	0.01	37	—	—
$5 \cdot 10^{-4}$	×20	0.01	41	428	—
$5 \cdot 10^{-4}$	×20	0.01	43	156	4102
$5 \cdot 10^{-4}$	×20	0.01	47	109	2030
$5 \cdot 10^{-4}$	×20	0.01	53	—	—
$5 \cdot 10^{-4}$	×20	0.01	59	603	—
$5 \cdot 10^{-4}$	×20	0.01	61	189	8555
$5 \cdot 10^{-4}$	×20	0.01	67	187	3135
$5 \cdot 10^{-4}$	×20	0.01	71	1478	—
$5 \cdot 10^{-4}$	×30	0.015	37	313	—
$5 \cdot 10^{-4}$	×30	0.015	41	235	—
$5 \cdot 10^{-4}$	×30	0.015	43	117	9869
$5 \cdot 10^{-4}$	×30	0.015	47	154	—
$5 \cdot 10^{-4}$	×30	0.015	53	2161	—
$5 \cdot 10^{-4}$	×30	0.015	59	158	—
$5 \cdot 10^{-4}$	×30	0.015	61	—	—
$5 \cdot 10^{-4}$	×30	0.015	67	225	—
$5 \cdot 10^{-4}$	×30	0.015	71	656	—
$2.5 \cdot 10^{-4}$	×30	0.0075	37	332	—
$2.5 \cdot 10^{-4}$	×30	0.0075	41	201	—
$2.5 \cdot 10^{-4}$	×30	0.0075	43	173	—
$2.5 \cdot 10^{-4}$	×30	0.0075	47	226	—
$2.5 \cdot 10^{-4}$	×30	0.0075	53	—	—
$2.5 \cdot 10^{-4}$	×30	0.0075	59	313	13363
$2.5 \cdot 10^{-4}$	×30	0.0075	61	—	—
$2.5 \cdot 10^{-4}$	×30	0.0075	67	283	—
$2.5 \cdot 10^{-4}$	×30	0.0075	71	214	17060

A dash in the "memorization" column means that train accuracy did not reach one within the whole budget: the network does not converge at all — the first type of failure (appendix B.2). A dash in the "grokking" column with the memorization column filled in is the second type, semi-grokking.

A summary by arm, in order of the effective rate on `W_QKV`:

base rate	multiplier	effective	memorized	grokked	grokking
$5{\cdot}10^{-4}$	×1 (control)	$5{\cdot}10^{-4}$	9/9	0/9	—
$2.5{\cdot}10^{-4}$	×30	0.0075	7/9	2/9	13363, 17060
$5{\cdot}10^{-4}$	×20	0.01	7/9	4/9	2030–8555
$5{\cdot}10^{-4}$	×30	0.015	8/9	1/9	9869
10^{-3}	×20	0.02	9/9	7/9	460–7281, median 4952

The comparison the series was set up for is between the rows with the same effective rate: 0.01 at ×10 and a base of 10^{-3} (the optimum of the method, 8/9, a median of 962 epochs, appendix B.2) against 0.01 at ×20 and a base of $5{\cdot}10^{-4}$ (4/9, and 2/9 of them without memorization at all). One and the same rate on the overdriven matrix, and an outcome twofold different.

A second pair, visible only across nine seeds: the multiplier ×20 at a base of 10^{-3} and the same multiplier at a base of $5{\cdot}10^{-4}$ — 7/9 against 4/9. The ratio of the rates is the same, the absolute rate is half as large, and the runs with grokking are almost half as many. On three seeds (37, 53, 71) that same arm gave 0/3, and it was over three seeds that the series was read before it was extended: the direction of the conclusion is the same, but "never groks" turned out to be "groks half as often".

The row ×20 at a base of 10^{-3} reproduces the corresponding row of appendix B.2 exactly — 7/9, a median of 4952, a range of 460–7281 — although the runs were made as a separate series; this is at the same time a check that the testbed is deterministic over seeds.

B.4. What follows from this for the method

Failure follows the **ratio** of the rates and not the absolute rate of the overdriven matrix: what breaks the network is not that `W_QKV` learns fast but that it learns much faster than the rest. This also sharpens the method of §4.2 itself: the overdrive works not because it gives the attention matrix some correct absolute rate, but because it shifts the order in which the representations take shape — and too large a shift destroys training regardless of how slowly everything else is going.

B.5. `W_in` overdrive with row norms preserved: the full map

How figure 4.3-a is built. The curves are smoothed by a running maximum over 25 epochs: the runs pass through hundreds of ricochets, and without the smoothing the plot turns into a comb of dips.

The reliability window. All twenty-seven runs in the band from x1.5 to x3 reach generalization; above x3 the share of successes falls, and at a tenfold overdrive nobody groks at all.

The overdrive without normalization. It too brings grokking about, but so late that the gain over no intervention whatever on the same seed comes to within a third of the path, and on a budget that suffices for the normalized runs most seeds do not reach the mark.

Memorization and the plateau. Memorization does not depend on the multiplier at all and is everywhere over by the first hundred epochs; only the length of the plateau changes, and it is the plateau that makes up nearly the whole path.

How the normalization works. Formally it is a row-wise $\ell 2$ normalization of the matrix — the norm is preserved along the dimension over which the summation runs, that is, over the inputs of every output neuron — with a fixed rather than a learned scale multiplier. From the device applied to grokking in Liu et al. (2022b) it differs in level: there the norm of the whole network is held on a sphere of fixed radius, here it is the norm of each neuron separately and only in one matrix. That turns out to be enough. The radius, moreover, is not assigned from outside: the row norm is remembered before the optimizer step and restored after it, so that it stays the one the row received at

initialization. The initialization shared by all the methods (§3.1) is thereby left intact — the network starts from exactly the same point as in the other sections.

The baseline of §3.1 at weight_decay = 0, nine seeds (the primes from 37 to 71), a budget of $3 \cdot 10^4$ epochs, a stop 10^3 epochs after grokking. The marks are medians of the epoch number over those runs that reached them; a row may therefore have a median at 90 % and a dash at 99.9 %. The last column is the median maximum of validation accuracy among the runs that did not reach 99.9 %.

With normalization.

multiplier	grok	10 %	50 %	90 %	99.9 %	max. without grok
×1 (normalization alone)	8/9	7393	9509	11091	17132	0.998
×1.5	9/9	5568	7120	8411	11844	—
×2	9/9	4469	5798	6715	9848	—
×3	**9/9**	2749	4611	5634	**6616**	—
×4	7/9	2321	3712	5120	7891	0.960
×5	7/9	1580	3204	5908	10278	0.966
×6	3/9	1208	3215	6149	5078	0.822
×10	0/9	766	785	1749	—	0.147

The intermediate marks show what the last one alone does not. The start of the climb speeds up with the multiplier monotonically and far beyond the edge of the window: x10 passes 10 % at epoch 766 against epoch 7393 for x1, that is, ten times earlier — and groks not once. The overdrive, in other words, duly speeds up the early growth of generalization; what it does not provide at large values is the ability to see the thing through to the end.

Without normalization.

multiplier	grok	10 %	50 %	90 %	99.9 %	max. without grok
×2	1/9	9073	17698	26746	15857	0.909
×3	2/9	6086	11224	16795	25556	0.983

The same multipliers without the norms preserved pass every mark two to three times later, and reach the last one in one or two runs out of nine. The spread meanwhile widens sharply: at x2 the maxima of the runs that did not grok scatter from 0.372 to 0.988, whereas at x3 they lie within 0.941–0.998. A smaller overdrive without stabilization does not make the result more reliable — it makes it more random.

A separate run of one of the x3 seeds at a budget of 10^5 epochs reaches grokking at epoch 57344 — that is, the matter is not a ceiling but a price: without normalization the path is nearly an order of magnitude longer and only a third shorter than untouched training on the same seed (89089 epochs). The comparison here has to be made per seed: seed 71 is one of the slowest in the baseline, and against the median over nine seeds (53152) that same run would come out not faster but eight percent slower.

For comparison: the attention LR overdrive (§4.2, appendix B.2) at its optimum passes 99.9 % at epoch 931. The method of this section is about eight times faster than untouched training and about seven times slower than the attention overdrive.

B.6. Two methods of demolition at the same thresholds: the full table

The baseline of §3.1 at weight_decay = 0, nine seeds — the primes from 37 to 71, a budget of $3 \cdot 10^4$ epochs, a stop 10^3 epochs after the first crossing of 0.999. Both methods were run with literally identical settings: damage thresholds D1 = 0.25 and D2 = 0.1, an observation window of 20 optimizer steps, and the same triggering conditions of §4.1, set out below. The only difference is the rollback policy, as §4.1 lays it down: after a rollback the escalation meets the next batches, the punch repeats the trial on the same ones (the justification is below and in appendix B.1). 18 runs in all. The marks are the epoch number of the first crossing; a dash means the mark was not reached within the whole budget.

Other tasks. For tasks other than the reference modular arithmetic, thresholds on train loss can be used in the same way instead of thresholds on train accuracy.

The triggering conditions 2.1–2.3 in full. The gate is checked once per epoch, at its boundary, and opens only when all three conditions hold at once. The values below are those at which every measurement in §4.1, appendix B.6 and appendix B.11 was made; in the code they are `train_acc_threshold`, `plateau_window` with `plateau_sig`, and `plateau_delay`.

- 2.1. The last train accuracy value of the epoch is no lower than TA1 = 0.99. After every demolition the threshold must be reached anew.
- 2.2. Train loss has stopped improving: the value of the current epoch is no lower than the value five epochs earlier, both of them rounded to two significant digits. The rounding is essential here — without it the per-epoch noise never lets a plateau be declared at all. The epoch at which the comparison first fails is remembered as the plateau epoch.
- 2.3. At least 50 epochs have passed since the remembered plateau epoch. The plateau counter, like the threshold 2.1, is reset after every demolition, so the minimum interval between two demolitions is 50 epochs plus the time to come out onto a plateau again, and not a fixed period.

The reset happens at the moment the demolition **began**, not when it ended: a demolition takes several epochs, repeated trials included, and the conditions are earned anew from its beginning. The check is carried out without short-circuiting — all three conditions are computed every epoch, even when an earlier one already fails, because condition 2.2 carries state: if it is not asked in time, it will remember as the plateau epoch the one at which it was first looked at, and the demolitions will shift later than the actual plateau.

A fourth condition that §4.1 does not have. In the code the gate also contains a lower bound on the epoch: a demolition is forbidden earlier than epoch 100 from the start of the run (`min_epochs = 100`). It is the only one that is not reset — it works once, and after epoch 100 it stands open to the end of the run. It is not given in §4.1 because on this baseline it is entirely covered by the three listed and is never the binding one: train accuracy reaches 0.99 no earlier than epoch 54 (a median of 62 over nine seeds, see below), and condition 2.3 adds another fifty on top. That is visible in the runs themselves — were the bound binding, on some of the seeds the first demolition would fall exactly on epoch 100, whereas in fact the first demolitions land on epochs 114–168 and the spread between them is set by the plateau and not by the bound. On a task where memorization comes noticeably earlier, the condition may well be needed.

Learning rate escalation (controlled demolition).

seed	10 %	50 %	90 %	99.9 %	maximum	demolitions	of them after grokking	last demolition	accuracy at the end
37	2455	2836	3163	3554	1.000	51	5	4155	0.830
41	2873	3670	3758	3859	1.000	58	8	4523	0.745
43	4990	5102	5142	—	0.997	79	0	16839	0.850
47	4074	4349	4408	4464	1.000	75	12	5384	0.998
53	3014	3605	3769	3921	0.999	54	4	4579	0.622
59	2562	3126	3135	3177	1.000	50	11	4126	0.980
61	2874	3628	3938	3970	1.000	59	8	4714	0.888
67	4022	4547	4557	4620	1.000	76	12	5525	0.805
71	3612	4182	4186	4199	1.000	63	5	4835	0.852
median	**3014**	**3670**	**3938**	**3946**	1.000	59	8	4714	0.850

8 of 9 grokked. The single failure — seed 43 — is not a climb cut off by the budget: accuracy stopped at 0.997, the tail is stable by the least-squares criterion (a slope of −0.007 in a window of 50 epochs and +0.001 in a window of 400), and the demolitions went on to epoch 16839 and gave no result.

