PROGRESS MEASURES FOR GROKKING VIA MECHANISTIC INTERPRETABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks often exhibit emergent behavior, where qualitatively new capabilities arise from scaling up the amount of parameters, training data, or training steps. One approach to understanding emergence is to find continuous *progress measures* that underlie the seemingly discontinuous qualitative changes. We argue that progress measures can be found via mechanistic interpretability: reverse-engineering learned behaviors into their individual components. As a case study, we investigate the recently-discovered phenomenon of "grokking" exhibited by small transformers trained on modular addition tasks. We fully reverse engineer the algorithm learned by these networks, which uses discrete Fourier transforms and trigonometric identities to convert addition to rotation about a circle. We confirm the algorithm by analyzing the activations and weights and by performing ablations in Fourier space. Based on this understanding, we define progress measures that allow us to study the dynamics of training and split training into three continuous phases: memorization, circuit formation, and cleanup. Our results show that grokking, rather than a sudden shift, arises from the combination of a gradual amplification of structured mechanisms encoded in the weights and the removal of components from the network that perform memorization.

1 INTRODUCTION

Neural networks often exhibit emergent behavior, in which qualitatively new capabilities, often *generalizing* beyond the training data, arise from scaling up model size, training data, or the number of training steps (Steinhardt, 2022; Wei et al., 2022a). This has led to a number of breakthroughs, via capabilities such as in-context learning (Radford et al., 2019; Brown et al., 2020) and chain-of-thought prompting (Wei et al., 2022b). However, it also poses risks: Pan et al. (2022) show that scaling up the parameter count of models by as little as 30% can lead to emergent reward hacking.

Emergence is most surprising when it is abrupt, as in the case of reward hacking, chain-of-thought reasoning, or other phase transitions (Ganguli et al., 2022; Wei et al., 2022a). We could better understand and predict these phase transitions by finding *hidden progress measures* (Barak et al., 2022): metrics that precede, and are causally linked to, the phase transition, and which vary more smoothly. For example, Wei et al. (2022a) show that while large language models show abrupt jumps in their performance on many benchmarks, their cross-entropy loss decreases smoothly with model scale. However, cross-entropy doesn't explain *why* the phase changes happen.

In this work we introduce a different approach to uncovering hidden progress measures: via *mechanistic explanations*.[1] A mechanistic explanation aims to reverse engineer the mechanisms of the network, generally by identifying the circuits (Cammarata et al., 2020; Elhage et al., 2021) within a model that implement a behavior. We instantiate this strategy to study *grokking*, where overparameterized models abruptly transition to a generalizing solution after initially overfitting. Specifically, we study modular addition, where a transformer takes inputs $a, b \in \{0, \ldots, P-1\}$ for some prime P and predicts their sum c mod P. Small transformers trained with weight decay on this task exhibit grokking (Appendix C.2).

[1] Interactive versions of figures, as well as the code to reproduce our results, are available at `bit.ly/progress-measures-grokking-website`.

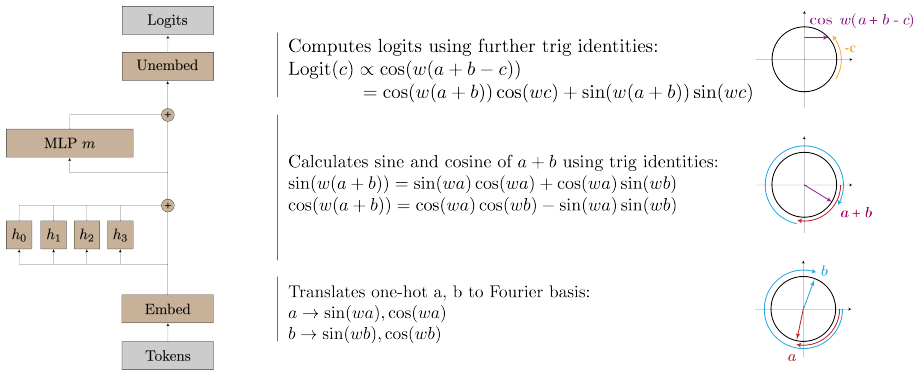

Figure 1: The algorithm implemented by the one-layer transformer for modular addition. Given two numbers a and b, the model projects each point onto a corresponding rotation using its embedding matrix. Using its attention and MLP layers, it then composes the rotations to get a representation of $a + b \mod P$. Finally, it "reads off" the logits for each $c \in \{0, 1, ..., P-1\}$, by rotating by $-c$ to get $\cos(w(a+b-c))$, which is maximized when $a + b \equiv c \mod P$ (since w is a multiple of $\frac{2\pi}{P}$).

We reverse engineer the weights of these transformers and find that they perform this task by projecting the inputs onto a circle and performing addition on the circle. Specifically, we show that the embedding matrix maps the inputs a, b to sines and cosines at a sparse set of *key frequencies* w_k. The attention and MLP layers then combine these using trigonometric identities to compute the sine and cosine of $w_k(a+b)$, and the output matrices shift and combine these frequencies.

We confirm this understanding with several lines of evidence. For example, we show that the embedding/unembedding matrices consist of sines and cosines at key frequencies. Most MLP neurons use a single key frequency and the activations and logits of the model are well approximated by components of the interpreted algorithm. Ablating key frequencies used by the model reduces performance to chance, while ablating the other 95% of frequencies slightly *improves* performance.

Using our understanding of the algorithm implemented by these transformers, we construct two measures of progress during training on this task—*restricted loss*, where we ablate every non-key frequency, and *excluded loss*, which measures performance with key frequencies removed. Both metrics improve continuously prior to when grokking occurs. We use these metrics to understand the training dynamics underlying grokking and find that training can be split into three phases: **memorization** of the training data; **circuit formation** of the mechanism that implements the generalizing algorithm; and **cleanup**, where weight decay removes the transformer's memorizing algorithm. Surprisingly, the sudden transition to perfect test accuracy in grokking occurs during cleanup, *after* the generalizing mechanism is learned. While significant work remains for using this approach on larger networks and more realistic tasks, we believe that our results show that using mechanistic interpretability to define progress measures can further our understanding of emergent phenomena.

2 Related Work

Phase Changes. Recent papers have observed that neural networks quickly develop novel qualitative behaviors as they are scaled up or trained longer (Ganguli et al., 2022; Wei et al., 2022a). McGrath et al. (2021) find that AlphaZero quickly learns many human chess concepts between 10k and 30k training steps and reinvents human opening theory between 25k and 60k training steps.

Grokking. Grokking was first reported in Power et al. (2022), which trained two-layer transformers on several algorithmic tasks and found that test accuracy often increased sharply long after achieving perfect train accuracy. Millidge (2022) suggests that this may be due to SGD being a random walk on the optimal manifold. Our results echo Barak et al. (2022) in showing that the network instead makes continuous progress toward the generalizing algorithm. Liu et al. (2022) construct small examples of grokking, which they use to construct phase diagrams with four separate "phases" of learning. Thilak et al. (2022) argue that grokking can arise even without explicit regularization from an optimization anomaly they dub the slingshot mechanism, which may act as an implicit regularizer.

Mechanistic interpretability and progress measures. There is a growing literature that seeks to mechanistically understand neural networks. The style of post-hoc mechanistic interpretability

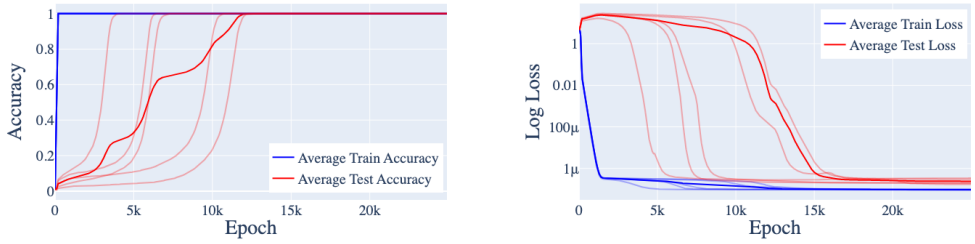

Figure 2: The train and test accuracy (left) and train and test loss (right) of one-layer transformers on the modular addition task described in Section 3, over 5 random seeds. These models consistently exhibit grokking: they quickly overfit early on in training, but then later learn to generalize.

in Section 4 is heavily inspired by the Circuits approach of Cammarata et al.(2020), Elhage et al. (2021), and Olsson et al. (2022). Meng et al. (2022) use patching and restoration to locate the components in GPT-2 that encode factual knowledge. Barak et al. (2022) introduce the notion of *progress measures*—metrics that improve smoothly and that precede emergent behavior. They prove theoretically that training would amplify a certain mechanism and heuristically define a progress measure. In contrast, we use mechanistic intepretability to discover progress measures empirically.

3 Setup and Background

We train transformers to perform addition mod P. The input to the model is of the form "$a\ b\ =$", where a and b are encoded as P-dimensional one-hot vectors, and $=$ is a special token above which we read the output c. In our mainline experiment, we take $P = 113$ and use a one-layer ReLU transformer, token embeddings with $d = 128$, learned positional embeddings, 4 attention heads of dimension $d/4 = 32$, and $n = 512$ hidden units in the MLP. In other experiments, we vary the depth and dimension of the model. We did not use LayerNorm or tie our embed/unembed matrices.

Our mainline dataset consists of 30% of the entire set of such problems (that is, 30% of the $113 \cdot 113$ pairs of numbers mod P). We use full batch gradient descent, weight decay (Hanson & Pratt, 1988) and the AdamW optimizer (Loshchilov & Hutter, 2017) with learning rate $\gamma = 0.001$ and weight decay parameter $\lambda = 1$. We perform $40,000$ epochs of training. As there are only $113 \cdot 113$ possible pairs, we can evaluate the model's performance on all points in the distribution.

Networks trained on this task consistently exhibit grokking (Power et al., 2022). As Figure 2 shows, the networks first overfit the training set: train accuracy quickly converges to 100% and the train loss quickly declines, while the test accuracy remains low and the test loss remains high. After around $10,000$ epochs, the network generalizes as test accuracy also increases to near 100%.[2]

To denote our transformer, we follow the conventions and notations laid out in Elhage et al. (2021). For our purposes, we focus on the embedding matrix W_E (of shape $d \times P$), the second matrix in the MLP layer W_{out} (of shape $d \times n$), and the unembedding matrix W_U (of shape $P \times d$).[3] Empirically, the transformer does not use the skip connection around the MLP layer as a significant part of its computation (Appendix A.1), so $L \approx W_U W_{out} N$, where L are the output logits and N is the neuron activations (which are in turn a function of the embeddings and the attention heads). We therefore also focus on the composition of W_U and W_{out}, $W_L = W_U W_{out}$ (of shape $P \times n$).

3.1 The Fourier multiplication algorithm

We claim that the learned networks use the following algorithm:

- Given two one-hot encoded tokens a, b, for various $k \in \mathbb{N}$, map these to $\sin(w_k a)$, $\cos(w_k a)$, $\sin(w_k b)$, and $\cos(w_k b)$ using the embedding matrix, where $w_k = \frac{2k\pi}{P}$.

[2] In Appendix C.2, we vary the the prime P as well as the network architecture and number of layers, and find that grokking does not depend on any of these factors. However, regularization is crucial: in Appendix D.1, we show that weight decay or some other form of regularization that encourages the network to "spread out" its computation is necessary for grokking to occur, and smaller weight decay λ causes the network to take longer to grok.

[3] We ignore the embedding and unembedding of the '$=$' token for simplicity.

- Compute $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$ using the trigonometric identities:
$$\cos(w_k(a+b)) = \cos(w_k a)\cos(w_k a) - \sin(w_k a)\sin(w_k b)$$
$$\sin(w_k(a+b)) = \sin(w_k a)\cos(w_k b) + \cos(w_k a)\sin(w_k b)$$
In our networks, this is computed in the attention and MLP layers.
- For each output logit c, compute $\cos(w_k(a+b-c))$ using the trigonometric identity:
$$\cos(w_k(a+b-c)) = \cos(w_k(a+b))\cos(w_k c) + \sin(w_k(a+b))\sin(w_k c).$$
This is a linear function of the already-computed values $\cos(w_k(a+b))$, $\sin(w_k(a+b))$ and is implemented in the product of the output and unembedding matrices W_L.
- The unembedding matrix also adds together $\cos(w_k(a+b-c))$ for the various ks. This causes the cosine waves to constructively interfere at $c^* = a + b \mod p$ (giving c^* a large logit), and destructively interfere everywhere else (and thus small logits for other cs).

We will justify this in detail in Section 4.

4 Reverse engineering a one-layer transformer

In this section, we describe four lines of evidence that our transformers are using the Fourier multiplication algorithm described in Section 3.1. Here we apply our analysis to the mainline model from Section 3; the results are broadly consistent for other models (as we confirm in Appendix C.2).