Random punch.

seed	10 %	50 %	90 %	99.9 %	maximum	demolitions	last demolition	accuracy at the end
37	750	1019	1209	—	0.967	149	29794	0.630
41	639	906	1133	—	0.981	150	25884	0.849
43	1458	1798	—	—	0.880	164	25474	0.811
47	1215	1409	1575	—	0.953	157	24142	0.839
53	1006	1189	1306	—	0.944	155	29870	0.766
59	1015	1221	1450	—	0.982	159	29732	0.802
61	1154	1465	—	—	0.851	146	18952	0.765
67	1006	1234	1736	—	0.930	160	29755	0.846
71	1062	1411	—	—	0.881	136	26511	0.807
median	**1015**	**1234**	**1378**	**—**	0.944	155	26511	0.807

0 of 9 grokked. Three runs did not even reach 90 %. Memorization (train accuracy = 1) comes equally early with both methods — for the punch a median of epoch 69 with a range of 57–83, for the escalation epoch 62 with a range of 54–86 — because up to the first demolition the runs proceed without intervention: the difference lies entirely in what happens afterwards.

What the medians add. Faster at the entrance does not mean further: for the escalation there is almost no distance between 90 % and 99.9 % (medians of 3938 and 3946 — the remainder is covered in a dozen epochs), whereas the punch spends the whole of the rest of the budget on that stretch. The number of demolitions is the reverse of what one would expect: per run the punch has three times as many (a median of 155 against 59), but only because its runs go the full $3 \cdot 10^4$ epochs while the escalation runs stop soon after grokking; per epoch of training it is the escalation that demolishes more often — once every 84 epochs against once every 194. And here is what the thousand epochs after grokking were left in the runs for: demolitions not switched off in time destroy what was achieved in all eight of the escalation runs that grokked — validation accuracy collapses to a median of 0.172 and a thousand epochs later stands at 0.85 (the worst case is 0.622); this is the price of the switch of §4.1 not being engaged here (appendix B.11).

The stability of the runs that fell short. Of the nine punch runs, five have a stable tail and four an unclear one; not one is rising, that is, the budget cuts off no climb. Unclearness here is not a sign of an insufficient budget but a consequence of the demolitions not ceasing: in a window of 50 epochs the slope is at times markedly positive and in a window of 400 negative (seed 53: +0.15 against −0.06 in units of accuracy per hundred epochs), that is, the curve oscillates about a level rather than rising towards it. The worst by maximum is seed 61 (0.851), and its tail is a stable one.

Calibration. The continuous force of the punch lands within the prescribed range of damage without fail: over all nine seeds **1376 accepted demolitions out of 1376** lie between D2 and D1, and 2193 trials were spent on them. The discrete duration of the escalation lands in the same corridor 313 times out of 565, over 946 trials: on this baseline the damage grows in a jump, and it is easy to overshoot the corridor by the length of a step; the analysis is in appendix B.1. The punch, in other words, calibrates more precisely but gives demolitions that are uniformly crude, while the escalation calibrates worse but every demolition that succeeds repeats the network's own movement — and in sum three times fewer of them are needed.

B.7. The untouched baseline without regularization: the full table

The baseline of §3.1 at weight_decay = 0 and without a single other change — no ricochets, no overdrive, no normalization, no decay mask — nine seeds (the primes from 37 to 71), a budget of $2 \cdot 10^5$ epochs, a stop 10^3 epochs after grokking. This is the quantity by which every estimate of speed-up in §4.2, §4.3, B.2, B.5 and R is divided. The marks are the epoch number of the first crossing.

seed	10 %	50 %	90 %	99.9 %	memorization	plateau
37	7617	13712	19385	31569	86	31483
41	7463	14581	21021	38534	54	38480
59	7642	11584	16446	41058	61	40997
43	12574	22972	31207	48048	56	47992
53	10934	20027	28447	**53152**	68	53084
61	10361	25111	36575	66736	62	66674
47	13417	29476	41543	79118	67	79051
71	17687	39078	57827	89089	73	89016
67	10541	23009	36987	97553	62	97491
median	10541	22972	31207	**53152**	62	53084

9 of 9 grokked. The spread of the grokking epoch is threefold, from 31569 to 97553; not one run exhausted its budget.

Memorization does not depend on the seed, the plateau depends on it entirely. Train accuracy reaches unity between epochs 54 and 86 on all nine seeds, a median of 62 — the same numbers as in B.6, where up to the first demolition the runs coincide with untouched ones. The whole of the variance falls on the plateau: from 31483 epochs to 97491, that is, from three hundred to fifteen hundred memorization durations. The separation of the two processes (§1) is observed in precisely this form: the first takes about sixty epochs regardless of the seed, the second three orders of magnitude more and with a threefold spread.

The order of the seeds along the climb is not preserved. Seed 67 passes 10 % fifth fastest and 90 % seventh, yet finishes last; seed 71 goes last at all the early marks and finishes eighth. The early marks, then, do not predict the late one — the same observation as in B.2 about the attention overdrive across multipliers.

Errorlessness is reached by no run. All nine stop a thousand epochs after the three nines, with a maximum validation accuracy of between 0.9991 and 0.9994, that is, with four or five errors out of 4656, and they do not remove those last errors. In the baseline with regularization the gap between 0.999 and errorlessness comes to sixteen epochs by the median (§4.0); without regularization it exceeds a thousand on all nine seeds. This is one more argument for dating grokking by the three nines: at weight_decay = 0 the fourth nine is not a slightly later moment but a different question.

The run at seeds 0/0, on which the earlier estimates rested, reaches 99.9 % at epoch 74022. Against these nine it takes seventh place out of ten and stands 1.39 times above the median, that is, it belongs to the slow tail of the distribution and not to its middle.

B.8. Slingshot and ricochet: a comparison table

The numbers on which the comparison in section 4.1, “Why this is not a slingshot”, rests; figure 4.1-SR shows the same runs.

sign	slingshot (weight_decay = 0)	ricochet (weight_decay = 0.5)
norm of the last layer’s weights	a staircase: 14 % of the steps decrease (noise on the plateau), net travel 0.86 of the full; between the first and the last event of the stretch ×6.63	an oscillation about a level: 58 % of the steps decrease, net travel 0.16 of the full; between the first and the last event ×1.10
range of the norm over one cycle	0.7 % [0.2–217.9]	5.8 % [2.3–11.7]
angular metric	away from events 86.2°; it does not answer the demolition itself with a spike: a peak of 88.3°, practically never going past the right angle, 79.4° in the following epochs	it does answer: a peak of 102.4° (up to 131.2°), and immediately after a collapse to 41.5°

sign	slingshot (weight_decay = 0)	ricochet (weight_decay = 0.5)
training loss	a spike $2.6 \cdot 10^6$ times above the level before the event	170 times

Both runs are on the baseline of §3.1 at eps = 1e-8, seeds 0/0. The slingshot is a run at weight_decay = 0, recorded step by step up to epoch 10900; the quantities are read off on the stretch of the norm's staircase (epochs 1501–4851, fig. 4.1-SR) over 17 events found by the training loss coming off the floor (`c1_tools/collapse_events.py`, `c4_experiments/task024/metrics.py`). The ricochet is a run at weight_decay = 0.5 over 1100 epochs; the quantities are read off after memorization over 21 events. All the quantities are medians over events. The definitions of the rows, common to both columns, are in `c4_experiments/task023/metrics.py`. *[The ricochet column was computed with the previous detector (a threshold of 50 medians of the loss over the run); the kernel detector finds 29 events instead of 21, and some of the rows change: the collapse of the angle after an event is to 46.7° instead of 41.5°, and the spike of the training loss is 80 times instead of 170. The switch is a separate decision, see task-024.]*

The angular metric by the phases of a slingshot. A slingshot is identified by the weight norm: it grows strongly and for a long time, and after it stabilizes it settles onto a plateau. The angular metric here shows not the demolition but **a change in the regime of movement**. On the plateau it holds close to 90° — a median of 87.4° over the stretch of the staircase — that is, the gradients of neighbouring steps are nearly orthogonal, like two random variables in a many-dimensional space, and the network is simply marking time. As the norm accelerates the metric falls — a median of 75.9° over the stretches of growth, and as far as 40° on the largest step of all: the movement becomes directed. Against a metric that lies near 90° the whole of the rest of the time, this is a noticeable and systematic change — but it is tied to the acceleration of the norm and not to the moment of demolition, which the metric does not answer with a spike.

B.9. The path under regularization: the full table

If one separates the regularization's contribution to the weight update from the gradient's, then the Euclidean distance the network travels under the action of the regularization alone before grokking arrives changes by no more than a factor of two or three, while weight_decay — and with it the epoch at which grokking arrives — changes by two orders of magnitude. Over the upper order and a half of weight_decay, from 1 to 0.056, the median path holds between 171 and 192 — that is, it varies within a tenth while the epoch of grokking grows seventeenfold. The measurements are in appendix B.9. One may conjecture that the network has to do a certain, barely varying amount of work on forming generalizations.

The baseline of §3.1 at eps = 1e-8, seed 37, nine values of weight_decay from 1 to 0.01 in steps of 10^0.25, a budget of $2 \cdot 10^5$ epochs, stopping 10^3 epochs after grokking. The path is the sum of the lengths of the regularization substeps: in decoupled AdamW that substep is `p ← p·(1 − lr·wd)`, so its length is `lr·wd·▫p▫` taken before the optimizer step (`c1_tools/decay_path.py`, measurement `c4_experiments/task027/`). The marks are the number of the epoch of the first crossing.

weight_decay	memorization	val ≥ 10 %	grokking	path to memorization	path to val ≥ 10 %	path to grokking
1	135	168	279	98.7	114.6	174.7
0.5623	122	320	494	63.5	124.3	177.7
0.3162	90	573	858	30.9	127.7	178.6
0.1778	65	929	1501	13.4	121.5	177.2
0.1	66	1583	2497	7.9	120.9	169.5
0.05623	67	2728	4110	4.5	122.8	166.1
0.03162	61	4895	7492	2.3	147.0	265.1
0.01778	108	6884	10734	2.4	199.3	342.3
0.01	61	7572	12160	0.7	156.4	245.4

weight_decay	memorization val ≥ 10 %	grokking	path to memorization	path to val ≥ 10 %	path to grokking
spread			**×132**	**×1.74**	**×2.06**

What changes together with weight_decay and what does not. The epoch at which the first stable signs of generalization appear grows from 168 to 7572, that is, by a factor of 45; the path to it meanwhile changes by a factor of 1.74. The path to complete memorization, on the contrary, follows weight_decay almost proportionally: from 98.7 at weight_decay = 1 down to 0.7 at 0.01, that is, by a factor of 132. What is constant, then, is not every distance, but precisely the distance to generalization.

The reference series on which the observation was first obtained (ten seeds per value, the same set of weight_decay) gives, at seed 37, a path to val ≥ 10 % of between 52.0 and 68.2 (a spread of ×1.31) and to grokking of between 79.6 and 123.1 (×1.55); by the medians of the ten seeds, ×1.39 and ×1.40.

The path to generalization is not strictly constant: it has a flat minimum in the middle of the range, and towards small weight_decay it grows again. The same shape is visible in the reference series as well. The absolute values of the two testbeds are not compared: the reference implementation has a different set of parameters, and its mark for grokking is val ≥ 0.99 against 0.999 here.

The table is built on a single seed. In the reference series the spread between seeds is wider than the spread over weight_decay (single runs give anywhere from 20.8 to 105.8 up to val ≥ 10 %, at medians of 55.3–76.7), so the only thing this table allows one to claim is that the distance changes little, not how exactly constant it is.

Nine seeds. The observation was obtained on seed 37; the series was repeated on all nine seeds of the paper — 81 runs, nine values of weight_decay per seed. Below are the median and the span of the path to grokking over the nine seeds for each value of the regularization.

weight_decay	grokking, median	path to grokking, median	span of the path across seeds
1	273	170.6	141.8–239.5
0.5623	515	189.7	144.7–225.7
0.3162	884	184.3	144.6–220.8
0.1778	1588	192.2	156.4–217.9
0.1	2676	181.1	160.3–217.4
0.05623	4673	182.5	166.1–229.8
0.03162	8606	279.0	201.9–461.5
0.01778	12453	424.5	342.3–488.5
0.01	15375	415.3	245.4–499.1

Within each seed the span of the path over the whole range of weight_decay comes to between ×2.06 and ×3.00 (median ×2.58), whereas the epoch of grokking grows by between ×43.6 and ×69.3 (median ×56). On not one seed did the picture turn over: everywhere the path changes an order of magnitude less than the time does.

The shape of the dependence is the same on all nine seeds, and it is arranged thus: over the upper order and a half of the range (weight_decay from 1 to 0.056) the median path lies between 171 and 192 — a span of a tenth against a seventeenfold growth of the epoch of grokking — while the whole of the growth of the path falls on the three smallest values, where the path increases roughly twofold. This is exactly the qualification for whose sake the series was run on nine seeds: the “constancy of the path” is exact only where the regularization is strong enough to lead the process; at the lower edge it stops being the leading one, and part of the work is by then being done by something else.

B.10. Single and double precision: the full table

The testbed of §3.1 at eps = 10^{-8}, three data and model seeds (0, 37, 53), two regimes of regularization and two precisions; in fp64 the whole model is converted, the optimizer’s moments included.

"Events" is the number of demolitions found by the `collapse_events` detector after memorization: slingshots in the weight_decay = 0 regime, ricochets at weight_decay = 0.5. "Depth" is the median training loss at the events. "Norm" is the norm of the last layer at the end of the run, and its growth from the start. "Angle" is the median angular metric away from the events (no closer than 20 epochs).

regime	precision	seed	events	depth	norm	growth of norm	angle	grokking
wd = 0	fp32	0	25	0.025	217.6	×37.6	87.0°	—
wd = 0	fp32	37	24	0.017	228.2	×39.5	87.1°	—
wd = 0	fp32	53	25	0.018	196.0	×34.0	87.4°	—
wd = 0	**fp64**	0	**0**	—	24.8	×4.29	94.8°	—
wd = 0	**fp64**	37	**0**	—	24.7	×4.26	94.9°	—
wd = 0	**fp64**	53	**0**	—	24.7	×4.29	94.7°	—
wd = 0.5	fp32	0	31	0.410	20.3	×3.50	62.1°	614
wd = 0.5	fp32	37	27	0.404	20.5	×3.55	62.5°	518
wd = 0.5	fp32	53	28	0.463	19.7	×3.41	61.9°	531
wd = 0.5	**fp64**	0	28	0.325	21.5	×3.72	63.8°	597
wd = 0.5	**fp64**	37	27	0.671	20.5	×3.55	63.2°	534
wd = 0.5	**fp64**	53	27	0.237	21.6	×3.75	64.6°	533

The separation is clean: in the "events" column the upper half of the table falls from two and a half dozen to zero on all three seeds, while the lower half does not change at all. There is not one intermediate case.

Within a budget of 5000 epochs not one run at weight_decay = 0 reaches three nines — neither in single precision nor in double; this is to be expected (with no regularization the baseline needs 31–98 thousand epochs, appendix B.7) and means only that the "grokking" column in the upper half of the table is empty for want of budget and not in substance. Long runs, set up separately, answer that question too. Each goes twice as far as the epoch at which its twin in fp32 groks:

seed	fp32 twin	fp64 budget	memorization	grokking	validation at the end	maximum
37	31569	60000	68	—	0.0168	0.021
53	53152	90000	91	—	0.0133	0.017

The maximum in both cases falls on the second epoch of training, that is, it is the noise of the initial initialization and not a climb that was halted. A third seed was not run: in fp64 a step costs 2.9 times as much, and these two runs took eight and twelve hours.

Slingshots in fp32 on the same budget, nine seeds of the baseline. So that the zero in double precision could not be put down to a short run, the same detector was run over nine runs of the untouched baseline at weight_decay = 0 (appendix B.7; single precision, the budget up to grokking). "In the first 5000" are the events that fall within the comparison budget above; "up to grokking" is over the whole run up to the epoch of the first crossing of 0.999.

seed	memorization	first slingshot	in the first 5000	up to grokking	grokking
37	86	184	24	202	31569

seed	memorization	first slingshot	in the first 5000	up to grokking	grokking
41	54	293	23	255	38534
43	56	251	25	314	48048
47	67	205	25	523	79118
53	68	221	25	365	53152
59	61	174	20	262	41058
61	62	273	23	451	66736
67	62	183	17	629	97553
71	73	236	25	618	89089

Medians: the first slingshot at epoch 221, 24 events in the first five thousand epochs, 365 up to grokking. The count in the first 5000 epochs (17–25) agrees with what the three seeds of the wd = 0 fp32 arm give in the table above (24–25) — that is, in the frequency of slingshots that arm stands out in no way among the nine runs of the baseline. The epoch of the first slingshot does not depend on how late the run groks: seed 67 takes its first slingshot earliest of all (183) and groks latest of all (97553).

B.11. Switching the demolitions off at the mark: the full table

A series belonging to §4.1. The baseline of §3.1 at weight_decay = 0, nine seeds, thresholds D1 = 0.25 and D2 = 0.1, a relaxation window of 20 steps, the gates of §4.1 (appendix B.6) — the settings of figure 4.1-c with no changes; the only difference is that the demolitions cease as soon as validation accuracy has crossed 0.999. A budget of $3 \cdot 10^4$ epochs, stopping $5 \cdot 10^3$ epochs after grokking. “Dip” is the worst validation accuracy after grokking, “above threshold” the share of the epochs after grokking at which it is no lower than 0.999, “final” its value at the end of the run. The “B.6” column gives alongside it the same quantity for the same method without the switch.

seed	grokking	B.6	demolitions	B.6	dip	above thresh-old	final	B.6
37	3554	3554	46	51	0.989	100 %	1.0000	0.830
41	3859	3859	51	58	0.980	45 %	0.9966	0.745
43	—	—	79	79	—	—	0.8503	0.850
47	4464	4464	63	75	0.981	76 %	0.9996	0.998
53	3921	3921	50	54	0.992	99 %	1.0000	0.622
59	3177	3177	39	50	0.898	89 %	1.0000	0.980
61	3970	3970	51	59	0.977	90 %	0.9976	0.888
67	4620	4620	64	76	0.967	49 %	0.9985	0.805
71	4199	4199	58	63	0.992	96 %	1.0000	0.852
median	**3946**	3946	**51**	59	0.981	90 %	**0.9998**	0.850

The epochs of grokking agree with appendix B.6 **to the unit** on all eight seeds that grokked, and the failure there is the same one (seed 43, the same 79 demolitions). This is precisely the check that before the mark the switch changes nothing: nor can it, since it looks at a quantity that does not exist before grokking. The number of demolitions falls by exactly as many as fell in the time after grokking by table B.6 (5, 0, 12, 4, 11, 8, 12, 5 — and 7 instead of 8 on seed 41, where one demolition began before the mark and ended after it). The final figure: 0.9966–1.0 against 0.622–0.998, the worst dip after grokking 0.90–0.99 against 0.14–0.18.

From this follows a practical conclusion for the method of §4.1: the demolitions are needed for the transition and are harmful after it, and the measure that takes account of this is a switch, not a weakening of the demolitions.

B.12. The FFN activation: ReLU against GELU

The ablation promised in §3.1. The activation is not specified in the setting of Power et al.; here ReLU is taken, following their published implementation and Nanda et al. (2023), and the question is what a replacement changes. The runs are the baseline of §3.1 with not one other change (weight_decay = 1, a budget of $2{\cdot}10^4$ epochs, stopping 10^3 epochs after grokking), nine seeds, differing in exactly one line of the model: the activation slot of the FFN block.

The replacement changes nothing essential. GELU groks in all nine runs with a median of 230 epochs (an interquartile span of 208–256) against 273 epochs (241–359) for ReLU on the same settings — that is, a sixth part faster by the median, and with the spans overlapping. Memorization comes by epoch 170 by the median (127–193) against a range of 73–197 for the whole ReLU series. The one difference worth mentioning is the tail: with GELU one run out of the nine reached three nines only at epoch 515 and did not reach flawlessness at all, whereas with ReLU the slowest run came in within 359.

The conclusion for the paper is a narrow one and exactly the one needed: the choice of activation on this testbed is not the lever by which anything in sections 4.1–4.3 is explained, and inheriting ReLU from the published implementation of Power et al. decides nothing on the result's behalf. This agrees with Manir and Rupa (2026), where ReLU and GELU grok at almost the same weight norm and only the speed of the descent to it differs; the size of the difference there, however, is much greater — a different architecture and a different optimizer.

C. Reproducibility details (hyperparameters, configurations, hardware, seeds)

Seeds. Every measurement was made on nine arbitrarily chosen and always identical seeds — the primes 37, 41, 43, 47, 53, 59, 61, 67, 71. The seed fixes both the split of the data and the initialization of the model, so a difference between two runs on one seed is a difference of the intervention and not of the sample. The spread across seeds is as a rule wide, and wide quantitatively: with no intervention at all, memorization sets in at epoch 57–83, and grokking anywhere between epoch 30757 and epoch 103491 — that is, three times later for the worst seed than for the best. This is why a claim tested on one seed counts as untested here.

Units. At a batch of 512 the training sample gives ten optimizer steps per epoch; all the marks and budgets below are in epochs.

Reproducibility. A run repeated with the same seed coincides with the original bit for bit — both when it is repeated from the start and when it is resumed from a checkpoint. All the branching experiments rest on this: a branch and its parent differ only by the intervention. On different hardware a bitwise match should not be expected — the numbers will drift quantitatively, but not qualitatively.

Medians. Wherever a single number is given over nine seeds, it is the median and not the mean, and the range stands beside it: the distributions are skewed, and nine points are too few for the mean not to depend on a single outlying run.

Volume. The measurements of the paper rest on several hundred runs: the hierarchy of the localization of weight_decay alone is 261 runs (appendix R), the map of overdrive multipliers 117 (appendix B.2), and the series of branching experiments some three hundred more.

The final normalization. In Vaswani there is no final normalization before the head, whereas on this testbed there is one — a divergence that arose historically. It can be switched off with the parameter `--transformer.final_norm false`, but all the measurements of the paper were taken with it in place.

The value of eps. The optimizer's eps is not stated in the text of the original, but in the published implementation of Power et al. it equals 10^{-8}; that is the value used here as well. The choice is not a cosmetic one: it is through this term that the slingshot mechanism works, the one the comparison in §4.1 is made against, and in the related works on slingshots it is manipulated explicitly. The method introduced here, by all appearances, does not itself rest on it — see the ablation of §8.2.

D. Details of the data and the preprocessing

The qualifier of the original on the batch size. I do not reproduce the original's qualifier "or half the training sample, whichever is smaller": at the base data fraction of 50 % (4656 equations) it does not come into play in any case, and its absence makes it possible, on the same testbed, to run the whole dataset as a single batch as well.