Our first line of evidence involves examining the network weights and activations and observing consistent **periodic structure** that was unlikely to occur by chance (Section 4.1). Moreover, when we take Fourier transforms, many components are either sparse or nearly sparse in the Fourier domain, supported on a handful of *key frequencies*.[4]

We next look into the actual **mechanisms** implemented in the model weights (Section 4.2). We show that the unembedding matrix W_L is rank 10, where each direction corresponds to the cosine or sine of one of 5 key frequencies. Projecting the MLP activations onto the components of W_L approximately produces multiples of the functions $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$, showing that the MLP layer does compute these sums.

To better understand the mechanism, we **zoom in** to individual neurons (Section 4.3). We find that most neurons are well-approximated by sines and cosines at a *single* frequency. Moreover, the corresponding direction in W_L also contains only that frequency. This suggests that the models' computations are (1) localized across frequencies and (2) mostly aligned with the neuron basis.

Finally, we use **ablations** to confirm that our interpretation is faithful (Section 4.4). We replace various components of the model by the components of the algorithm we found above and find that doing so does not harm and sometimes improves model performance.

4.1 Suggestive evidence: surprising periodicity

The first line of evidence that the network is using the algorithm described in Section 3.1 is the surprising periodicity in the activations of the transformer. That is, the output of every part of the network is periodic as a function of the input tokens.

Periodicity in the embeddings. We start by examining the embeddings. We apply a Fourier transform along the input dimension of the embedding matrix W_E then compute the ℓ_2-norm along the other dimension; results are shown in Figure 3. We plot only the components for the first 56 frequencies, as the norm of the components for frequencies k and $P-k$ are symmetric. The embedding matrix W_E is sparse in the Fourier basis–it only has significant nonnegligible norm at 6 frequencies. Of these frequencies, 5 frequencies are the *only* frequencies used by other parts of the model (corresponding to $k \in \{14, 35, 41, 42, 52\}$). We dub these the *key frequencies* of the model.

Periodicity in attention heads and MLP neuron activations. This periodic structure recurs throughout the network. As an example, we plot the attention weight at position 0 for every combination of two inputs for head 0 in Figure 4. The attention exhibits a periodic structure with frequency

[4]Note that the actual frequencies and even number of frequencies vary over different training runs and architectures (Appendix C.2).

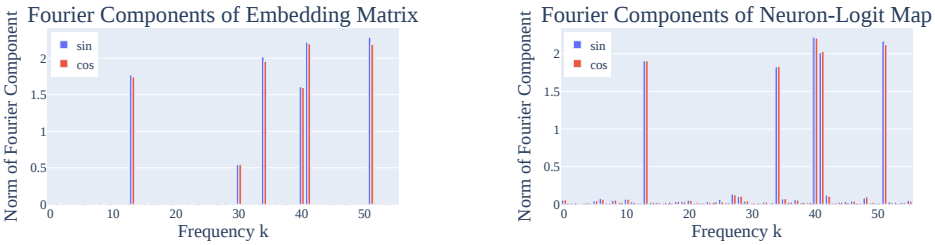

Figure 3: *(Left)* The norms of the Fourier components in the embedding matrix W_E. As discussed in Section 4.1, the sparsity of W_E in the Fourier basis is evidence that the network is operating in a Fourier basis. Of the six non-zero frequencies, five frequencies appear in later parts of the network, corresponding to $k \in \{14, 35, 41, 42, 52\}$. *(Right)* The norms of the direction corresponding to sine and cosine waves in the neuron-logit map weights W_L. As discussed in Section 4.2, W_L is well approximated by $\sin(w_k c)$ and $\cos(w_k c)$ for the 5 key frequencies w_k.

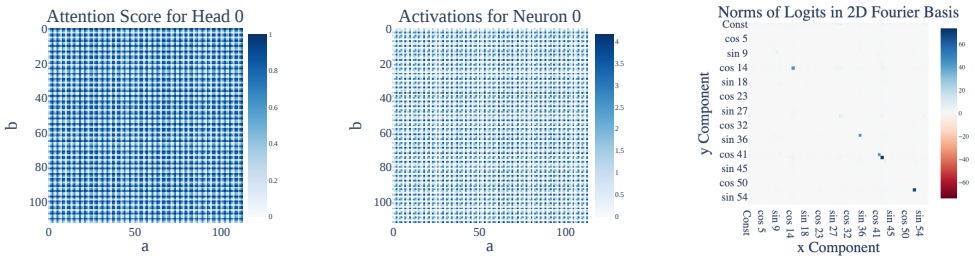

Figure 4: *(Left)* The attention score for head 0 from the token '$=$' to 'a', as a function of inputs a, b. *(Center)* The activations of MLP neuron 0 given inputs a, b. Both the attention scores and the neuron activations are periodic (Section 4.1). *(Right)* The norm of the Fourier components of logits. There are 20 significant components corresponding to the 5 key frequencies (Section 4.1).

$k = 35$. Other heads (Appendix C.1) also exhibit periodicity, always with one of five key frequencies. In Figure 4, we plot the activations of neuron 0, given inputs a and b. The activations are periodic with frequency $k = 42$. We see similar patterns for other MLP neurons (Appendix C.1).

Periodicity in logits. Finally, the logits also are periodic. In Figure 4, we represent the logits in the 2D Fourier basis over the inputs then take the ℓ_2-norm of each component. There are only twenty components with significant norm, corresponding to the products of sines and cosines for the five key frequencies w_k. These show up as five 2×2 blocks in Figure 4.

4.2 Mechanistic Evidence: Composing Model Weights

We now demonstrate that the model computes its logits using the trigonometric identity for $\cos(w_k(a + b - c))$. First, we'll show that the logits are well approximated (95% of variance explained) by terms of the form $\cos(w_k(a + b - c))$. Second, we'll show that W_L is rank 10 and can be written as:

$$W_L = \sum_{k \in \{14,35,41,42,52\}} u_k \cos(w_k c)^T + v_k \sin(w_k c)^T$$

for some $u_k, v_k \in \mathbb{R}$. Third, note that our model implements the logits for a, b as:

$$\text{logits}(a, b) = \text{MLP}(a, b) W_L = \sum_k \text{MLP}(a, b) u_k \cos(w_k c)^T + \text{MLP}(a, b) v_k \sin(w_k c)^T$$

We'll show that the projections of $\text{MLP}(a, b)$ in the $\cos(w_k c)$ and $\sin(w_k c)$ directions are multiples of $\cos(w_k(a + b))$ and $\cos(w_k(a + b))$ respectively. Taken together, these three steps demonstrate that the network computes its logits using the corresponding trigonometric identity.

Logits are well approximated by a weighted sum of $\cos(w_k(a + b - c))$s. First we approximate the output logits as a sum of the form $\sum_k \alpha_k \cos(w_k(a + b - c))$ for $k \in \{14, 35, 41, 42, 52\}$ and fit the coefficients α_k via a linear regression. This approximation explains 95% of the variance in the original logits. This is surprising—naively, the output logits are a $113 \cdot 113 \cdot 113$ dimensional vector,

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_{14}c)$	$44.6\cos(w_{14}a)\cos(w_{14}b) - 43.6\sin(w_{14}a)\sin(w_{14}b) \approx 44.1\cos(w_{14}(a+b))$	93.2%
$\sin(w_{14}c)$	$44.1\sin(w_{14}a)\cos(w_{14}b) + 44.1\cos(w_{14}a)\sin(w_{14}b) \approx 44.1\sin(w_{14}(a+b))$	93.5%
$\cos(w_{35}c)$	$40.7\cos(w_{35}a)\cos(w_{35}b) - 43.6\sin(w_{35}a)\sin(w_{35}b) \approx 42.2\cos(w_{35}(a+b))$	96.8%
$\sin(w_{35}c)$	$41.8\sin(w_{35}a)\cos(w_{35}b) + 41.8\cos(w_{35}a)\sin(w_{35}b) \approx 41.8\sin(w_{35}(a+b))$	96.5%
$\cos(w_{41}c)$	$44.8\cos(w_{41}a)\cos(w_{41}b) - 44.8\sin(w_{41}a)\sin(w_{41}b) \approx 44.8\cos(w_{41}(a+b))$	97.0%
$\sin(w_{41}c)$	$44.5\sin(w_{41}a)\cos(w_{41}b) + 44.5\cos(w_{41}a)\sin(w_{41}b) \approx 44.5\sin(w_{41}(a+b))$	97.0%
$\cos(w_{42}c)$	$64.6\cos(w_{42}a)\cos(w_{42}b) - 68.5\sin(w_{42}a)\sin(w_{42}b) \approx 66.6\cos(w_{42}(a+b))$	96.4%
$\sin(w_{42}c)$	$67.8\sin(w_{42}a)\cos(w_{42}b) + 67.8\cos(w_{42}a)\sin(w_{42}b) \approx 67.8\sin(w_{42}(a+b))$	96.4%
$\cos(w_{52}c)$	$60.5\cos(w_{52}a)\cos(w_{52}b) - 65.5\sin(w_{52}a)\sin(w_{52}b) \approx 63.0\cos(w_{52}(a+b))$	97.4%
$\sin(w_{52}c)$	$64.5\sin(w_{52}a)\cos(w_{52}b) + 64.5\cos(w_{52}a)\sin(w_{52}b) \approx 64.5\sin(w_{52}(a+b))$	98.2%

Table 1: For each of the directions in the unembedding matrix, we project the MLP activations in that direction then perform a Fourier transform. For brevity, we omit terms with coefficients less than 13% of the largest coefficient. We then compute the fraction of variance explained (FVE) if we replace the projection with a multiple of a single term of the form $\cos(w_k(a+b))$ or $\sin(w_k(a+b))$, and find that this is consistently close to 1.

but it's well approximated with just the 5 directions predicted by our interpretation. If we evaluate test loss using this logit approximation, we actually see an *improvement* in loss, from $2.4 \cdot 10^{-7}$ to $4.7 \cdot 10^{-8}$.

The W_L is well approximated by $\cos(w_k c)$ and $\sin(w_k c)$. We perform a discrete Fourier transform (DFT) on the logit axis of W_L and look at the 10 directions corresponding to $\sin(w_k c)$ and $\cos(w_k c)$. When we approximate W_L with these 10 directions, the residual has Frobenius norm that is under 0.55% of the norm of W_L.

The unembedding matrix "reads off" terms of the form $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$ from the MLP neurons. Finally, we take the dot product of the MLP activations with each of the 10 directions in W_L. Table 1 displays the results in the Fourier basis: projecting the MLP activations onto the $\cos(w_k c)$ direction only has non-trivial terms of the form $\cos(w_k a)\cos(w_k b)$ and $\sin(w_k a)\sin(w_k b)$, and projecting onto the $\sin(w_k c)$ direction gives us terms of the form $\sin(w_k a)\cos(w_k b)$ and $\sin(w_k a)\cos(w_k b)$, with approximately equal coefficients.

4.3 Zooming In: Approximating Neurons with Sines and Cosines

In the previous section, we showed how the model computes its final logits by examining the weights of W_L. Now, we examine the role that individual neurons play in the model, by approximating their behavior on all $113 \cdot 113$ pairs of possible inputs in the input distribution.

Attention heads approximately compute products of sinusoidal functions. In order to compute terms like $\cos(w_k(a+b))$, the model needs to multiply the results of the embedding matrix together. One natural place to perform this computation is the attention heads. By performing a DFT on the scores of the 4 attention heads from the third sequence position ('=' token) to the first sequence position ('a' token), we find the scores are well approximated by $0.5 + C(\cos(w_k a) - \cos(w_k b)) + D(\sin(w_k a) - \sin(w_k b))$ for a single frequency w_k, where C, D are constants (Appendix C.1.1). As the token embeddings are also sinusoidal functions of the input, the output of the attention heads is approximately a quadratic form of sinusoidal functions, as is necessary for the identified algorithm.

Most MLP neurons are well approximated by a quadratic form of sinusoidal functions of a single frequency. If our interpretation is correct, then we expect that the MLP neurons will compute quadratic forms on $\sin(w_k a), \sin(w_k b), \cos(w_k a), \cos(w_k b)$ of a single frequency w_k. To check this, we attempt to approximate every MLP neuron with such a quadratic form for each of the 5 frequencies identified in Section 4.1. Out of a total of 512 neurons, 433 (84.6%) have over 85% variance explained with some frequency. Figure 5 displays the histogram.

Maps to the logits are localized by frequency. We next partition these 433 neurons by the frequencies with the highest variance explained. In each of these clusters, the weights of the map W_L from the neurons to the logits are only non-trivial for two components in the Fourier basis—corresponding to sine and cosine functions with the corresponding frequency. We plot the W_L weights of the 44 neurons in the $k = 14$ cluster in the Fourier basis in Figure 5. As the figure shows, the non-negligible components are $\sin\left(\frac{2k\pi}{P}\right)$ and $\cos\left(\frac{2k\pi}{P}\right)$ for $k = 14$, consistent with the hypothesis that these neurons only use the $k = 14$ sine and cosine waves.