F. A detailed breakdown of the methods: regularization

(one selective verbatim quotation per method; the brief summary table is in §2)

1. **Weight regularization, ℓ2 (weight_decay).** The original trigger: the penalty on the ℓ2 norm of the parameters is switched on after the train loss reaches zero and pulls the weights towards a low-norm generalizing solution; it raises data efficiency more strongly than any other intervention. `"adding weight decay has a very large effect on data efficiency, more than halving the"` (2201.02177, line 220) — *"adding weight regularization has a very large effect on data efficiency, more than halving [the required number of examples]"*.
2. **Dropout.** The stochastic zeroing of activations (in the decoder) acts as an implicit regularization that lowers the effective dimensionality of the solution; at a noticeable dropout rate grokking disappears altogether. `"With a significant dropout rate, the generalization time can be brought down to under 103 steps and the grokking phenomenon vanishes completely"` (2205.10343, line 811) — *"with a significant dropout rate the generalization time falls to under 10^3 steps, and the phenomenon of grokking vanishes completely"*.
3. **A penalty for symmetry/commutativity (abelian degrok → symmetry-prompting).** A task-specific loss term penalizing the violation of a known symmetry (the commutativity `a+b = b+a`); it injects the structure of the task directly and accelerates training. `"we further add the logits level MSE loss of training samples "a + b =" and "b + a =" as regularizer"` (2311.06597, line 482) — *"we additionally add an MSE loss at the level of the logits for the examples 'a + b =' and 'b + a =' as a regularizer"*.
4. **ℓ∞-norm regularization.** A penalty on the largest component of the weights by absolute value, motivated by an analysis of the max-margin solution of modular addition; it moves the network into the rich regime (feature learning). `"This connection motivates us to use the explicit ℓ∞-norm regularization"` (2407.12332, line 283) — *"this connection motivates us to use an explicit ℓ∞ regularization"*.
5. **Other norms: ℓ1 / nuclear (ℓ1 / nuclear norm).** Replacing ℓ2 with a sparsifying (ℓ1) or a low-rank (nuclear) norm; a small but non-zero regularization of this kind is enough for grokking — that is, ℓ2 is not privileged. `"gradient descent with a small but non-zero regularization of P (e.g., ℓ1 or nuclear norm regularization) results in grokking"` (2506.05718, line 56) — *"gradient descent with a small but non-zero regularization of P (for example, of the ℓ1 or the nuclear norm) leads to grokking"*.
6. **Logit-norm regularization (logit-norm).** What is penalized is the norm of the logits themselves rather than that of the weights; it counteracts the unbounded growth of the logits (Softmax Collapse) in a targeted way and yields grokking without the usual weight regularization. `"leads to grokking without additional regularization"` (2501.04697, line 1229) — *"…leads to grokking without additional regularization"*.
7. **Flatness regularization (flatness / relative flatness).** What is regularized is the geometry of the minimum (its flatness) rather than the norm of the parameters; grokking-like behaviour is induced by directly controlling the flatness. `"grokking-like behavior can be actively induced by manipulating flatness"` (2509.17738, line 182) — *"grokking-like behaviour can be actively induced by manipulating the flatness [of the minimum]"*.
8. **Geometric priors (entropy / Lipschitz / the nuclear norm).** Priors on a "simple" geometry of the representations (low rank, a bounded Lipschitz constant, low entropy) shorten

the time to generalization; what is regularized is the structure of the representation, not the norm of the weights. `"Geometry-promoting priors can also reduce time-to-generalization relative to the baseline"` (2603.01968, line 813) — *"geometry-promoting priors also reduce the time to generalization relative to the baseline"*.

9. **The spectral-entropy regularizer (spectral-entropy, DeMoss).** A penalty on the spectral entropy of the weights — in essence on their intrinsic dimension — pushing towards a low-complexity representation; it produces a sharp transition in complexity (a rise during memorization → a fall at generalization). `"we introduce a regularization method based on spectral entropy that encourages networks toward low-complexity representations by penalizing their intrinsic dimension"` (2412.09810, line 58) — *"we introduce a method of regularization based on spectral entropy, which pushes networks towards low-complexity representations by penalizing their intrinsic dimension"*.

G. A detailed breakdown of the methods: the optimizer and gradient transformation

(the same group as in §2; by their nature these methods are implicit regularizers / shapers of the gradient)

1. **An adaptive optimizer / the slingshot (Adam/AdamW).** Without explicit regularization, grokking almost exclusively coincides with the onset of the adaptive optimizer's "slingshots"; a smaller ε shifts grokking earlier. `"Grokking as reported in [17] almost exclusively happens at the onset of Slingshots"` (2206.04817, line 73) — *"grokking, as reported in [17], almost exclusively arises at the moment the slingshots begin"*.
2. **Momentum.** Momentum reduces both the time to grokking and the critical amount of data α. `"Momentum appears to play a major role both in reducing grokking time and α"` (2301.02679, line 1102) — *"momentum, apparently, plays a major role in reducing both the grokking time and α"*.
3. **Tuning lr and the product lr·λ.** From the phase diagram of the hyperparameters, generalization can be "de-delayed"; formally the delay is inversely proportional to the product of the learning rate and the weight regularization. `"show how we can "de-delay" generalization"` (2205.10343, line 118) — *"we show how generalization can be 'de-delayed' "*.
4. **Grokfast (the amplification of slow gradients).** A low-frequency filter amplifies the slow component of the gradient, the one responsible for generalization, and accelerates grokking. `"amplifying this low-frequency component of G(ω) accelerates the speed of generalization under the grokking phenomenon"` (2405.20233, line 186) — *"amplifying this low-frequency component of G(ω) accelerates the speed of generalization under the phenomenon of grokking"*.
5. **NeuralGrok (a learned gradient transformation).** A neural network learns the optimal transformation of the gradient, one that accelerates generalization; along the way the normalization of the gradient plays the role of a regularization. `"we propose NEURALGROK, a novel gradient-based approach"` (2504.17243, line 57) — *"we propose NeuralGrok, a new gradient-based approach"*.
6. **Egalitarian GD / column normalization.** The equalization of the singular values of the gradient matrices (in a simplified form, an L2 normalization of the columns); a simple method of accelerated grokking. `"Egalitarian Gradient Descent (EGD). We propose a novel and simple method for accelerated"` (2510.04930, line 93) — *"egalitarian gradient descent (EGD). We propose a new and simple method of accelerated [grokking]"*.
7. **Grad / OrthoGrad (the orthogonal gradient).** The component along the weights (NLM), which drives the scale of the logits upwards, is removed from the gradient; this does away with the phase of initial overfitting. `"▫Grad optimizers lead to generalization without a phase of initial overfitting"`

(2501.04697, line 646) — *" Grad optimizers lead to generalization without a phase of initial overfitting"*.

8. **Preconditioned GD (Gauss–Newton / Levenberg–Marquardt).** Preconditioning removes the spectral bias and passes through the lazy regime uniformly, eliminating the characteristic delay in generalization. `"using PGD to explore the lazy regime uniformly, without spectral bias, thereby eliminating the characteristic delay in generalization"` (2601.03162, line 131) — *"…using PGD to explore the lazy regime uniformly, without spectral bias, thereby eliminating the characteristic delay in generalization"*.

H. A detailed breakdown of the methods: the numerical stability of softmax

(all three are from Prieto et al., 2025 (2501.04697); they aim at one and the same cause — Softmax Collapse — but at different points)

1. **StableMax / Taylor-Softmax (a stable replacement for softmax).** The exponential of the softmax is replaced by a function that does not saturate at large logits (StableMax — a piecewise-rational form; Taylor-Softmax — an expansion of the exponential in a Taylor series). This removes the absorption error in the softmax and yields grokking without regularization. `"that StableMax leads to grokking in commonly studied settings without regularization"` (2501.04697, line 451) — *"…that StableMax leads to grokking in the commonly studied settings without regularization"*.
2. **Raising the FP precision to float64.** The most direct device against Softmax Collapse is to increase the bit width of the arithmetic: with float64 the threshold of the absorption error is many times higher, the collapse sets in later, and the model has time to pick up test accuracy. It treats the symptom (a numerical threshold) rather than the cause (the unbounded growth of the logits). `"Increasing floating point precision. The simplest way to avoid SC is to extend the FP precision"` (2501.04697, line 379) — *"increasing floating point precision. The simplest way to avoid SC is to extend the FP bit width"*.
3. **Label smoothing.** The target labels are softened (muted values instead of hard 0/1), so the model has no need to inflate the logits without limit for the sake of a confident prediction; this prevents SC and yields partial generalization without regularization. `"label smoothing does prevent SC and lead to some"` (2501.04697, line 1369) — *"label smoothing does indeed prevent SC and leads to some [generalization]"*.

A note. Prieto et al. have other numerical mitigations of the same collapse as well (their own implementation of `log_softmax` outside the CUDA kernels, a reduction of the input dimensionality, a smaller ε in Adam, regularization of the logit norm). Regularization of the logit norm is assigned to the "Regularization" block (§2, F-6); the rest are variants of the same numerical theme, and on a detailed re-check they could be set out as separate rows.

I. A detailed breakdown of the methods: weight initialization

(the whole group requires a NON-STANDARD initialization — that is the common lever; they differ in WHAT exactly they change in the initialization)

1. **A large initialization scale (Omnigrok).** The initial weights are multiplied by a constant α>1 (relative to Kaiming): the network starts in the "lazy" (linearized) regime and is long in passing to feature learning — hence the delay; it works even on non-algorithmic data (MNIST). `"we increase the scale of the weight initialization distribution (by multiplying the initial weights, sampled with Kaiming uniform initialization, by a constant α > 1)"` (2210.01117, line 332) — *"we increase the scale of the weight initialization distribution (multiplying the initial weights, drawn by Kaiming-uniform, by a constant α > 1)"*.
2. **A small initialization scale.** The reverse lever: a very small initialization, on the contrary, softens or removes grokking, bringing the network closer to direct feature learning. `"using a very small weight initialization can`

`substantially mitigate the grokking effect"` (2407.12332, line 653) — *"using a very small weight initialization can substantially mitigate the grokking effect"*.

3. **Constraining the weight norm to a sphere (constrained / spherical, Goldilocks).** The weight norm is fixed on a sphere of the required radius (constant weight norm): instead of the dynamics of the norm under weight decay, the geometrically "correct" radius is prescribed, the one at which generalization is attainable. `"grokking by constraining the model on a small weight norm sphere"` (2210.01117, line 395) — *"…grokking by constraining the model on a sphere of small weight norm"*.
4. **A Fourier initialization of the embeddings.** The first dimensions of the embedding matrix are set deterministically to cos/sin values at the key frequencies — the model is given an explicit periodic prior from the very start (after the manner of the mechanistic analysis of Nanda et al.). `"the first 10 dimensions of the token embedding matrix are deterministically initialized with cosine and sine values at five key frequencies"` (2603.05228, line 375) — *"the first 10 dimensions of the token embedding matrix are deterministically initialized with cosine and sine values at five key frequencies"*.
5. **A zero initialization of the output layer (zero-init).** The top (output) layer is initialized with zero — this accelerates feature learning compared with an ordinary initialization. (Zeroing the last layer had been used earlier by Clauw et al., 2024, but only as a device for stabilization under a large initialization.) `"zero-init accelerates the feature learning process compared to normal initialization"` (2509.21519, line 319) — *"zero-init accelerates the feature learning process compared with an ordinary initialization"*.
6. **A shift of the mean of the initialization distribution.** The centre of the distribution of the initial weights is shifted away from zero; the size of the shift governs the delay — under an initialization centred away from zero the delay of grokking is markedly stronger. `"the resulting delay in grokking is far more pronounced than in the case of an initialization"` (2507.11645, line 529) — *"the resulting delay in grokking is far more pronounced than in the case of an initialization [centred at zero]"*.
7. **A biased (anisotropic) initialization.** The initial weights are deliberately biased against a particular input direction; in linear regression such a non-standard initialization was necessary in order to observe grokking at all. `"This unusual alteration to the initial weights was required for a clear demonstration of grokking"` (2310.17247, line 318) — *"this unusual alteration to the initial weights was required for a clear demonstration of grokking"*.

J. A detailed breakdown of the methods: transfer and warm start

(the lever is the reuse of already "grokked" knowledge or structure from another model or another phase; three of the four set the starting point of the weights, while KD transfers the knowledge through soft targets)

1. **Pre-grokked frozen modules (warm start).** A module (an embedding or a transformer) that has already grokked on a related task is frozen and put in place of the random initialization — this accelerates the grokking of a new, related task (especially of difficult asymmetric expressions). `"pre-grokked embedding with addition accelerates grokking in 2a ± b, 2a ± 3b, and pre-grokked Transformer with multiplication does in ab ± b"` (2402.16726, line 649) — *"a pre-grokked embedding for addition accelerates grokking in 2a ± b, 2a ± 3b, and a pre-grokked Transformer for multiplication does so in ab ± b"*.
2. **Transfer of the embeddings from a weak model (GrokTransfer).** The already learned input embedding is extracted from a weaker model and used (through a linear map) to initialize the embedding of the strong target model — the delay is removed. `"the learned input embedding from this weaker model is extracted and used to initialize the embedding in the target, stronger`

model" (2504.13292, line 60) — *"the learned input embedding from this weaker model is extracted and used to initialize the embedding of the target, stronger model"*.

3. **Distillation from a teacher that has grokked (KD).** A teacher model that has already grokked on a distribution p1 is distilled into a student — this induces and accelerates grokking on a different distribution p2, including below the critical data threshold. The only one in the block that does not touch the initialization: the knowledge is transferred through soft targets. "Knowledge Distillation (KD) from a model that has already grokked on a distribution (p1) can induce and accelerate grokking on a different distribution (p2)" (2511.04760, line 58) — *"knowledge distillation (KD) from a model that has already grokked on a distribution p1 can induce and accelerate grokking on a different distribution p2"*.
4. **Grokked lottery tickets / edge-popup pruning.** A "winning ticket" subnetwork found in the generalizing phase (a grokked ticket) is taken out of the network; training such a subnetwork — or even merely cutting it out with a mask, without changing the weights — sharply reduces delayed generalization. "utilizing lottery tickets obtained during the generalizing phase (termed grokked tickets) significantly reduces delayed generalization across various tasks" (2310.19470, line 58) — *"utilizing lottery tickets obtained during the generalizing phase (termed grokked tickets) significantly reduces delayed generalization across various tasks"*.

K. Properties of the task and the data that affect grokking (NOT tools)

(taken out of the list of methods: these are changes to the task or the dataset itself, not training-side tools — see §2. Kept for completeness as a reference on the dataset-side factors.)

1. **More training data / the fraction of the sample.** The volume (or the fraction) of the training data sets how much optimization is needed before generalization: the less data, the longer the delay (and with a sufficient volume grokking arrives with no regularization at all). "We show empirically that the amount of optimization required for generalization quickly increases as the dataset size decreases" (2201.02177, line 117) — *"we show empirically that the amount of optimization required for generalization grows quickly as the size of the dataset decreases"*.
2. **Reducing the size of the sample (strengthening grokking).** The other side of the same coin: by deliberately reducing the sample (from 60k to 1k, say), grokking is made pronounced even on non-algorithmic data. "we reduce the size of the training set from 60k to 1k samples (by taking a random subset)" (2210.01117, line 331) — *"we reduce the size of the training set from 60k to 1k examples (by taking a random subset)"*.
3. **The composition of the data: the ratio φ of inferred to atomic facts.** Not the size but the *structure* of the data: the fraction of inferred facts relative to atomic ones (φ) sets the speed of grokking directly — a large φ pulls generalization up almost to the pace at which the model fits the training data. "A very large ratio can push generalization to improve at a similar pace as the model fits the training data" (2405.15071, line 228) — *"a very large ratio can pull generalization up to the pace at which the model fits the training data"*.
4. **Data selection by coherence.** Grokking is strengthened by the *choice* of the training examples alone, with every other hyperparameter held fixed — that is, the lever lies in the set itself and not in the optimization. "grokking can be amplified solely through data selection, with any other hyperparameter fixed" (2506.05718, line 70) — *"grokking can be amplified purely through the selection of the data, with any other hyperparameter fixed"*.
5. **Adding spurious dimensions (concealment).** Superfluous dimensions carrying spurious information are added to the data — this induces grokking even on algorithmic tasks (and outside neural networks). "a mechanism by which to induce grokking on algorithmic datasets via the addition of

`dimensions containing spurious information"` (2310.17247, line 68) — *"a mechanism by which grokking can be induced on algorithmic datasets through the addition of dimensions containing spurious information"*.

6. **Removing or introducing a symmetry of the data.** Controlling the symmetries of the training set changes generalization; the sharpest formulation of it is this: breaking the symmetry in the data is necessary in order to recover generalization (it was first noted by Salah et al., 2024, 2411.05353; the clearest statement of all is in Tomàs et al., 2026). `"breaking symmetry in the training data is needed to recover generalization"` (2604.00316, line 182) — *"breaking the symmetry in the training data is needed in order to recover generalization"*.
7. **Constructing the dataset (grokking on a ResNet-18).** A specially constructed dataset brings grokking about even in a convolutional network (ResNet-18), which is to say that the distribution of the data is a lever in its own right, independent of the architecture. `"we construct a dataset that ultimately induces grokking in a ResNet-18"` (2504.03162, line 932) — *"we construct a dataset that ultimately induces grokking in a ResNet-18"*.
8. **Data augmentation.** Real sparse knowledge graphs are supplemented with synthetic data, raising φ (the ratio of inferred to atomic facts) above the threshold required for grokking; it is notable that even factually incorrect synthetic data strengthens the reasoning circuits. `"augmenting existing knowledge graphs with carefully designed synthetic data to raise the ratio ϕr of inferred facts to atomic facts above the threshold required for grokking"` (2504.20752, line 60) — *"supplementing existing knowledge graphs with carefully designed synthetic data so as to raise the ratio φr of inferred facts to atomic facts above the threshold required for grokking"*.

L. A catalogue of the methods for producing grokking: the full table

The classification and the representatives of the groups are in §2; here is the full list, with an indication of whether a device requires a special initialization.

The catalogue includes only **training-side tools** — devices applicable to a fixed task; changes to the dataset and the task themselves are set apart in a separate subsection below and in appendix K. "First mention" is the earliest work known to me that introduces the device as a lever, in the chronological order of the arXiv ids; a separate column says whether the device requires a non-standard weight initialization.

Regularization (at the level of the loss function)

Method	First mention	Special initialization	Essence
Weight regularization, ℓ2 (weight_decay)	Power et al., 2022 (2201.02177)	No	The classical trigger; raises data-efficiency more than any other
Dropout	Liu et al., 2022a (2205.10343)	No	Sharply reduces the time to generalization, to the point where grokking disappears
A penalty for breaking symmetry/commutativity (abelian degrok, symmetry-prompting)	Tan et al., 2023 (2311.06597)	No	Later developed as symmetry-prompting (Hwang et al., 2026)
Regularization of the ℓ∞ norm (ℓ∞-norm)	Mohamadi et al., 2024 (2407.12332)	No	Brings about the transition into the rich regime

Method	First mention	Special initialization	Essence
Other norms: ℓ1 / nuclear (nuclear norm)	Tan et al., 2023 (2311.06597)	No	The nuclear norm — Notsawo et al., 2025 (2506.05718)
Regularization of the logit norm (logit-norm)	Prieto et al., 2025 (2501.04697)	No	Grokking with no additional regularization
Regularization of flatness (flatness / relative flatness)	Han et al., 2025 (2509.17738)	No	Grokking is induced by controlling the flatness of the minimum
Geometric priors (entropy/Lipschitz/nuclear norm)	Hwang et al., 2026 (2603.01968)	No	Low-rank priors accelerate generalization
A spectral-entropy regularizer on the weights (spectral-entropy)	DeMoss et al., 2024 (2412.09810)	No	A penalty on the intrinsic dimension of the weights; an interventional framework — Truong et al., 2026 (2604.13123)
Alignment of the Jacobians with the training data (GrokAlign)	Walker et al., 2025 (2506.12284)	No	Cosine alignment of the network's Jacobian with the training data; ties grokking to robustness against perturbations of the input
Geometric regularization of the dimension (GeomDR)	Kazanskii, 2026 (2607.11666)	No	A spectral penalty on the effective dimension of the hidden representations; the collapse of the dimension precedes grokking
Suppression of the radial component (radial suppression)	Tiwari et al., 2026 (2606.32000)	No	An anisotropic penalty; a 6.3-fold speed-up over a strong weight_decay is claimed
A penalty on the logits (logit regularization)	Beck et al., 2026 (2602.12039)	No	An analysis of the implicit bias of such a penalty; adjoins the logit-norm above

The optimizer and the gradient transformation

Method	First mention	Special initialization	Essence
An adaptive optimizer / slingshot (Adam/AdamW, tuning of ε)	Thilak et al., 2022 (2206.04817)	No	Grokking with no explicit regularization; a small ε shifts it earlier

Method	First mention	Special initialization	Essence
Momentum	Gromov, 2023 (2301.02679)	No	Lowers the time to grokking and the critical volume of data
Tuning of lr and of the product lr·λ	Liu et al., 2022a (2205.10343)	No	Formalized in the delay law (Truong et al., 2026a, 2603.13331)
Grokfast (amplification of slow gradients)	Lee et al., 2024 (2405.20233)	No	A low-frequency filter on the gradient
NeuralGrok (a learned gradient transformation)	Zhou et al., 2025 (2504.17243)	No	Plus normalization of the gradient as a regularization
Egalitarian GD / column normalization	Saheb Pasand et al., 2026 (2510.04930)	No	Equalization of the singular values of the gradient
Grad / OrthoGrad (an orthogonal gradient, removal of NLM)	Prieto et al., 2025 (2501.04697)	No	Removes the phase of initial overfitting
Preconditioned GD (Gauss–Newton / Levenberg–Marquardt)	Jiang et al., 2026 (2601.03162)	No	Removes the spectral bias and the delay
Muon (an orthogonalized update)	Tveit et al., 2025 (2504.16041)	No	Shortens the mean epoch of the transition; the mechanism is not measured in the paper, and the runs of different tasks are merged into a single sample
CvAdamW (a dynamic weight_decay driven by the heat capacity of attention)	Pandey, 2026 (2607.20552)	No	Scales the regularization by the variance of the attention logits, whose peak precedes the transition

Numerical stability of softmax

Method	First mention	Special initialization	Essence
StableMax / Taylor-Softmax (a stable softmax)	Prieto et al., 2025 (2501.04697)	No	Prevents Softmax Collapse; grokking with no regularization
Raising the FP precision to float64	Prieto et al., 2025 (2501.04697)	No	Postpones the collapse (treats the symptom, not the cause)
Label smoothing	Prieto et al., 2025 (2501.04697)	No	Partial generalization with no regularization

Weight initialization (requires a special initialization)

Method	First mention	Special initialization	Essence
A large initial norm with a small weight_decay	Lyu et al., 2024 (2311.18817)	Yes	It is proved that the coupling of the early and the late implicit biases gives rise to grokking
A large initialization scale (Omnigrok)	Liu et al., 2022b (2210.01117)	Yes	$\times\alpha$ on the weights; grokking even on non-algorithmic data (MNIST)
A small initialization scale	Mohamadi et al., 2024 (2407.12332)	Yes	Reducing the scale softens or removes the delay
Constraining the weight norm to a sphere (Goldilocks / spherical)	Liu et al., 2022b (2210.01117)	Yes	Fixing the L2 norm of the weights at the required radius
Fourier initialization of the embeddings (cos/sin)	Yıldırım, 2026 (2603.05228)	Yes	A deterministic periodic prior
Zero initialization of the output layer (zero-init)	Tian et al., 2025 (2509.21519)	Yes	Accelerates feature learning; earlier in Clauw, 2024 (2408.08944) — as a device for stabilization
A shift of the mean of the initialization distribution	Salah et al., 2025 (2507.11645)	Yes	Shifting the centre of the initialization changes the delay
A biased (anisotropic) initialization	Miller et al., 2024 (2310.17247)	Yes	Needed for explicit grokking in linear regression

Transfer and warm start

Method	First mention	Special initialization	Essence
Pre-grokked frozen modules (warm start)	Furuta et al., 2024 (2402.16726)	Yes	Frozen weights from a related task accelerate grokking
Transfer of embeddings from a weaker model (GrokTransfer)	Xu et al., 2025 (2504.13292)	Yes	Initialization of the embedding by a linear map from a weaker model
Distillation from a teacher that has grokked (KD)	Singh et al., 2025 (2511.04760)	No	Induces grokking below the critical data threshold
Grokked lottery tickets / edge-popup pruning	Minegishi et al., 2025 (2310.19470)	Yes	A subnetwork from the generalizing phase removes the delay

Changes to the task and the data — NOT counted among the methods

Changes to the dataset or the task itself (the size and the fraction of the sample, the composition φ, data selection, symmetry, spurious dimensions, construction of the dataset, transfer to real data, augmentation) are **not included** in the table of methods: these are not training-side tools but a change of the task itself — grokking is thereby "carried over" rather than won by a new mechanism. Works

that did apply a genuine tool remain in the table through that tool, in the other blocks (Power → weight_decay; Omnigrok → initialization; Notsawo → regularization; Miller → initialization; Salah → control of the activation). Purely dataset-side works (Wang — φ; Gu — construction of the dataset; Abramov — augmentation) are not counted among the methods. The dataset-side factors are documented separately as properties of the task (the Appendix, section K), not as tools.