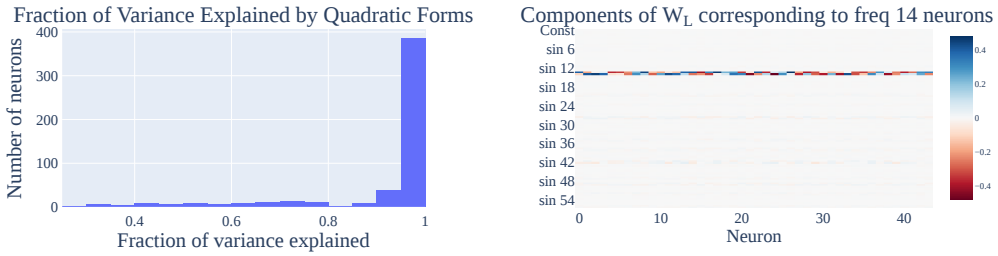

Figure 5: (Left) Most neurons are well-approximated by quadratic forms on sinusoidal functions of the inputs. (Right) The weights in W_L corresponding to each of the 44 neurons of frequency 14. The non-trivial components correspond to $\sin(w_k)$ and $\cos(w_k)$ for $k = 14$.

4.4 Correctness checks: ablations

In previous sections, we showed that various components of the model were well-approximated by sparse combinations of sines and cosines. We verify that these approximations are faithful to the model's functionality, by replacing each component with its approximation. This generally does not hurt the performance of the model and in some cases *improves* it.

MLP neurons. In Section 4.3, we identified 433 neurons that were well-approximated by a quadratic form. We replace each of these neurons' activation value by the corresponding quadratic form, leaving the other neurons untouched. This increases loss by only 3% in relative terms (from $2.41 \cdot 10^{-7}$ to $2.48 \cdot 10^{-7}$) and has no effect on accuracy.

We can instead apply a subtly different ablation to the MLP layer and replace its activations with $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$ in the key frequencies. This *improves* loss by 77% (to $5.54 \cdot 10^{-8}$), validating that the logits are calculated by trig identities of neurons as detailed in Section 4.2.

Logit frequencies. Next, we ablate various components of the final logits in the Fourier space. To do so, we take a 2D DFT on the $113 \cdot 113 \cdot 113$ logit matrix over all $113 \cdot 113$ pairs of inputs to get the logits in the Fourier basis, then set various pairs of frequencies in this basis to 0.

We begin by ablating the components corresponding to each of the key frequencies. As reported in Table 2, ablating any key frequency causes a significant increase in loss. This confirms that the five frequencies identified in previous sections are indeed necessary components of the transformer.

In contrast, ablating other frequencies does not hurt the model at all. We ablated three randomly selected other frequencies and observed no impact on performance (Table 2). Finally, we ablated *all* $113 \cdot 113 - 40$ of the Fourier components besides key frequencies; this *improved* performance. Insofar as other frequencies are used, they appear to serve as noise that degrades performance.

Directions in W_L. In Section 4.2, we found that W_L is well approximated by the 10 directions corresponding to the cosine and sine of key frequencies. If we project the MLP activations to these 10 directions, loss decreases 50% to $1.19 \cdot 10^{-7}$. If we instead projected the MLP activations onto the nullspace of these 10 directions, loss increases to 5.27—worse than uniform. This tells us that the logits depend on these and *only* these 10 directions.

5 Understanding grokking behavior using progress measures

We now use our mechanistic understanding of the network to define two *progress measures*: metrics that can be computed during training that track the progress of the model over the course of training. This allows us to better understand *how* the network reaches its final solution.

5.1 Progress measures

We translate the ablations in Section 4.4 into progress measures: restricted loss and excluded loss.

Restricted loss. Since the final network uses a sparse set of frequencies w_k, it makes sense to check how well intermediate versions of the model can do using only those frequencies. To measure this, we perform a 2D DFT on the logits to write them as a linear combination of waves in a and b,

Frequency w_k ablated	(None)	5	11	38	All unused
Loss	$2.41 \cdot 10^{-7}$	$2.40 \cdot 10^{-7}$	$2.42 \cdot 10^{-7}$	$2.42 \cdot 10^{-7}$	$7.24 \cdot 10^{-8}$
Frequency w_k ablated	14	35	41	42	52
Loss	$1.04 \cdot 10^{-3}$	$1.21 \cdot 10^{-3}$	$3.34 \cdot 10^{-3}$	$7.60 \cdot 10^{-2}$	$4.64 \cdot 10^{-2}$

Table 2: The loss of the transformer when ablating three randomly sampled frequencies $k = 5, 11, 38$, *everything except for* the five key frequencies, and each of the five key frequencies. Ablating key frequencies causes a performance drop, while the other ablations do not harm performance.

and set all terms besides the constant term and the 10 terms corresponding to $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$ for the five key frequencies to 0. We then measure the loss of the ablated network.

Excluded loss. The excluded loss is opposite of the restricted loss. Instead of keeping the important frequencies w_k, we remove *only* those key frequencies from the logits but keep the rest. Intuitively, this metric tracks how much the model's performance on the *training* data comes from the Fourier multiplication algorithm versus memorization. This exploits the fact that the Fourier multiplication algorithm produces sparse logits in predictable dimensions, while memorization does not (and thus will be mostly unaffected by ablating a few directions). So the ablation will only damage the component of training performance coming from the Fourier multiplication algorithm

Beyond these measures, we will also measure 1) the Gini coefficient (Hurley & Rickard, 2009) of the norms of the Fourier components of W_E and W_L, which measures the sparsity of W_E and W_L in the Fourier basis, and 2) the ℓ_2-norm of the weights during training, since weight decay should push these down once the train loss is near zero.

5.2 Phases of grokking: memorization, circuit formation, and cleanup

We use these progress measures to study how the model evolves throughout training. We use the same mainline model from Section 4 and plot the excluded loss, restricted loss, ratio of test loss to restricted loss, and sum of squared weights in Figure 6. We find that training splits into three partially overlapping phases, which we call the memorization, circuit formation, and cleanup phases. (We show similar results for other models in Appendix C.2.)

Memorization (Epochs 0k–1.4k). We first observe a decline of both excluded and train loss, with test and restricted loss both remaining high and the Gini coefficient is relatively flat. In other words, the model memorizes the data, and the frequencies w_k used by the final model are unused.

Circuit formation (Epochs 1.4k–9.4k). In this phase, excluded loss rises, sum of squared weights falls, restricted loss starts to fall, and test and train loss stay flat. This suggests that the model's behavior on the train set transitions smoothly from the memorizing solution to the Fourier multiplication algorithm. The fall in the sum of squared weights suggests that circuit formation likely happens due to weight decay. Notably, the circuit is formed *well before* grokking occurs.

Cleanup (Epochs 9.4k–14k). In this phase, the restricted loss continues to drop, test loss suddenly drops, sum of squared weights sharply drops, and the ratio of test to restricted loss plateaus then sharply decreases. As the Fourier multiplication circuit both solves the task well and has lower weight than the memorization circuits, weight decay encourages the network to shed the memorized solution in favor of focusing on the Fourier multiplication circuit. This is most cleanly shown in the sharp increase in the Gini coefficient of the Fourier norms of both the embed and unembed matices W_E and W_L, suggesting that the network is becoming sparser in the Fourier basis.

5.3 Grokking and Weight Decay

In the previous section, we saw that each phase of grokking corresponded to an inflection point in the ℓ_2-norm of the weights. This suggests that weight decay is an important component of grokking and drives progress towards the generalizing solution. In this section, we support the hypothesis via ablation—in the absence of weight decay, grokking fails to occur.

In Appendix C.2 and D.1, we provide additional evidence showing that both weight decay and the right amount of data are necessary for grokking. We find that smaller amounts of weight decay causes the network to take significantly longer to grok (echoing the results on toy models from

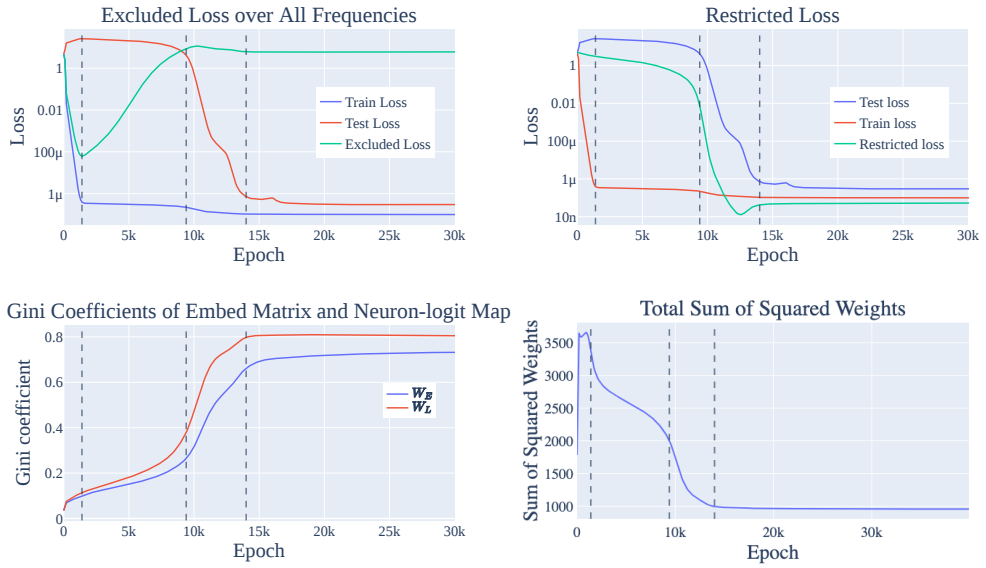

Figure 6: How each of the progress measures in Section 5.1 changes over the course of training. The lines delineate the 3 phases of training: memorization, circuit formation, and cleanup (and a final stable phase). *(Top Left)* Excluded loss increases during circuit formation, while train and test loss remain flat. *(Top Right)* The restricted loss begins declining before test loss declines. *(Bottom Left)* The Gini coefficient of the norms of the Fourier components of W_E and W_L increase sharply during cleanup. *(Bottom Right)* The sums of squared weights decreases smoothly during circuit formation and more sharply during cleanup, indicating that both phases are linked to weight decay.

Liu et al.(2022)) and that our networks do not grok on the modular arithmetic task without weight decay. We also find that when the networks are provided with enough data, there is no longer a gap between the training and test losses (instead, both decline sharply some number of epochs into training). Finally, in Section D.3 we replicate this result on several additional simple algorithmic tasks.

6 Conclusion and Discussion

In this work, we use mechanistic interpretability to define progress measures for small transformers trained on a modular addition task. We find that the transformers embed the input onto rotations in $\mathbb{R}^2$ and compose the rotations using trigonometric identities to compute $a + b \mod 113$. Using our reverse-engineered algorithm, we define two progress measures, along which the network makes continuous progress toward the final algorithm prior to the grokking phase change. We see this work as a proof of concept for using mechanistic interpretability to understand emergent behavior.

In Appendix F, we provide three different scenarios where this approach can be used to understand emergent phenomena in general. Below, we outline areas of future work:

Larger models and realistic tasks. In this work, we studied the behavior of small transformers on a simple algorithmic task, solved with a single circuit. On the other hand, larger models use many more circuits implemented on orders of magnitude more neurons to solve significantly harder tasks.

Automated mechanistic analysis and task-independent progress measures. The analysis reported in this work required significant amounts of manual effort, and our progress metrics are specific to small networks on one particular algorithmic task. Methods for automating the analysis and finding task-independent progress measures seem necessary for this approach to scale to other, larger models.

Discovering phase change thresholds. While the progress measures we defined in Section 5.1 increase relatively smoothly before the phase transition (and suffice to allow us to understand grokking for this task) we lack a notion of criticality that would allow us to predict *when* the phase transition

will happen ex ante. Future work should develop theory and practice in order to apply progress measures to predict the timing of emergent behavior.

7 Reproducibility Statement

An annotated Colab notebook containing the code to replicate our results, including download instructions for model checkpoints, is available at `https://bit.ly/grokking-progress-measures-website`.

References

Boaz Barak, Benjamin L Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. *arXiv preprint arXiv:2207.08799*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 2020. doi: 10.23915/distill.00024. https://distill.pub/2020/circuits.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova Dassarma, Dawn Drain, Nelson Elhage, et al. Predictability and surprise in large generative models. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1747–1764, 2022.

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

Niall Hurley and Scott Rickard. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741, 2009.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *arXiv preprint arXiv:2205.10343*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of chess knowledge in alphazero. *arXiv preprint arXiv:2111.09259*, 2021.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *arXiv preprint arXiv:2202.05262*, 2022.

Beren Millidge. Grokking 'grokking', 2022. URL `https://www.beren.io/2022-01-11-Grokking-Grokking/`.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Jacob Steinhardt. More is different for ai, Feb 2022. URL `https://bounded-regret.ghost.io/more-is-different-for-ai/`.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

A MATHEMATICAL STRUCTURE OF THE TRANSFORMER

We follow the conventions and notation of Elhage et al. (2021) in describing our model. Here, we briefly recap their notation and examine it in our specific case.

We denote our hyperparameters as follows: $d_{vocab} = 113$ is the size of the input and output spaces (treating '=' separately), $d_{model} = 128$ is the width of the residual stream (i.e. embedding size), $d_{head} = 32$ is the size of query, key and value vectors for a single attention head, and $d_{mlp} = 512$ is the number of neurons.

We denote the parameters as follows: W_E (embedding layer); W_{pos} (positional embedding); W_Q^j (queries), W_K^j (keys), W_V^j (values), W_O^j (attention output) (the 4 weight matrices of head j in the attention layer); W_{in} and b_{in} for the input linear map of the MLP layer; W_{out} and b_{out} for the output linear map of the MLP layer; and W_U (unembedding layer). Note that we do not have biases in our embedding, attention layer or unembedding, and we do not tie the matrices for the embedding/unembedding layers.

We now describe the mathematical structure of our network. Note that loss is only calculated from the logits on the final token, and information only moves between tokens during the attention layer, so our variables from the end of the attention layer onwards only refer to the final token. We use t_i to denote the token in position i (as a one-hot encoded vector), p_i to denote the ith positional embedding, $x_i^{(0)}$ to denote the initial residual stream on token with index i, $A^{(i)}$ to denote the attention scores from $=$ to all previous tokens from head i, $x^{(1)}$ to denote the residual stream after the attention layer on the final token, N to denote the neuron activations in the MLP layer on the final token, $x^{(2)}$ the final residual stream on the final token, L the logits on the final token.

The logits are calculated via the following equations:

$$
\begin{aligned}
x_i^{(0)} &= W_E t_i + p_i \\
A^j &= \text{softmax}(x^{(0)T} W_K^{j^T} W_Q^j x_2^{(0)}) \\
x^{(1)} &= [\sum_j W_O^j W_V^j (x^{(0)} \cdot A^j)] + x_2^{(0)} \\
N &= \text{ReLU}(W_{in} x^{(1)}) \\
x^{(2)} &= W_{out} N + x^{(1)} = W_{out} \text{ReLU}(W_{in} x^{(1)}) + x^{(1)} \\
L &= W_U x^{(2)}
\end{aligned}
$$

A.1 EMPIRICAL MODEL SIMPLIFICATIONS

We make two empirical observations:

- The attention paid from '=' to itself is trivial. In practice, the average attention paid is 0.1% to 0.4% for each head, and ablating this does not affect model performance.
- The skip connection around the MLP layer is not important for the model's computation and can be ignored. Concretely, if we set it to zero or to its average (zero or mean ablation) then model accuracy is unchanged, and loss goes from $2.4 \cdot 10^{-7}$ to $9.12 \cdot 10^{-7}$ and $7.25 \cdot 10^{-7}$ respectively. This is a significant increase in loss, but from such a small baseline that we can still ignore it and reverse engineer the model's computation.

A consequence of the first observation is that the attention is now a softmax over 2 elements, i.e. a sigmoid over the difference. And $x_2^{(0)}$ is constant, as it is independent of x and y, and the embedding and positional embedding of '=' are fixed. So $A_0^j = \sigma({x_2^{(0)}}^T W_Q^{j^T} W_K (x_0^{(0)} - x_1^{(0)})$ (and $A_1^j = 1 - A_0^j$)

A consequence of the second observation is that $L \approx W_U W_{out} N$, which we denote as $W_L = W_U W_{out}$. From the perspective of the network, W_L is the meaningful matrix, not either of its constituents, since they compose linearly.

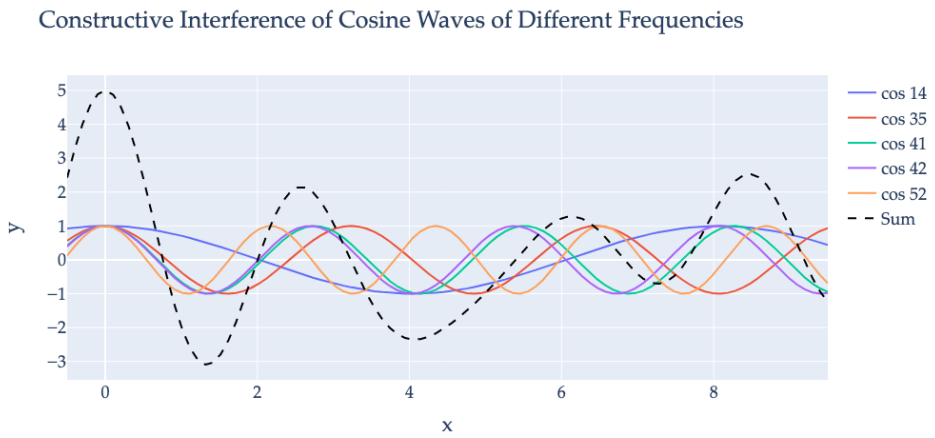

Figure 7: As discussed in Appendix B, while for every $k \in [0, ...P-1]$, $\cos\left(\frac{2k\pi}{P}x\right)$ achieves its maximum value (1) at $x = 0 \mod 113$, it still has additional peaks at different values that are close to the maximum value. However, by adding together cosine waves of the 5 keyfrequencies, the model constructs a periodic function where the value at $x = 0 \mod 113$ is significantly larger than its value anywhere else.

B Why use constructive intereference?

As demonstrated in Section 4 and Appendix C.2.1, small transformers trained on this task use several different frequencies which they add together. The reason for this is to end up with a function whose value at $x = 0 \mod 113$ is significantly larger than any other x.

For example, consider the function $f_{14}(x) = \cos\left(\frac{2\pi \cdot 14}{113}x\right)$. This function has period 113 and is maximized at $x = 0 \mod 113$. However, other values of x cause this function to be close to 1: $f_{14}(8) = f_{14}(105) = 0.998$, $f_{14}(16) = f_{14}(105) = 0.994$, etc.

Now consider $f_{35}(x) = \cos\left(\frac{2\pi \cdot 35}{113}x\right)$. While this function also has period 113 and is maximized at $x = 0 \mod 113$, it turns out that $f_{35}(8) = f_{35}(105) = -0.990$. This means that by adding together f_{14} and f_{35}, we end up with a function that is not close to 1 at $x = 8 \mod 113$. Similarly, while $f_{35}(16) = 0.961$, $f_{52}(16) = -0.56$, and so adding a third frequency reduces the peak at $x = 16 \mod 113$.

We show the constructive interference resulting from the cosine waves for the five frequencies used by the mainline model in Figure 7.

C Supporting evidence for mechanistic analysis of modular arithmetic networks

C.1 Further analysis of the specific training run discussed in the paper

In this section, we provide additional evidence relating to the mainline model.

C.1.1 Periodicity in the activations of other attention heads

In Figure 8 we plot the attention patterns from the final token '=' to the first token a for all 4 attention heads, as a heatmap over the inputs a and b, as this is a scalar for each head. We observe a striking periodicity and further that heads 1 and 3 represent the same frequency while heads 0 and 2 are different.

As shown in Appendix A.1, the attention paid from '=' to itself is negligible, so $A_0^j = 1 - A_1^j$ and it suffices to plot attention to a.

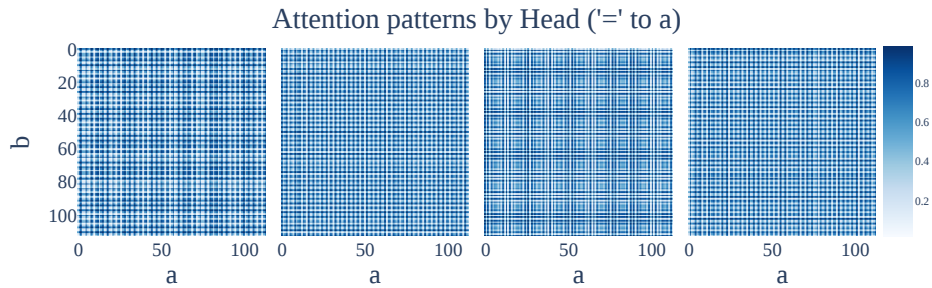

Figure 8: Attention patterns for each head, from the '=' token at the third sequence position to the a token at the first sequence position, as a heatmap over the inputs. All four attention heads exhibit striking periodicity.

Head	k	α^j	β^j	FVE
0	35	-0.26	-0.14	99.03%
1	42	0.27	-0.04	98.49%
2	52	0.29	-0.05	99.07%
3	42	-0.26	0.04	97.91%

Table 3: For each attention head, we show the pattern from '=' to a is well approximated by $0.5 + \alpha(\cos(w_k a) - \cos(w_k b)) + \beta(\sin(w_k a) - \sin(w_k b))$ and give the coefficients and fraction of variance explained for this approximation.

C.1.2 The attention pattern weights are well approximated by differences of sines and cosines of a single frequency.

The periodicity of the attention heads has a striking form—A_0^j is well approximated by $0.5 + \alpha^j(\cos(w_k a) - \cos(w_k b)) + \beta^j(\sin(w_k a) - \sin(w_k b))$, for some frequency w_k and constants α^j and β^j (which may differ for each head). Note further that this simplifies to $0.5 + \gamma(\cos(w_k(a + \theta)) - \cos(w_k(b + \theta)))$ for some constants γ and θ. We show the coefficients and fraction of variance explained in Table 1

Mechanistic Analysis of Attention Patterns. We can further mechanistically analyse *how* the model achieves this form. The following is a high-level sketch of what is going on:

First, note that the attention score on position 0 and head j is just a lookup table on the input token a (of size P). To see why, note that $A_0^j = m x_0^{(0)T} W_K^{j\,T} W_Q^j x_2^{(0)}$. $x_2^{(0)}$ is constant since the token is always '=' and $x_0^{(0)} = W_E t_0 + p_0$. So this reduces to $t_0 \cdot C_j + D$ for some constant vector $C_j = W_E^T W_K^{j\,T} W_Q^j x_2^{(0)} \in \mathbb{R}^p$ and some scalar $D = p_0^T W_K^{j\,T} W_Q^j x_2^{(0)}$. As t_0 is one-hot encoded, this is just a lookup table, which we may instead denote as $C_j[a]$

Next, note that the attention pattern from $=\rightarrow 0$ is $\sigma(C_j[a] - C_j[b])$. As argued in Appendix A.1, the attention paid $=\rightarrow=$ is negligible and can be ignored. So the softmax reduces to a softmax over two elements, which is a sigmoid on their difference. As form of C_j does not mention the token index or value, it is the same for position 0 and 1.

We now show that C_j is well-approximated by a wave of frequency w_{k_j} for some integer k_j. That is, $C_j[a] \approx F_j \cos(w_{k_j} a) + G_j \sin(w_{k_j} a)$. We do this by simply computing C_j and fitting the constants F_j and G_j to minimize ℓ_2 loss, and display the resulting coefficients for each head in Figure 9. This fit explain 99.02%, 95.21%, 99.10%, 92.42% of the variance of C_j respectively.

For each head j, $\sigma(C_j[a] - C_j[b]) \approx 0.5 + E_j(C_j[a] - C_j[b])$ for some constant E_j—that is, the sigmoid has some linear approximationm. It is always the case that we can take a linear approximation to scalar function, by a 1D Linear Regression. The intercept will be 0.5 by symmetry. The striking thing is that, because the inputs to the sigmoid for the attention heads are over a fairly wide range ($[-5, 5]$ roughly), the linear approximation to the sigmoid is a fairly good fit, explaining 97.5% of the variance.