Architecture and representation

Method	First mention	Special initialization	Essence
More width / a larger model	Huang et al., 2024 (2402.15175)	No	Large models grok on less data
More depth (deep grokking)	Fan et al., 2024 (2405.19454)	Conditionally	Groks in combination with a large initialization
Manual sparse partitioning of the parameters (MoE-like)	Huang et al., 2024 (2402.15175)	No	Accelerates the emergence of modular addition
Control of the nonlinearity of the activation	Salah et al., 2024 (2411.05353)	No	The shape of the activation governs the presence and the delay of grokking
A bounded spherical topology / bounded unembedding	Yıldırım, 2026 (2603.05228)	No	L2 normalization of the residual stream removes the memorization phase
Uniform Attention Ablation (CBOW)	Yıldırım, 2026 (2603.05228)	No	Replacing attention with uniform summation bypasses the delay
Fourier encoding of the inputs	Xu et al., 2025 (2504.13292)	No	Memorization and generalization at the same time
The scale of the output / the “laziness” parameter α	Kumar et al., 2023 (2310.06110)	No	Governs the separation of the lazy→rich timescales

Dynamics, noise and the training regime

Method	First mention	Special initialization	Essence
Vanilla GD + MSE with no regularization	Gromov, 2023 (2301.02679)	No	Grokking on modular arithmetic without any regularization at all
Noise injection (gradient / input / transverse σ, the temperature η/B)	Power et al., 2022 (2201.02177)	No	Developed further: Beck et al., 2024 (σ, 2410.04489); Ersoy and Wiesner, 2026 (η/B, 2606.17120)
The edge of linear separability (tuning λ=d/N→½)	Beck et al., 2024 (2410.04489)	No	Grokking at the critical point of separability with no regularization
Extended training	Power et al., 2022 (2201.02177)	No	A necessary condition — training far past overfitting

M. Addition and division: why the measurements use division

Addition and division. Of the operations on which grokking is observed, addition `x1 + x2 (mod p) = y` is the one whose solution has been taken apart down to the mechanism: Nanda et al. (2023) showed that the trained transformer implements a Fourier scheme (the "clock") — the embeddings decompose over a few key frequencies, adding numbers becomes adding angles, and the result is read off by trigonometric identities. But addition is also the simplest of the operations that grok: its equation table is so regular that the structure is plain to the naked eye. The division table can be read too — regularity is discernible in it. The regularity itself, however, is incomparably more intricate, and I do not expect mechanistic interpretability to decode it in full any time soon. And yet it groks no harder: division usually needs neither a larger network nor more steps than addition. Division is therefore what the measurements of this paper use.

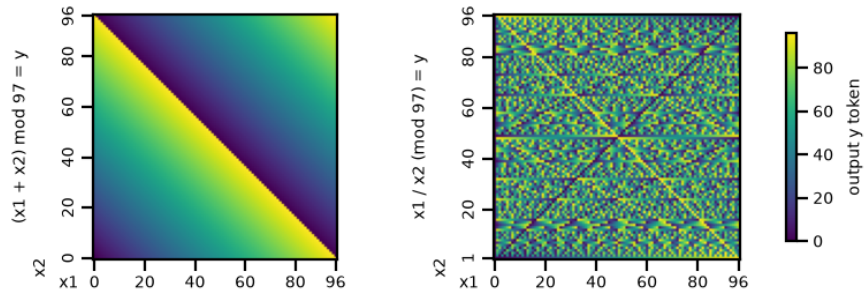


Figure 2. *The complete equation tables of the two tasks at p = 97 — addition on the left, division on the right. Both axes carry the input tokens in their natural order; colour intensity encodes the number of the output token (scale on the right).*

N. The grokking threshold: why three nines

Grokking threshold. Grokking is dated by the moment validation accuracy reaches **0.999**. At this dataset size a threshold of 0.999 means "at most four errors" on the baseline's validation set, while thresholds of 0.9999 - 1.0 mean "no errors at all". Such a demand is expensive: the last four errors sometimes take the network longer to fix than the whole of the path before them and, above all, it fixes them inconsistently. On one baseline seed the three nines are reached at epoch 246 and errorlessness at 869, whereas the median gap across nine seeds is sixteen epochs. To date grokking by a quantity that, on a third of the runs, moves threefold on the strength of one or two examples is to measure not the phenomenon but the tail of its polishing. "Grokking" below therefore means 0.999.

O. Artificial ricochets: the phenomenon, the ablation and slingshots

Three discussions moved out of §4.1: what a ricochet looks like on a run, what collapsing in a random direction gives instead of the escalation, and how a ricochet differs from a slingshot.

O.1. A ricochet on a run

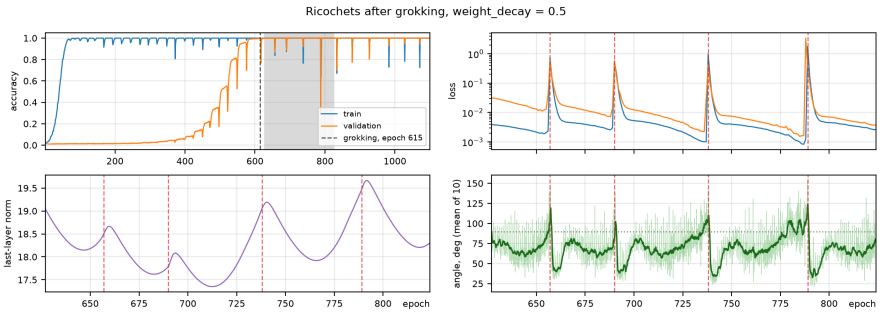


Figure 4.1-b. *Ricochets after grokking at weight_decay = 0.5, seeds 0/0, accuracy thresholds D1 = 0.25, D2 = 0.1. Above, the accuracy of the whole run: the dashed rule is grokking at epoch 615, the grey band is the window of the lower panels. Below, over epochs 627-824, four ricochets in a row (red dashes): the loss, the norm of the last layer's weights, and the angular metric (light — per step, dark — the mean of 10).*

O.2. Ablation: a demolition in a random direction

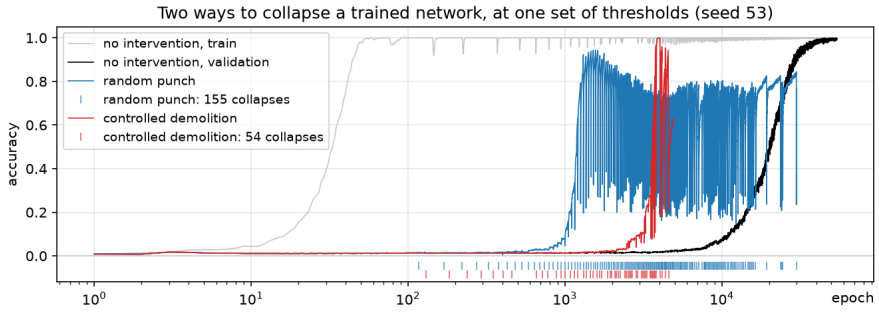


Figure 4.1-c. *Two methods of collapsing a network at one set of thresholds, on the baseline of §3.1 at weight_decay = 0, seed 53: validation accuracy with no intervention (black; the light grey is the same run's training accuracy) groks at epoch 53152, with a random punch (blue) and with learning-rate escalation (red); the ticks under the axis are the accepted demolitions. The punch passes 10 % at epoch 1006 against the escalation's 3014, but never rises above 0.944; the escalation groks at epoch 3921 after 50 demolitions, and the demolitions that follow break the network. Medians over nine seeds are in appendix B.6.*

The mechanism of the method is escalation of the learning rate, that is, an amplification of the very motion the network is making anyway. The natural alternative is to collapse the network with an *arbitrary* motion: if all that matters is destabilization as such, a random push of the right strength ought to work just as well. That variant is implemented and tested, and it is built like the scheme of the previous section, except that what is searched for is not the length of the escalation but the **amplitude of a single blow**.

The scheme of a random punch.

- 1.1. Save a checkpoint with the optimizer's state and, unlike the main method, restore **everything** on a rollback, the order in which batches are served included: the retry runs on the same data. The reason for the difference lies in how the retry itself is built. For the escalation, a retry shortens a deterministic path by one step and therefore stops just short of the cliff every time on the same data, leaving the network undamaged (appendix B.1); for the punch, a retry halves the **amplitude**, and the direction of the blow is drawn from a generator of its own and does not depend on the order of batches at all, so nothing prevents

it from replaying the data. Measured on nine seeds: the two policies give indistinguishable results (appendix B.6).

- 1.2. Add noise to the weights: the direction is drawn afresh for every blow, the amplitude is the current value of the search.
- 1.3. Watch the train accuracy on the batch over R steps after the blow (R = 20, the relaxation period of the main method). The measure of damage is the worst value within that window.
- 1.4. If the worst value stayed **above** the threshold D1, the blow is too weak: return to the checkpoint and double the amplitude.
- 1.5. If it fell **below** the threshold D2, the blow is too strong: return to the checkpoint and halve the amplitude. Once both bounds are known — the weak one and the strong one — the search moves from doubling to bisection.
- 1.6. The blow is accepted when the damage lands between D2 and D1; if 30 attempts fail to achieve that, the demolition is skipped. After an accepted blow the network is left to recover.

Everything else is shared: the firing conditions, the relaxation period, the rollback to the checkpoint when the damage grows during relaxation, the setup of §3.1 at weight_decay = 0, the budget and the thresholds (D1 = 0.25, D2 = 0.1). It matters that the direction be new each time — under blows along one and the same vector, validation accuracy stays at chance for the whole budget: the network learns to compensate for one and the same displacement.

The random punch works, and works fast: generalization arrives an order of magnitude earlier than in the same setup without demolitions, and three times faster than under controlled demolition. Further on, however, the punches do not stop and what has been reached is not held: having come close to full generalization, the network falls back and oscillates until the end of the budget, less and less confidently, never once approaching 99.9 %. Meanwhile the strength that has to be searched for grows by almost two orders of magnitude over the run: the network gets used to the blows, and the calibration drives them ever higher until each demolition is heavier than the last. This is exactly the picture that makes §4.1 switch the demolitions off after grokking, and the ablation confirms it independently: while the punches continue, what has been reached does not consolidate. The possible reasons I take up in the sections that follow.

Plain destabilization, then, really is able to bring out the internal process of accumulating effective representations of the input, slow as it is without regularization. But destabilization damages the network markedly more and, judging by the growth of the strength it needs, calls for finer tuning than a method that repeats the network's own motion. The assessments in this section are qualitative, but they do not rest on single runs: both methods were tested on nine seeds — the same primes from 37 to 71 on which the ablation of table 8.3-a was made. The random punch reached 99.9 % on none of them, the escalation on eight of nine; the full per-run tables and the details of the comparison are in appendix B.6.