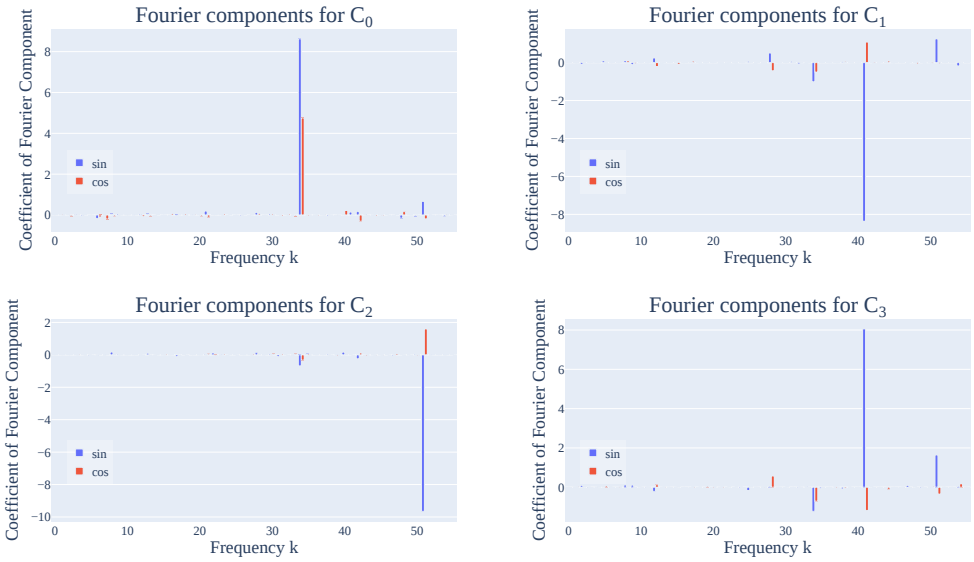

Figure 9: We plot the attention pattern weights, C_j in the Fourier basis for each of the four heads $j \in \{0, 1, 2, 3\}$. We observe significant sparsity, with almost all of each term being associated with a single frequency.

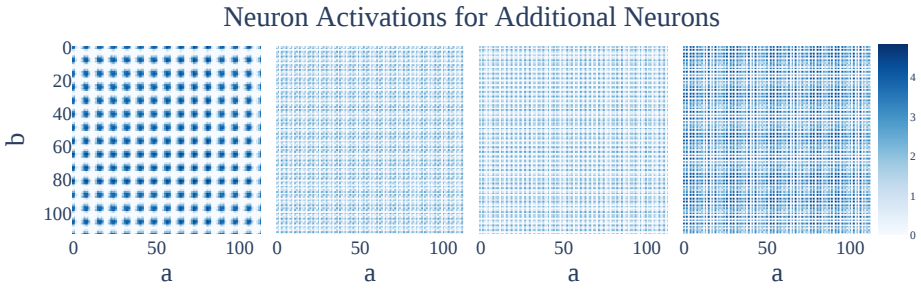

Figure 10: Plots of neuron activations for MLP neurons 1, 2, 3 and 4, for inputs $a, b \in \{0, 1, ..., 112\}$. As with Neuron 0, all of the activation patterns are periodic in both inputs.

We validate that this is all that is going on, by replacing the sigmoid with the best linear fit. This *improves* loss from $2.41 \cdot 10^{-7}$ to $2.12 \cdot 10^{-7}$.

Bringing it all back together, we see that the attention patterns of each head will be well approximated by $0.5 \pm C_j(\cos(w_{k_j}(a+\theta_j)) - \cos(w_{k_j}(b+\theta_j)))$ - the softmax is linear, with an intercept of 0.5, and the weights C_j map each token to a score that is a wave in a single frequency. This exactly gives us the periodic form shown in Figure 8.

C.1.3 Periodicity in the activations of additional neurons

In Figure 10, we display the activations of four more MLP neurons, as a function of the inputs. As with neuron 0, the activations of these neurons are also periodic in the inputs.

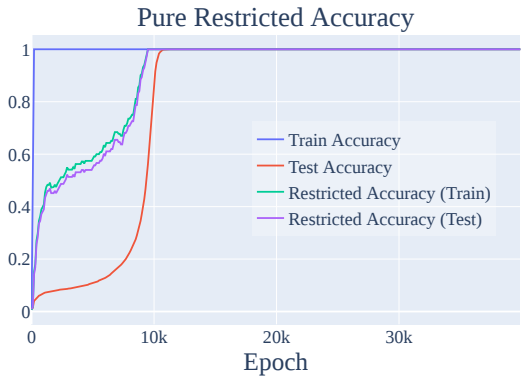

Figure 11: Accuracy when restricting Fourier Components to the five key frequencies. As with restricted loss, this shows that the model figures out how to generalize modulo deleting noise before it removes the noise.

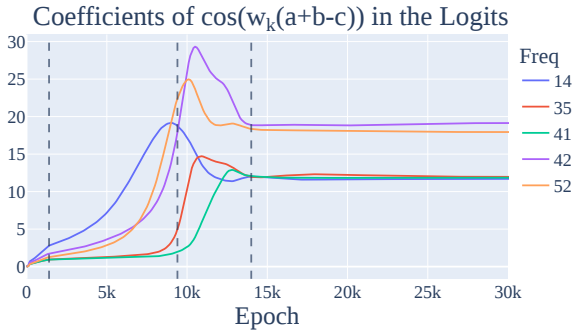

Figure 12: The coefficients of $\cos(w(a+b-c))$ in the logits over the model's training. As with the metrics in the paper, this shows a nice interpolation and growth of each cosine term.

C.1.4 Additional grokking figures for mainline run

In Figure 11, we display the accuracy of the model when restricting the model to use only the five key frequencies. As with restricted loss, this *improves* model performance during training.

In Figure 12, we show the coefficients of the five key frequencies in the logits, calculated by regressing the logits against the five $\cos(w_k(a+b-c))$ terms.

In Figure 13, we plot the excluded loss if we exclude each of the five key frequencies (as opposed to all five key frequencies).

All three of these figures have inflection points corresponding to the relevant phases of grokking, discussed in Section 5.1.

C.2 Additional results from different runs

In this section, we plot relevant figures from other runs, either with the same architecture (Appendix C.2.1) or with different architectures or experimental setups (Appendix C.2.2). Note that in general, while all models learn to use variants of the modular arithmetic algorithm, they use a varying number of *different* key frequencies. In order to find the key frequencies to calculate the excluded and

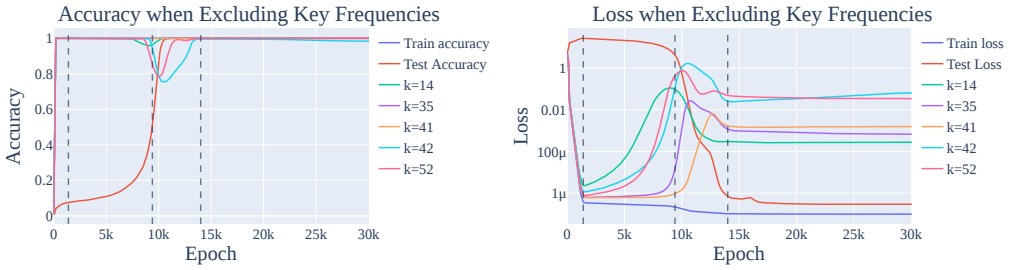

Figure 13: The excluded accuracy (left) and loss (right) if we exclude each of the five key frequencies for our mainline model. As with the excluded loss results in Section 5.1, this shows that the model interpolates between memorising and generalising.

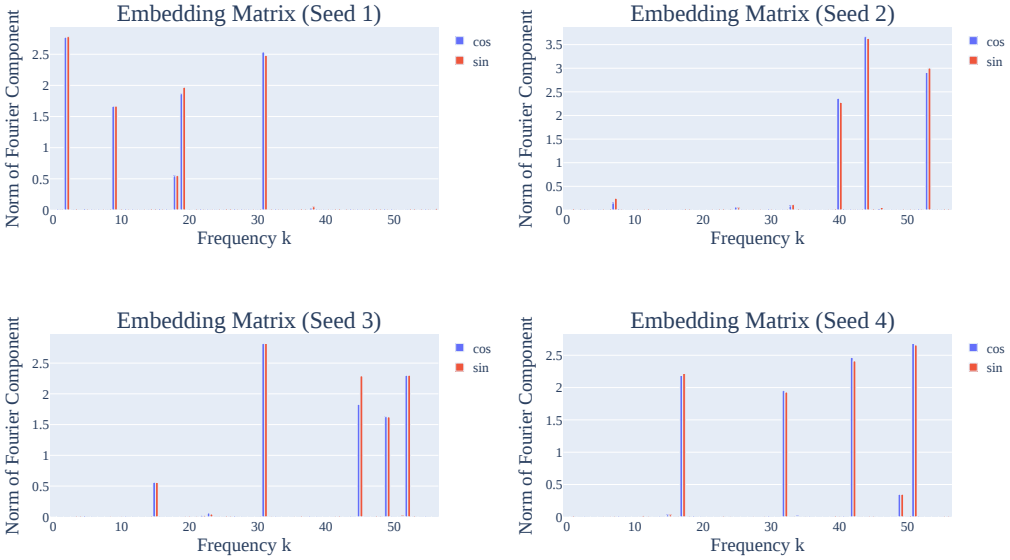

Figure 14: The norms of the Fourier components in the embedding matrix W_E for each of four other random seeds for the original (1 layer) architecture. As discussed in Section 4.1 and Appendix C.2.1, the sparsity of W_E in the Fourier basis is evidence that the network is operating in a Fourier basis.

restricted loss, we perform a DFT on the neuron-logit map W_L, then take the frequencies with nontrivial coefficients.[5]

C.2.1 Additional results for different runs with the same architecture

In this section, we provide evidence that all 4 other runs using the experimental setup of our mainline model also use the fourier multiplication algorithm, and then confirm that the same phases of grokking also occur on these runs.

Confirming that the other seeds use the Fourier Multiplication Algorithm. In Figure 14, we show the norms of the Fourier components of the embedding matrix W_E for each of the 4 other random seeds. As with the mainline model, the matrices are sparse in the Fourier basis. In Figure 15, we show the norms of the Fourier components of the neuron-logit map W_L for the 4 other random

[5]One method for getting a general (model-independent) progress measure for this task is to compute the excluded loss for each of the 56 unique frequencies then take the max. We omit the plots for this variant of the excluded loss as they are broadly similar.

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_2 c)$	$147.4\cos(w_2 a)\cos(w_2 b) - 145.8\sin(w_2 a)\sin(w_2 b) \approx 146.6\cos(w_2(a+b))$	99.2%
$\sin(w_2 c)$	$145.5\cos(w_2 a)\sin(w_2 b) + 145.6\sin(w_2 a)\cos(w_2 b) \approx 145.5\sin(w_2(a+b))$	99.1%
$\cos(w_9 c)$	$49.3\cos(w_9 a)\cos(w_9 b) - 48.0\sin(w_9 a)\sin(w_9 b) \approx 48.6\cos(w_9(a+b))$	96.4%
$\sin(w_9 c)$	$48.6\cos(w_9 a)\sin(w_9 b) + 48.5\sin(w_9 a)\cos(w_9 b) \approx 48.5\sin(w_9(a+b))$	96.7%
$\cos(w_{19} c)$	$58.0\cos(w_{19} a)\cos(w_{19} b) - 58.3\sin(w_{19} a)\sin(w_{19} b) \approx 58.2\cos(w_{19}(a+b))$	95.4%
$\sin(w_{19} c)$	$59.3\cos(w_{19} a)\sin(w_{19} b) + 59.4\sin(w_{19} a)\cos(w_{19} b) \approx 59.4\sin(w_{19}(a+b))$	93.9%
$\cos(w_{31} c)$	$94.4\cos(w_{31} a)\cos(w_{31} b) - 96.4\sin(w_{31} a)\sin(w_{31} b) \approx 95.4\cos(w_{31}(a+b))$	98.4%
$\sin(w_{31} c)$	$97.2\cos(w_{31} a)\sin(w_{31} b) + 97.1\sin(w_{31} a)\cos(w_{31} b) \approx 97.2\sin(w_{31}(a+b))$	98.7%

(a) Seed 1

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_{40} c)$	$97.0\cos(w_{40} a)\cos(w_{40} b) - 99.4\sin(w_{40} a)\sin(w_{40} b) \approx 98.2\cos(w_{40}(a+b))$	97.3%
$\sin(w_{40} c)$	$81.3\cos(w_{40} a)\sin(w_{40} b) + 81.3\sin(w_{40} a)\cos(w_{40} b) \approx 81.3\sin(w_{40}(a+b))$	92.7%
$\cos(w_{44} c)$	$309.1\cos(w_{44} a)\cos(w_{44} b) - 338.7\sin(w_{44} a)\sin(w_{44} b) \approx 323.9\cos(w_{44}(a+b))$	98.5%
$\sin(w_{44} c)$	$327.3\cos(w_{44} a)\sin(w_{44} b) + 327.2\sin(w_{44} a)\cos(w_{44} b) \approx 327.3\sin(w_{44}(a+b))$	98.9%
$\cos(w_{53} c)$	$192.1\cos(w_{53} a)\cos(w_{53} b) - 192.2\sin(w_{53} a)\sin(w_{53} b) \approx 192.1\cos(w_{53}(a+b))$	97.3%
$\sin(w_{53} c)$	$166.7\cos(w_{53} a)\sin(w_{53} b) + 166.8\sin(w_{53} a)\cos(w_{53} b) \approx 166.8\sin(w_{53}(a+b))$	95.7%

(b) Seed 2

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_{31} c)$	$156.1\cos(w_{31} a)\cos(w_{31} b) - 156.5\sin(w_{31} a)\sin(w_{31} b) \approx 156.3\cos(w_{31}(a+b))$	99.3%
$\sin(w_{31} c)$	$150.7\cos(w_{31} a)\sin(w_{31} b) + 150.7\sin(w_{31} a)\cos(w_{31} b) \approx 150.7\sin(w_{31}(a+b))$	98.9%
$\cos(w_{45} c)$	$72.5\cos(w_{45} a)\cos(w_{45} b) - 76.8\sin(w_{45} a)\sin(w_{45} b) \approx 74.6\cos(w_{45}(a+b))$	95.9%
$\sin(w_{45} c)$	$74.7\cos(w_{45} a)\sin(w_{45} b) + 74.6\sin(w_{45} a)\cos(w_{45} b) \approx 74.6\sin(w_{45}(a+b))$	96.6%
$\cos(w_{49} c)$	$45.9\cos(w_{49} a)\cos(w_{49} b) - 45.5\sin(w_{49} a)\sin(w_{49} b) \approx 45.7\cos(w_{49}(a+b))$	97.0%
$\sin(w_{49} c)$	$45.8\cos(w_{49} a)\sin(w_{49} b) + 45.8\sin(w_{49} a)\cos(w_{49} b) \approx 45.8\sin(w_{49}(a+b))$	96.9%
$\cos(w_{52} c)$	$71.6\cos(w_{52} a)\cos(w_{52} b) - 72.1\sin(w_{52} a)\sin(w_{52} b) \approx 71.9\cos(w_{52}(a+b))$	98.5%
$\sin(w_{52} c)$	$68.7\cos(w_{52} a)\sin(w_{52} b) + 68.7\sin(w_{52} a)\cos(w_{52} b) \approx 68.7\sin(w_{52}(a+b))$	97.9%

(c) Seed 3

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_{17} c)$	$66.0\cos(w_{17} a)\cos(w_{17} b) - 63.5\sin(w_{17} a)\sin(w_{17} b) \approx 64.8\cos(w_{17}(a+b))$	96.4%
$\sin(w_{17} c)$	$66.4\cos(w_{17} a)\sin(w_{17} b) + 66.4\sin(w_{17} a)\cos(w_{17} b) \approx 66.4\sin(w_{17}(a+b))$	94.9%
$\cos(w_{32} c)$	$68.7\cos(w_{32} a)\cos(w_{32} b) - 68.4\sin(w_{32} a)\sin(w_{32} b) \approx 68.5\cos(w_{32}(a+b))$	96.2%
$\sin(w_{32} c)$	$68.0\cos(w_{32} a)\sin(w_{32} b) + 68.0\sin(w_{32} a)\cos(w_{32} b) \approx 68.0\sin(w_{32}(a+b))$	96.3%
$\cos(w_{42} c)$	$100.4\cos(w_{42} a)\cos(w_{42} b) - 96.0\sin(w_{42} a)\sin(w_{42} b) \approx 98.2\cos(w_{42}(a+b))$	97.9%
$\sin(w_{42} c)$	$100.2\cos(w_{42} a)\sin(w_{42} b) + 100.1\sin(w_{42} a)\cos(w_{42} b) \approx 100.1\sin(w_{42}(a+b))$	98.6%
$\cos(w_{51} c)$	$118.0\cos(w_{51} a)\cos(w_{51} b) - 116.2\sin(w_{51} a)\sin(w_{51} b) \approx 117.1\cos(w_{51}(a+b))$	99.0%
$\sin(w_{51} c)$	$114.3\cos(w_{51} a)\sin(w_{51} b) + 114.2\sin(w_{51} a)\cos(w_{51} b) \approx 114.2\sin(w_{51}(a+b))$	98.5%

(d) Seed 4

Table 4: For each of the directions in the neuron-logit map W_L of the final models from 4 other random seeds (Appendix C.2.1), we project the MLP activations in that direction then perform a Fourier transform. For brevity, we omit terms with coefficients less than 15% of the largest coefficient. We then compute the fraction of variance explained (FVE) if we replace the projection with a multiple of a single term of the form $\cos(w_k(a+b))$ or $\sin(w_k(a+b))$, and find that this is consistently close to 1.

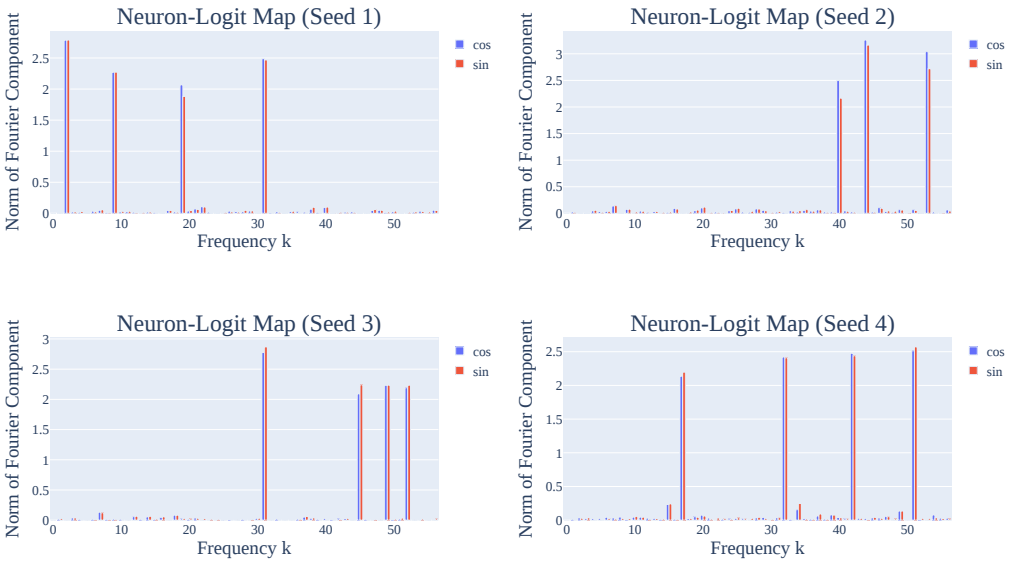

Figure 15: The norms of the direction corresponding to sine and cosine waves in the neuron-logit map weights W_L. As with the mainline model discussed in the main body and discussed in Appendix C.2.1, W_L is consistently sparse, providing is evidence that all four are operating in a Fourier basis.

Seed	Test Loss	Excluded Loss (Key frequencies removed)	Restricted Loss (Other frequencies removed)
1	$2.07 \cdot 10^{-7}$	$6.5 \cdot 10^{0}$	$5.7 \cdot 10^{-8}$
2	$2.1 \cdot 10^{-7}$	$1.1 \cdot 10^{1}$	$6.2 \cdot 10^{-8}$
3	$2.05 \cdot 10^{-7}$	$6.7 \cdot 10^{0}$	$5.5 \cdot 10^{-8}$
4	$2.33 \cdot 10^{-7}$	$6.8 \cdot 10^{0}$	$6.0 \cdot 10^{-8}$

Table 5: As discussed in Appendix C.2.1, ablating the key frequencies for each of the networks reduces performance to worse than chance, while ablating all other frequencies *improves* performance.

seeds. The matrices are sparse in the Fourier basis, enabling us to identify 3 or 4 key frequencies for each of the seeds. Again, note that these are different frequencies per seed.

Using the key frequencies identified in the neuron-logit map, we repeat the experiment in Section 4.2, where we "read off" the MLP activations in the 6 or 8 directions corresponding to the key frequencies. As with our mainline model, this lets us identify the trigonometric identities for $\cos(w_k(a+b))$ and $\sin(w_k(a+b))$ being computed at the MLP layer. We confirm that the trigonometric identities are a good approximation by approximating the activations with a single term of the form $\cos(w_k(a+b))$ or $\sin(w_k(a+b))$—as with the mainline model, the fraction of variance explained is consistently close to 100%.

Finally, we ablate the key frequencies from the logits as in Section 4.4 and report the results in Table 5. As with the mainline model, ablating all of the key frequencies reduces performance to worse than chance, while ablating everything *but* the key frequencies *improves* test performance.

Progress measures and grokking. Finally, we confirm the progress measure and grokking results from the mainline model on other runs with the same architecture. In Figure 16, we display the train, test, and restricted loss for each of the four other random seeds. In Figure 17, we display the Gini coefficients of the Fourier components of the embedding matrix W_E and the neuron-logit map W_L for each of the four other random seeds. The shape of the curves are very similar to those of the mainline model, allowing us to divide grokking on these models into the same three phases identified in the main text. Interestingly, while all of the models complete memorization by around 1400 epochs, circuit formation and cleanup occur at different times.

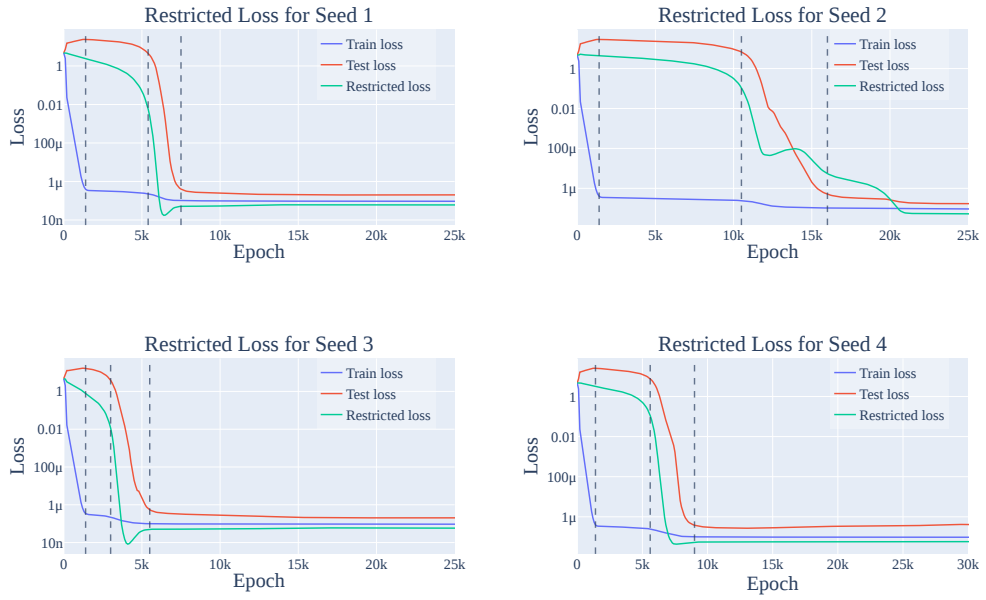

Figure 16: The train, test, and restricted loss for each of the four other random seeds described in Appendix C.2.1. The lines delineate the 3 phases of training: memorization, circuit formation, and cleanup (and a final stable phase). As with the mainline model, restricted loss consistently declines prior to train loss. Note that while the shapes of the loss curves are similar to each other and those of the mainline model, the exact time that grokking occurs (and thus the dividers between the phases of grokking) differ by random seed. Interestingly, memorization is complete by around 1400 steps for all five runs.

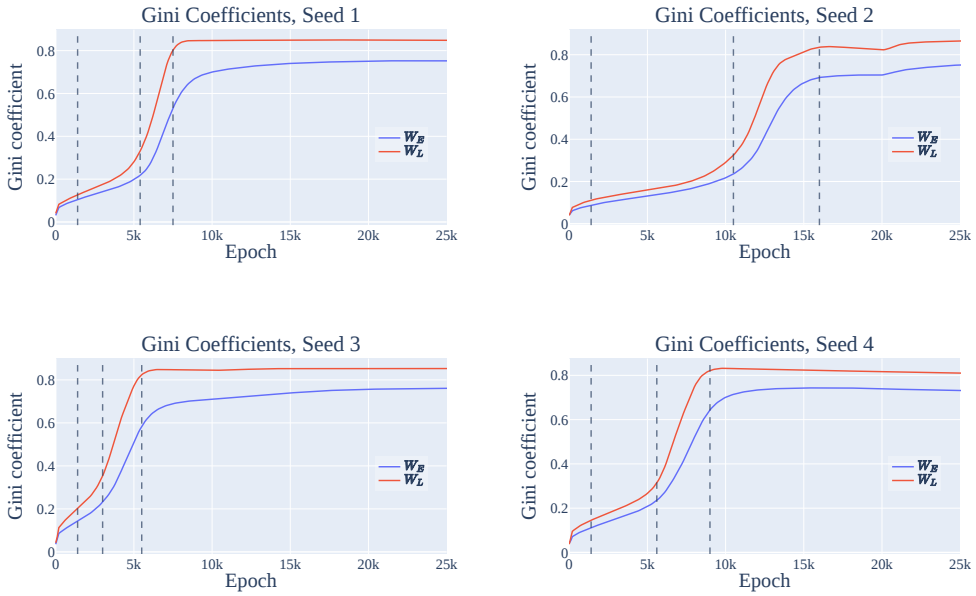

Figure 17: The Gini coefficients (a measure of sparsity) of the Fourier components of the embedding matrix W_E and the neuron-logit map W_L for each of the four other random seeds. The lines delineate the 3 phases of training: memorization, circuit formation, and cleanup (and a final stable phase). As with the mainline model, sparsity increases slowly during memorization and circuit formation, and then quickly during cleanup.