O.3. Why this is not a slingshot

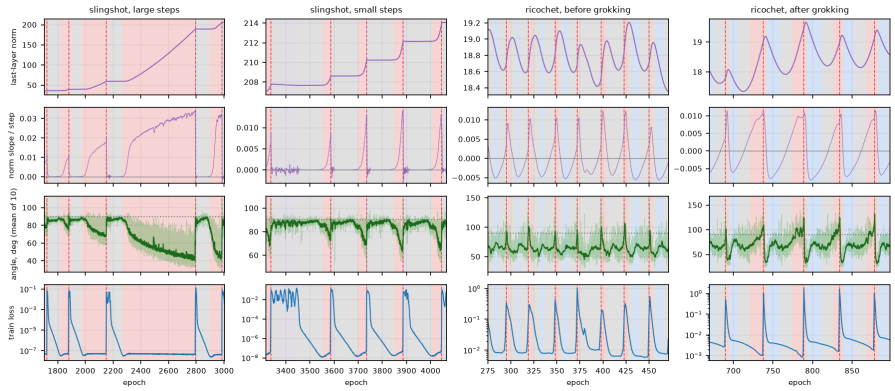


Figure 4.1-SR. *[The number is provisional, pending a pass over the figure numbering.] Slingshots and ricochets on one setup (§3.1, eps = 1e-8, seeds 0/0), by the phase of the last layer's weight norm. The two left columns are slingshots at weight_decay = 0: the large steps of the norm (epochs 1703-3005, the norm growing from 36 to 206) and the small ones (epochs 3318-4066). The two right columns are ricochets at weight_decay = 0.5: before grokking (epochs 275-471) and after it (epochs 670-898, grokking at 615). Top to bottom: the norm of the last layer's weights; its increment per step (a moving average over 20 steps); the angular metric (light — per step, dark — the mean of 10, dashed — 90°); the training loss on a logarithmic scale. Red dashes are the events, the background is the phase of the norm: grey — a plateau, pink — growth, blue — decline.*

The slingshot is described in (Thilak et al., 2022, 2206.04817) as follows: when training continues far beyond complete memorization, the norm of the last layer's weights alternates cyclically between rapid growth and a plateau, and the transition between the phases is accompanied by a spike in the training loss. What follows leans on that work for its terminology and does not set out to revise it: the study of slingshots is not the subject of this paper.

Slingshots as observed in the setting described there (weight_decay = 0), and ricochets, which anyone sees under appreciable weight regularization as sharp brief drops in accuracy, are — as far as the measurements go — two different phenomena, sharing only a temporary collapse of the network and its rapid recovery.

The norm of the last layer's weights. This is what Thilak et al. (2022, abstract and section 1, observations 1-2) single out as the sign by which a slingshot is easily tracked. Every slingshot is preceded by a phase in which the norm grows at an accelerating rate: the increment per step rises throughout the growth and breaks off to zero at the moment of the collapse, after which the norm sits on a plateau (figure 4.1-SR, second row; on the largest step the norm grows from 60 to 189 over 500 epochs). Under ricochets the norm oscillates in something like a sinusoid, and the ricochet itself falls on part of the rising stretch only.

The angular metric. Before a slingshot, as the norm runs away upwards, the angular metric begins to fall — to very small values: to 40° on the largest step, to no lower than 63° on the small ones. At the moment of the collapse the metric returns to 88° and practically never goes past the right angle (no higher than 91.5°). In the case of a ricochet there is a median level away from the events that is below 90° — about 69° in the example shown — and before a ricochet the metric rises until it reaches values well above 90°: up to 106° before grokking and up to 131° after it, on the windows of the figure.

One may suppose that before a slingshot the network enters a particular state that is bad for every training example in the same way, which is what brings the angular metric down. A ricochet, by contrast, is preceded by the network arriving at a place in the landscape for which its present rate is inadmissibly large, and it is knocked out of that region by too strong a rebound off one of the features of the landscape — almost certainly off one of the "banks of the river".

P. Hidden accumulation: the full tables by run

Reaching a threshold. A mark counts as reached not at the first crossing, but from the epoch after which accuracy no longer drops below it. At the lower levels validation accuracy is noisy, and a single outlier steps over 2 % long before the network has learned anything: on one of the seeds such an outlier falls on the second epoch.

The gap from the switching-on of the controlled demolitions to the maximum of validation accuracy, in epochs. The baseline of §3.1, nine seeds; demolitions in the band of §4.1 (D1 = 0.25, D2 = 0.1), budget checkpoint + 7000 epochs, stopping 2000 epochs after the 0.98 mark. The family on which the points of measurement were taken is noted at each table.

By fractions of the way from memorization to a settled 10 %, weight_decay = 0.

seed	memorization	25 %	50 %	75 %	10 % (100 %)
37	4939	4586	3525	2531	2438
41	5451	3722	6561	3287	1535
43	6119	6912	4390	2383	1914
47	4374	4019	3443	708	1442
53	4219	2314	3353	3384	3104
59	3144	4572	3532	2587	1832
61	3217	5125	5099	2239	2350
67	3542	4545	5343	1768	807
71	4325	4217	2742	1831	2318
median	**4325**	**4545**	**3532**	**2383**	**1914**

By fractions of the way from memorization to a settled 10 %, weight_decay = 0.01. This is the table behind the first table of §7.1: the fractions of the way and the checkpoints are taken from the runs of the decayed family, the demolitions are in the same band and on the same budget. The "memorization" column is a run from scratch with the demolitions open from epoch zero; on seed 71 that run is the only one in the whole table not to have reached 0.98, and it stopped on budget at epoch 7199; what goes into the column, there as everywhere, is the epoch of its maximum, and the maximum itself is 0.976.

seed	memorization	25 %	50 %	75 %	10 % (100 %)
37	3087	2715	1160	869	754
41	2869	1909	1831	1105	880
43	3175	2014	1302	1221	683
47	4156	2705	1525	916	614
53	2564	1899	1543	1375	1244
59	2619	1196	1101	785	799
61	4184	1889	1091	973	1025
67	2836	1821	1268	978	603
71	6497	2215	1557	1050	502
median	**3087**	**1909**	**1302**	**978**	**754**

By the levels of validation accuracy, weight_decay = 0.

seed	2 %	4 %	8 %	16 %	24 %	32 %
37	4048	2874	3919	1863	2119	948
41	4837	4989	3319	1291	942	509
43	3884	2323	1865	1744	1056	2273
47	3043	1359	2113	1242	1622	407
53	3510	3086	2599	1309	2947	1428
59	5122	3415	1258	880	525	460

seed	2 %	4 %	8 %	16 %	24 %	32 %
61	4455	5432	3180	1596	1045	1503
67	5544	3041	2104	522	566	877
71	4066	2340	1682	1167	2425	1375
median	**4066**	**3041**	**2113**	**1291**	**1056**	**948**

Q. Weight norms at grokking: measurements across all the methods

A great body of work ascribes a considerable role in grokking to the change of the weight norm. The testbed can record the weight norms of the whole network and of any layers, in any combinations and with any grouping. Let us look at how the norm of the network as a whole behaves, and the norm of the separate groups of matrices. Below, the matrices are grouped by kind and combined across both blocks — the question is how a kind of matrix behaves, not an individual block.

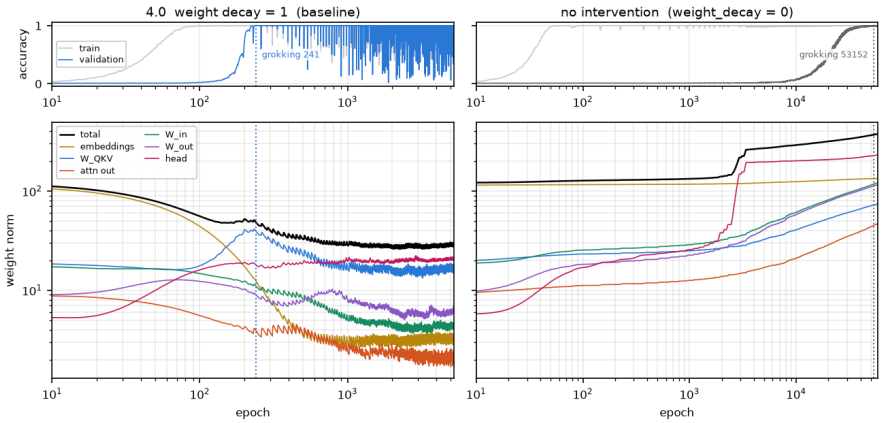


Figure 7.2.1. *Weight norms on the baseline of §4.0 (weight_decay = 1) and on the control with no regularization (weight_decay = 0), seed 53. Above, accuracy; below, the norms: black is the whole network, the colours are six groups of matrices in the order in which the signal passes through them. Both axes are logarithmic and there is no smoothing.*

The behaviour of the weight norms on the baseline made it possible to build theories of a "Goldilocks zone" in which grokking takes place, or to influence grokking through the weight norms. On the control the rapid growth of the last layer's norm is clearly visible — a consequence of the slingshots of (Thilak et al., 2022) — and there is some trend in the behaviour of the other layers as well, even if in the other direction.

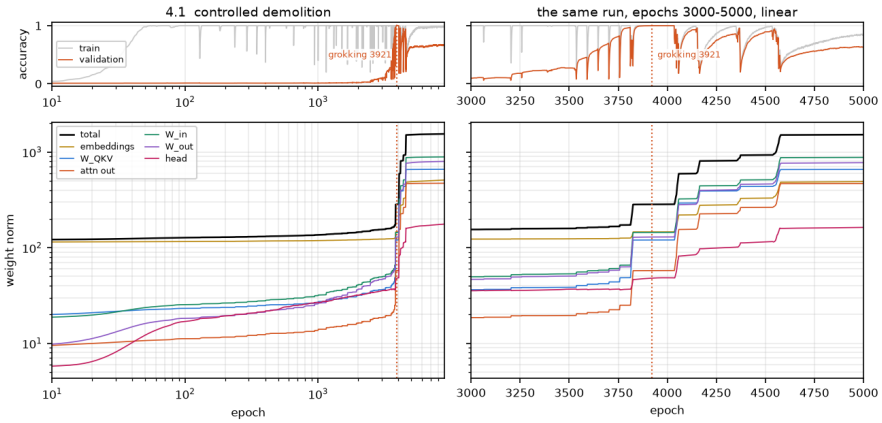


Figure 7.2.2. *Weight norms under controlled demolitions (§4.1): on the left the whole run on a logarithmic scale, on the right epochs 3000–5000 on a linear one. Seed 53, weight_decay = 0.*

A comparison with the three methods proposed here shows a varied and quite different behaviour. Under controlled ricochets the norm creeps upwards in barely noticeable steps for a long while, but everything changes once we reach the stretch where grokking shows itself outright: there follows a sharp jerk upwards, which goes on after grokking as well and happens almost entirely inside the ricochets. This looks nothing at all like a striving to get into some zone.

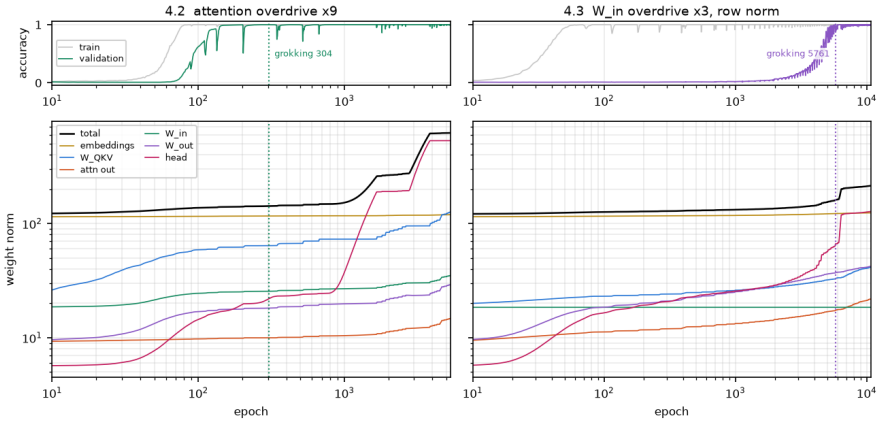


Figure 7.2.3. *Weight norms for the two methods based on overdrive: on the left the attention LR overdrive (§4.2), on the right the* `W_in` *overdrive with row normalization (§4.3). Seed 53, weight_decay = 0.*

The two other methods show behaviour that is on the whole qualitatively like the control measurement. It is notable that in these two cases, as in the control at weight_decay = 0, the norm of the embedding matrices in effect does not change, although on the baseline it made the largest contribution to the overall fall of the weight norms, and from the ablations we know that grokking can be brought about by pressing on it alone (appendix R). Here too the ricochets raise the weight norm abruptly and appreciably, but in the classical baseline regularization quickly brings it back, whereas in all the other methods that does not happen.

The measurements were taken on a single seed. Biases and normalization parameters are included in the overall norm but have no curve of their own.

R. Localizing weight_decay: which matrices are necessary

The ablation that Observation 1 of §4.2 rests on. The regularization is applied not to the whole network but to a chosen subset of matrices, and the question is which subset suffices for grokking.

Table R-a. The hierarchical ablation: which matrices weight_decay is applied to and what comes of it. The baseline of §3.1 unchanged, a budget of $2 \cdot 10^4$ epochs, nine seeds — the primes from 37 to 71, one run per seed (261 runs in all). The grokking threshold is 0.999, as throughout this work (§3.1). Biases and the gains of the normalizations take no part in the ablation. The "grok" column is how many of the nine runs reached 99.9 %. The marks are medians of the epoch number over those runs that reached them; a row may therefore have a median at 90 % and a dash at 99.9 %. The last column is the median of the maximum validation accuracy over the runs that did not reach 99.9 %.

level	where decay is applied	grok	10 %	50 %	90 %	99.9 %	max. without grok
0	all parameters (baseline)	9/9	168	217	245	273	—
0	all matrices	9/9	197	237	267	326	—
0	nothing (control)	0/9	10541	13717	17916	—	0.414
1	blocks only	8/9	305	411	535	1090	0.998
1	embeddings only	9/9	1516	2472	3352	4330	—
1	all matrices except embeddings	9/9	266	331	391	510	—
1	head only	0/9	—	—	—	—	0.031
2	attention halves only	8/9	667	970	1176	1534	0.999
2	FFN halves only	0/9	—	—	—	—	0.023
2	`W_QKV` only, both blocks	9/9	1748	2667	3457	4114	—
2	`W_out` only, both blocks	9/9	4320	6991	8136	9472	—
2	`W_in` only, both blocks	0/9	7460	16175	—	—	0.096
2	`W_down` only, both blocks	0/9	15969	—	—	—	0.102
3	block 0, `W_QKV`	3/9	5380	7143	10288	11614	0.981
3	block 0, `W_out`	1/9	7709	10622	14016	8832	0.955
3	block 0, `W_in`	0/9	9309	14987	17525	—	0.924
3	block 0, `W_down`	0/9	10688	13753	15832	—	0.703
3	block 1, `W_QKV`	0/9	10448	13489	16573	—	0.330
3	block 1, `W_out`	0/9	10744	14312	—	—	0.245
3	block 1, `W_in`	0/9	15172	16252	—	—	0.108
3	block 1, `W_down`	0/9	14981	—	—	—	0.059
4	K and V (no Q)	9/9	2036	3096	4015	4915	—
4	Q and V (no K)	9/9	2888	4479	5800	6555	—
4	V only	8/9	4933	8219	9402	9996	0.988
4	Q and K (no V)	5/9	5471	7682	9971	17709	0.996
4	K only	1/9	6317	9077	11841	14216	0.996
4	Q only	1/9	7659	12126	15467	13836	0.963
5	V only, block 0	2/9	8112	12002	15306	16102	0.944
5	V only, block 1	0/9	11002	14822	—	—	0.247

Levels 1–2 and 4 apply decay to both copies of the matrices in question (in both blocks), level 3 and level 5 to the matrices of one block; the level-4 rows are therefore comparable with the "attention halves" row and not with the "block 0, `W_QKV`" row. The last four rows of level 2 are same-name pairs: each matrix taken in both blocks at once, but without its neighbour within the half. They were added later than the rest precisely because reading a pair off two level-3 rows turned out to be impossible (see below). Full memorization (train accuracy 1.0) sets in, in every run without exception, between epoch 73 and epoch 197: where the decay is applied governs generalization alone.

The first thing the table shows is the price of localization. Decay on every parameter brings the network to 99.9 % in 273 epochs; any local subset is an order of magnitude slower: the attention halves 1534 epochs, `W_QKV` of both blocks 4114, the embeddings 4330, K and V 4915, V alone 9996. No localization comes near full decay; what is gained by it is not speed but a view of which

matrices take part. Arranging the local subsets in a strict order is something these data do not permit, and there is no need to: the gaps between neighbours in the list are smaller than the spread over seeds — for the embeddings and the `W_QKV` pair the medians differ by 5 % at almost coinciding ranges — and for the subject of this paper it is enough that `W_QKV` is on that list.

The second is the role of **V**. It runs through every level: subsets that include V grok in 8–9 runs of nine, those without V less often and later. But the statistics do not confirm that V is necessary: the Q and K pair, in which V is absent, reaches full generalization in five runs of nine, twice as slowly as the pair with V (17709 against 4915 for K and V). On a single seed this row looked like a halt at 0.83; nine seeds show that V speeds generalization up rather than creating it. For the same reason the claims about Q and K taken separately have been withdrawn: each of them reaches the 99.9 % mark once in nine.

The third is the asymmetry between the blocks, which held up and grew more precise. All four matrices of the first block carry the network past 10 % and reach the median of 90 % within the budget; of the second block's matrices only `W_QKV` and `W_out` manage it, while the second block's `W_in` and `W_down` stay at the level of 0.06–0.11 — that is, **below the control** with no regularization at all. Level 5 shows the same on the V pair: the V of the first block alone groks twice in nine, the V of the second alone not once, whereas both copies together grok eight times. The sufficient minimum turns out to be not a matrix but a same-name pair of matrices in both blocks.

The four same-name-pair rows added to level 2 show that two level-3 rows cannot be added up in one's head: a pair does not behave as the average of its halves. In attention the pair beats both of them: `W_QKV` in both blocks groks nine runs of nine in 4114 epochs, whereas the same `W_QKV` of the first block groks three of nine and of the second none; `W_out` of both blocks also groks nine of nine (9472), against one of nine for the first block and zero for the second. In the FFN it is the other way round: `W_in` of both blocks stays at 0.096 and `W_down` of both blocks at 0.102 — that is, the pair lands not between its halves (0.924 and 0.108 for `W_in`) but level with the worse of them and four times below the control. A harmful copy, then, is not averaged with a useful one but outweighs it. Hence the general rule: in the attention matrices it is the pair that works and not the matrix, and in the FFN matrices it is the pair that gets in the way, again not the matrix.

The fourth is a result nobody anticipated. Decay applied **to the FFN halves only** or **to the head only** leaves the network at 0.023 and 0.031, whereas the control with no regularization whatever reaches 0.414. Regularizing these matrices is not merely useless — it is worse than its absence, and that is robust: nine runs of nine for the FFN and eight of nine for the head hold at the level they have reached for the last four hundred epochs rather than going on rising.

A caveat about the limit of the method: within a budget of $2 \cdot 10^4$ epochs the control with no regularization reaches 90 % (a median of 17916) but does not reach 99.9 % in any of the nine runs — the baseline needs 53152 epochs for that by the median of nine seeds, and no fewer than 31569 even on the fastest of them (appendix B.7). A row of the table should therefore be read relative to the control: a "0/9" with a maximum below 0.414 means that decay on those matrices hinders, while a "0/9" with a higher maximum means that it helps but does not have the time.

This does not mean that grokking needs that matrix alone: under other hyperparameters grokking can be obtained with a different set of regularized matrices as well, and — as I shall show later in this section — with no regularization whatever. Other works have shown that grokking is reachable on networks that have no attention block at all. But for the baseline I have chosen, the subsets that include V are a good working option: they generalize in almost every run, and twice as fast as those in which V is absent. This subset can be called neither the cheapest nor strictly necessary — the statistics over nine seeds show that grokking happens without V as well, merely less often and later, and nine seeds do not allow it to be compared with the embeddings on speed.

The embeddings. The row "embeddings only" deserves a mention of its own: weight_decay applied to the embedding matrices alone and to nothing else brings all nine runs of nine to full generalization — more slowly than full decay (4330 epochs against 273), but reliably. This is the more curious in that works on grokking, by an established tradition, are precisely the ones that do not apply the regularization to the embeddings: it is customary to exempt them from weight decay, following the practice of training language models. Here it turns out that the embeddings alone are enough, and that exempting them from the regularization is a decision that would be worth justifying rather than inheriting. The price of the habit itself is measured by a row of its own: decay on all matrices except

the embeddings brings all nine runs to 99.9 %, but in 510 epochs against 326 for the same matrices together with the embeddings — that is, exempting the embeddings does not abolish grokking but slows it down by a factor of one and a half. This is the ablation promised in §3.1.

S. The angular metric: definition, origin and interpretation

The angular metric is computed extremely simply: I take the gradient on the current batch and on the previous one and compute the cosine angle between them. The quantity is very noisy, so an informative picture is visible only on a smoothed value (a window of order 10–20).

In a space of that many dimensions any two random directions are almost guaranteed to be orthogonal, so any deviation of the angular metric from 90° is extremely valuable information (in the terms of information theory: the less probable a signal, the more information it carries).

The indicator itself — the cosine of the angle between two successive gradients — has long been known in the choice of step size. The scalar product of neighbouring gradients as a regulator of the step appears in Mirzakhmedov and Uryasev (1983; cited in Pflug, 1988) and in Pflug (1983): a negative product means the step is larger than the optimal one, and its expectation is negative if the process is oscillating (Pflug, 1988). In the training of neural networks, the cosine of the angle between neighbouring gradients was already used to adjust the learning rate by Chan and Fallside (1987; cited in Hassoun, 1995); in contemporary work this signal serves as a diagnostic of stationarity (Chee and Toulis, 2018). What is new here is not the metric itself but its use as an online diagnostic of the moment at which the learning_rate becomes destructive for the network.

The expected behaviour of the metric on different landscapes:

- for a loss landscape with wide valleys and steady slopes it is natural to expect an angle substantially smaller than 90°;
- an ordinary broken landscape is characterized by angles of about 90° — or a little smaller, if a trend shows through the landscape;
- if the optimizer is stuck — in a local minimum, say (if such things exist on the landscape at all) — the angle is logically to be expected substantially greater than 90°;
- and if the landscape is a river hemmed in between the steep walls of a canyon, as is conjectured (Wen et al., 2024, 2410.05192), and the network at too large a rate catches one of the walls and starts thrashing from wall to wall like a crippled Imperial fighter — the angle will likewise be substantially greater than 90°, signalling a rate inadmissibly high for that landscape.

T. Metastability: an alternative explanation of delayed generalization

An exposition belonging to §9: what Ersoy and Wiesner (2026) propose, what it was obtained on, and what of it is visible on the testbed of the present work. The subject itself does not belong to the subject matter of this paper, and no position on it is taken here.

What is proposed. Ersoy and Wiesner (2026) describe delayed generalization as the capture of the network by a metastable state, which it leaves when the noise of SGD throws it over an energy barrier. The escape time obeys Arrhenius kinetics with an effective temperature proportional to lr/B. A deliberate “locking in” by way of L2 reproduces the outward signs of grokking — the long delay, the sharpness of the transition and the sensitivity to initialization.

What it was obtained on. The result is derived on deep linear networks, where the landscape is solved exactly; the transfer to non-linear networks and to transformers the authors mark as a conjecture.

What of it is visible on the testbed here. What agrees: the state holds for a very long time and is left abruptly under a sufficient perturbation. What diverges: with no perturbation whatever the network arrives at generalization on its own — a median of 53152 epochs over nine seeds (appendix B.7) — while the runs that fail to grok are for the most part not standing still but go on rising slowly, finishing the budget between 0.92 and 0.998; the single verified case of a halt came at 0.997, that is, right at the threshold.

The landscape. In the literature the landscape of this task is described as an ill-conditioned valley — flat along the direction of movement and steep across it (Notsawo et al., 2023); measurements of the curvature along the training trajectory on the same setting find no significant negative curvature anywhere except in the slingshots themselves. This is that picture of a river in a canyon to which appendix S refers (Wen et al., 2024). The natural slingshots, moreover, are cyclic (Thilak et al., 2022): the network leaves the state many times over and returns to the same surface.