C.2.2 Results for other experimental setups

In this section, we provide further evidence that small transformers grok on the modular addition task, by varying the size of the network, the amount of training data, and the size of the prime P.

1-Layer Transformers with Varying Fractions of Training Data. We find that grokking occurs for the modular addition task with $P = 113$ for many data fractions (that is, the fraction of the $113 \cdot 113$ pairs of inputs that the model sees during training), as shown in Figure 18. Smaller amount lead to slower grokking, but sufficiently large fractions of data ($\geq 85\%$) lead to immediate generalization, as shown in Figure 18.

As with the results in Appendix C.2.1, all of the 1-layer transformers in this section also converge to using the Fourier multiplication algorithm.

2-Layer Transformers. As shown in Figure 20, 2-layer transformers also exhibit some degree of grokking. However, this is complicated by the slingshot mechanism (Thilak et al., 2022). We display the excluded loss of a 2-layer transformer in Figure 21 and find it shows a similar pattern to the mainline 1-layer transformer, in that it improves relatively smoothly *before* grokking occurs.

Smaller and larger primes. We also examined smaller and larger prime moduli. For $P = 53$ (Figure 22), we explored a variety of weight decays to observe grokking in the small prime case. With the original weight decay setting of $\lambda = 1$, we found that the models never generalized. However, increasing the weight decay to $\lambda = 5$ does allow the model to grok. We speculate that this is because the memorization solution is significantly smaller (since there are only $53 \cdot 53$ total pairs), thereby requiring more aggressive weight decay for the generalizing solution to be favored.

For $P = 401$ (Figure 23), we could not get grokking, even by varying the weight decay parameter. Instead, the model immediately learns the generalizing solution. We believe this is because the amount of data seen by the model is greatly increased compared to the $P = 113$ case (from 30% of $113 \cdot 113$ pairs to 30% of $401 \cdot 401$ pairs), thereby favoring the generalizing solution from the start.

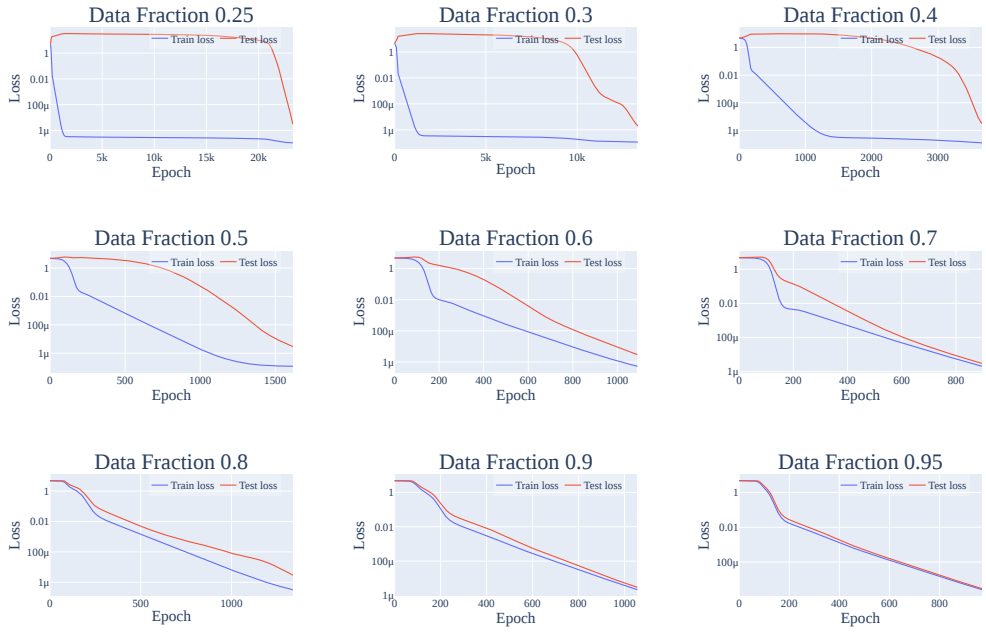

Figure 18: Training and test losses for a 1-layer transformer on the modular addition task with $P = 113$, with varying fractions of the $113 \cdot 113$ pairs of possible inputs used in training. Grokking occurs when $< 70\%$ of the dataset is used during training, and is very noticeable when $< 50\%$ of the dataset is used. Lower fractions of data lead to slower grokking. Note that we've scaled the x-axes for more detail; all 9 runs shown here achieve $< 10^{-6}$ train loss by step 1400.

Figure 19: Number of steps for train/test loss to be < 0.01, as a function of the amount of training data. While train loss immediately converges to a low number, grokking takes significantly longer with lower fractions of data.

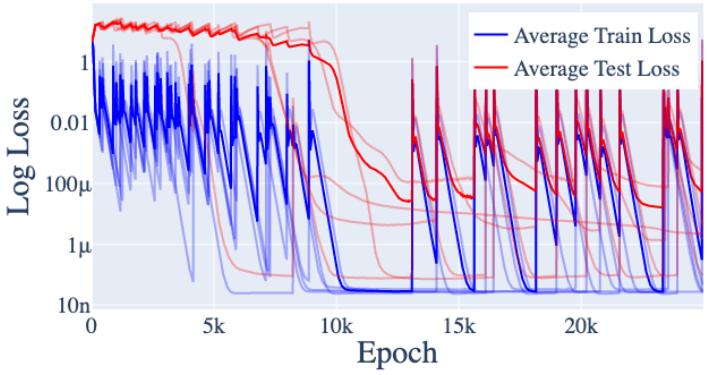

Figure 20: Training and test loss for a 2-layer version of the original architecture. Average across 5 random seeds is in bold.

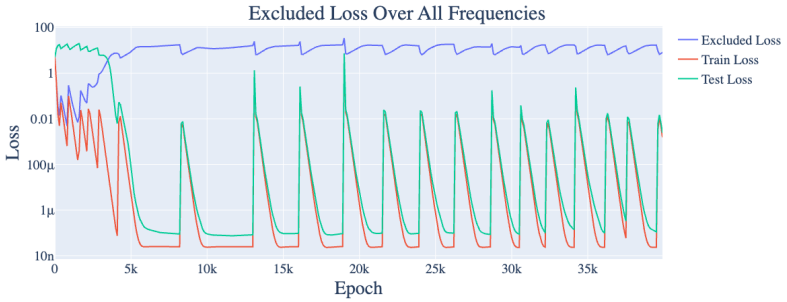

Figure 21: Training, test, and full excluded loss for a 2-layer version of the original architecture. One random seed chosen for readability.

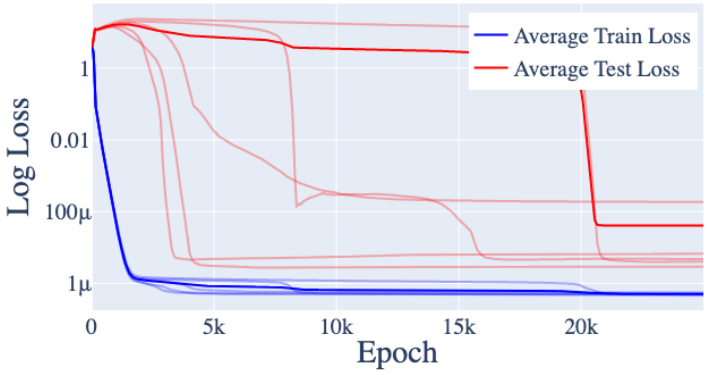

Figure 22: The training and test losses for $P = 53$ and all other hyperparameters except weight decay ($\gamma = 5$) the same as the main training run discussed in the paper. The averages are bold, and all contributing runs are partially transparent. Note that grokking occurs.

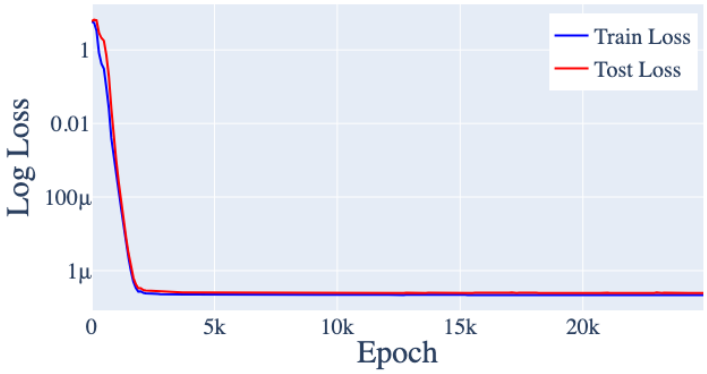

Figure 23: The training and test losses for $P = 401$ and all other hyperparameters the same as the main training run discussed in the paper. Grokking doesn't occur, even across a variety of weight decays.

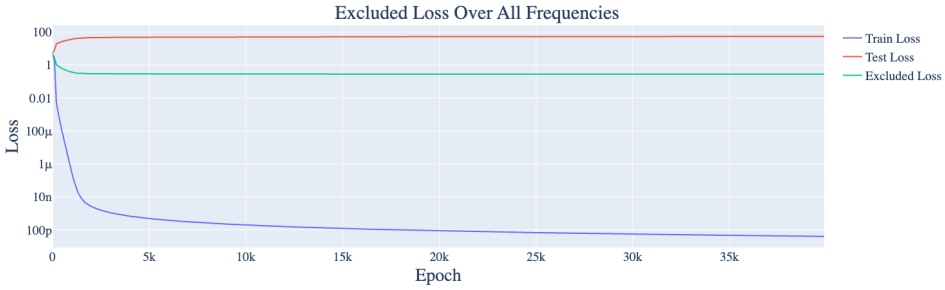

Figure 24: Training, test, and full excluded loss for a 1-layer version of the original architecture without weight decay. One random seed chosen for readability. Note that not having weight decay prevents grokking.

D Additional results on grokking

D.1 Both regularization and limited data are necessary for grokking

As discussed in Section 6 and Appendix C.2, the weight decay and the amount of data seem to have a strong effect on whether grokking occurs. To confirm this, we experiment with removing weight decay and varying the amount of data on 1-layer transformers. In Figure 24, we give the training, test, and full excluded loss for a typical training run with $\lambda = 0$ (no weight decay). As the figure shows, no grokking occurs, and excluded loss does not increase, suggesting that the model does not form the circuit for generalizing algorithm at all.

In Figure 18, we show the test loss curves for models trained with weight decay $\lambda = 1$ and on various fractions of the data. Though all the train losses are approximately the same—that is, they memorize at the same rate, models trained on smaller fractions of data take longer to grok.

In Figure 25, we display the test and train loss of models trained with $\lambda = 0.3$ and $\lambda = 3.0$. Smaller amounts of weight decay lead to slower grokking, while larger amounts of weight decay lead to faster grokking—on average, it takes around 3k epochs for models to grok with weight decay $\lambda = 0.3$, 5-10k epochs for the models to grok with weight decay $\lambda = 1.0$, and 20k epochs for the models to grok with weight decay $\lambda = 3.0$.

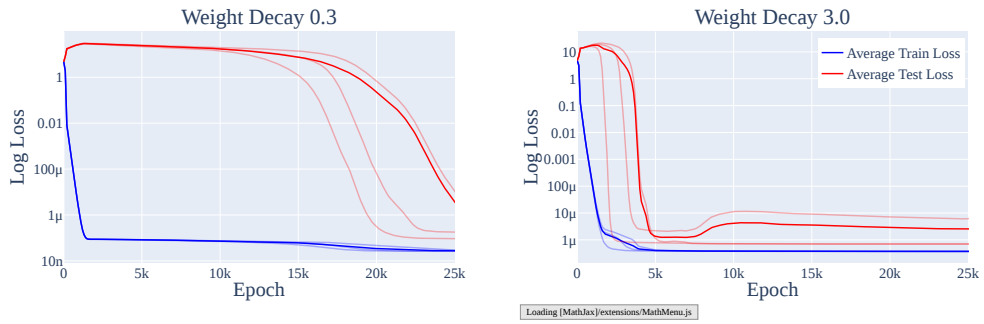

Figure 25: The train and test loss over the course of training with weight decay $\lambda = 0.3$ (left) and $\lambda = 3.0$ (right). Less aggressive weight decay leads to slower grokking.

Finally, we test whether other forms of regularization can also induce grokking. We replaced weight decay with the following types of regularization while keeping all other hyperpameters the same:

1. **Dropout** We add dropout Srivastava et al. (2014) to the MLP neurons, with $p \in \{0.2, 0.5, 0.8\}$. That is, for each individual neuron, we set it to 0 with probability p during training, and also multiply the outputs of the other neurons by $\frac{1}{1-p}$.

2. ℓ_1 **Regularization** We add an ℓ_1 penalty to the loss term. We use $\lambda \in \{1, 10, 100\}$. Note that we *do not decouple* the updates with respect to the ℓ_1 penalty from optimization steps done with respect to the log loss (as is done for ℓ_2 regularization via AdamW Loshchilov & Hutter (2017)).

In each case, we ran three random seeds. We show the results in Figure 26. While grokking did not occur with ℓ_1 regularization, we found that it does occur for two out of the three seeds using dropout with $p = 0.2$ or $p = 0.5$. We speculate that this is because both dropout and weight decay encourage the network to spread out computation (which is required for the Fourier multiplication algorithm), while ℓ_1 regularization encourages the network to become more sparse in the neuron basis and thus *less* sparse in the Fourier basis, preventing the network from learning the Fourier Multiplication Algorithm.

D.2 The slingshot mechanism often occurs, but is unnecessary for grokking

As noted in Section C.2, our 2-layer transformers exhibit significant slingshots (Thilak et al., 2022) during training. We speculate that this is due to how gradients of different scale interact with adaptive optimizers. We were even able to induce slingshots on a 1-layer by reducing the precision of the loss calculations (as this causes many gradients to round to 0 and thus greatly increases the differences in scale of gradients).

However, as many of our 1-layer models do not exhibit slingshots but nonetheless grok, the slingshot mechanism is unnecessary for grokking to occur, in the presence of weight decay. We speculate that the slingshots of Thilak et al. (2022) (which co-occur with grokking for training runs *without* weight decay) serve as an implicit regularization mechanism that favors the simpler, generalizing solution over the more complicated

D.3 Additional evidence from other algorithmic tasks

We now provide addition analysis of grokking phenomena on 3 additional algorithmic tasks and confirm that limited data is an important part of grokking:

1. **5 digit addition.** We sample pairs of random 5 digit numbers and have the model predict their sum
2. **Predicting repeated subsequences.** We take a uniform random sequence of tokens, randomly choose a subsequence to repeat, and train the model to predict the repeated tokens.

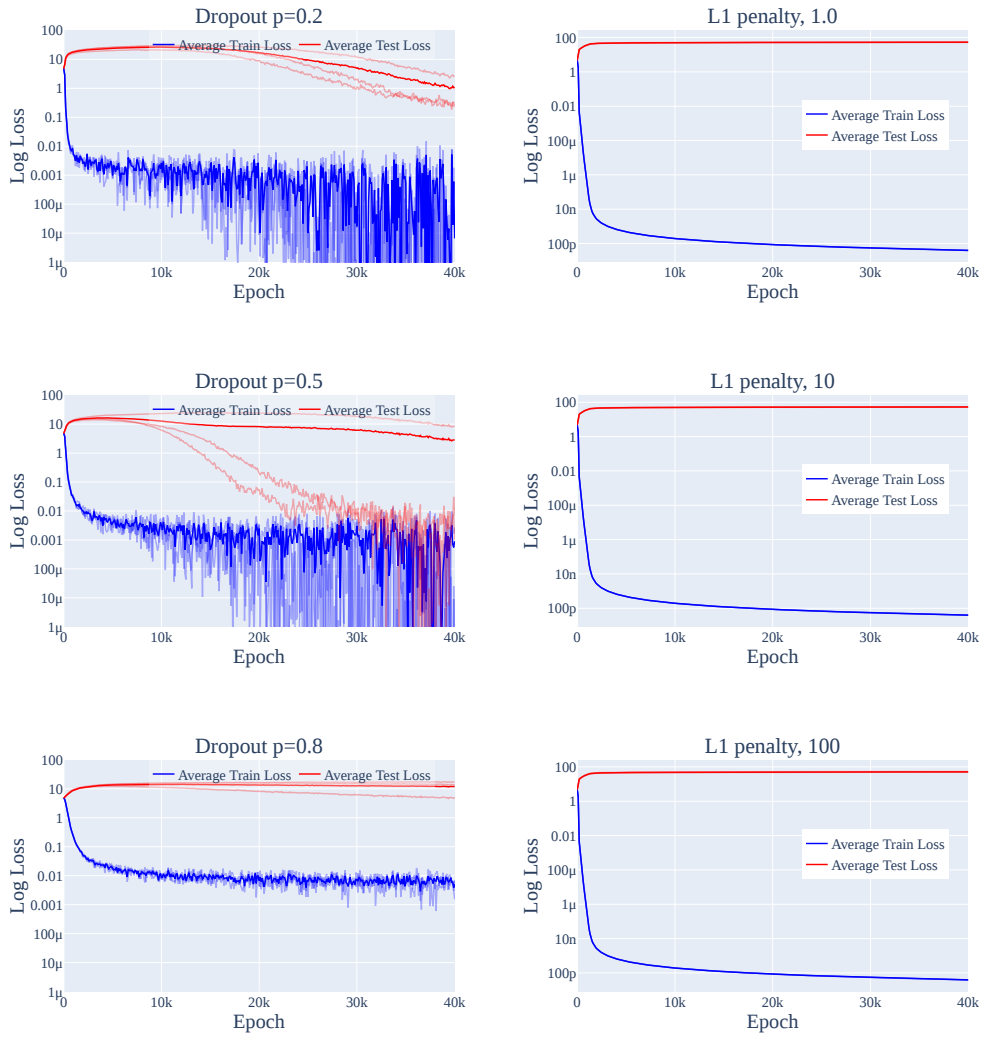

Figure 26: The train and test loss over the course of training with two types of regularization, dropout and ℓ_1 regularization.

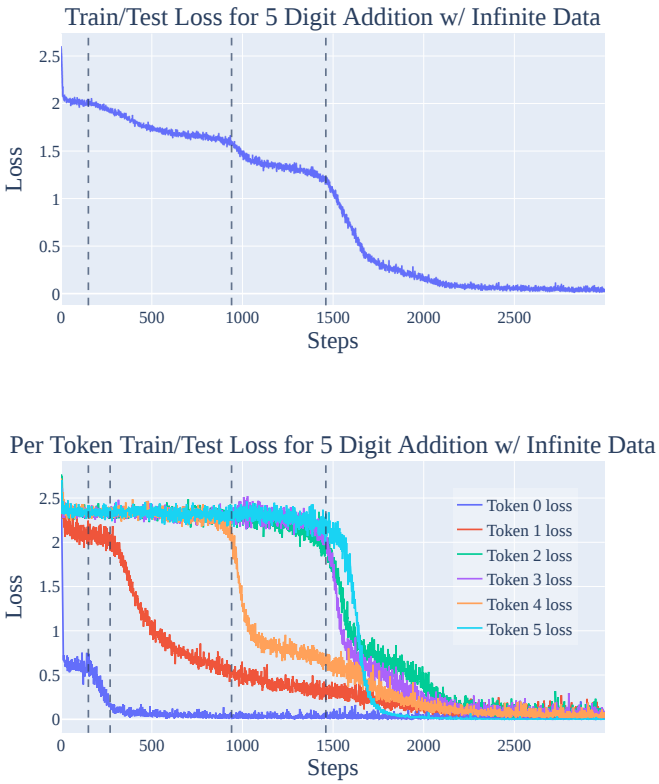

Figure 27: (*Top*) The training/test loss for 5 Digit Addition trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs.(*Bottom*) The train/test loss *per token* for 5 Digit Addition, trained with randomly generated data at each step. Note that phase changes in the average loss correspond to phase changes in individual tokens, though one phase change (token 1, around step 270) is not visible on the averaged loss as it overlaps with the end of the first phase change (token 0, starting around step 150).

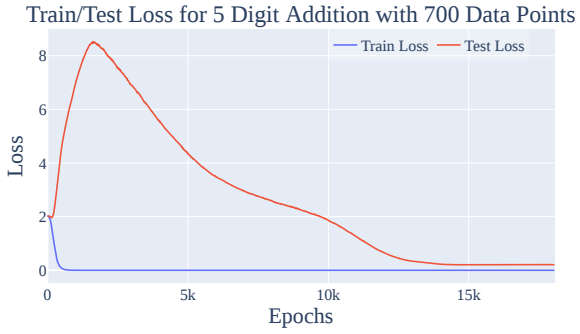

Figure 28: The train and test loss for 5 Digit Addition trained on 700 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.

3. **Skip trigram.** We feed in a sequence of tokens from 0 to 19, of which exactly one is greater than or equal to 10, and the model needs to output the token that is ≥ 10. This can be solved with learning 10 skip trigrams.

We use a 1-layer full transformer for 5-digit addition, a 2-layer attention only transformer for predicting repeated subsequences, and a 1-layer attention only transformer for the skip trigram task. Otherwise, we use the same hyperparameters as in the mainline model.

5 Digit Addition We first consider the case where we train on the approximately infinite data regime. For each minibatch, we randomly new sample 5 digit numbers. We report the results in Figure 27. Train loss coincides with test loss, so grokking does not occur, as the model almost never sees the same pair of 5 digit numbers twice, with 10^{10} such pairs. Interestingly, the various small bumps in Figure 27 correspond to the model learning how to calculate each of the 6 tokens in the output. However, grokking does occur when we restrict the model to only see 700 data points, as shown in Figure 28.

Repeated subsequence As with the 5-digit addition task, we find that restricting the amount of data is necessary and sufficient for grokking on the repeated subsequence task. In Figure **??**, the model sees new data at every step exhibits no grokking. In contrast, clear grokking occurs when we restrict the model to only see 512 data points in Figure 30.

Skip trigram As with the previous tasks, we find that restricting the amount of data is necessary and sufficient for grokking on the skip trigram task. The model that sees new data at every step exhibits no grokking in Figure 31. Meanwhile, the model restricted to only see 512 data points exhibits clear grokking in Figure 32.

Taken together, these results echo the importance of limited data for grokking.

E Further speculations on grokking

E.1 An intuitive explanation of grokking

In this section, we speculate on what might be happening "under the hood" when a model groks and explore why this phenomena happens. The evidence is only suggestive, so this a promising direction for future research.

Grokking occurs when models, trained on algorithmic tasks with certain hyperparameters, initially overfit the training data where train loss significantly improves while test loss worsens and the two diverge. But later in training, there is a sudden improvement in test loss, so test and train loss converge. In contrast to Power et al. (2022) but in line with Liu et al. (2022), grokking does *not*

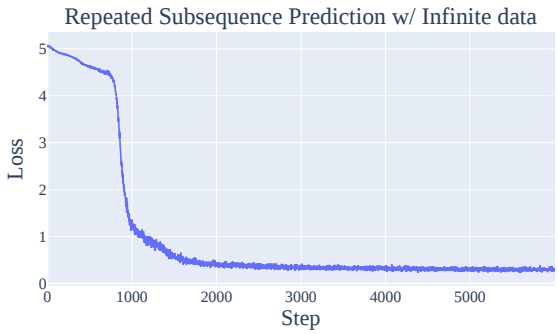

Figure 29: The training/test loss for repeated subsequences trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs. There sharp phase change corresponds to the model forming induction heads. (Olsson et al., 2022)

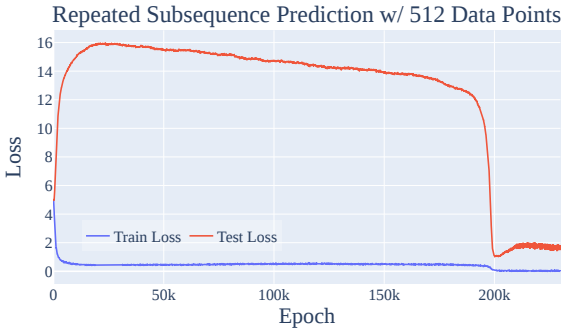

Figure 30: The train and test loss for the repeated subsequence task, trained on 512 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.

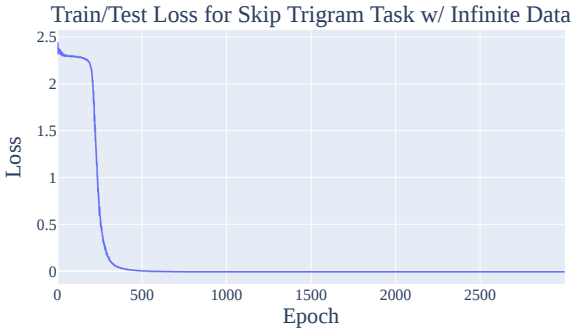

Figure 31: The training/test loss for the skip trigram task, trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs. The sharp phase change corresponds to the network learning all of the skip trigrams.

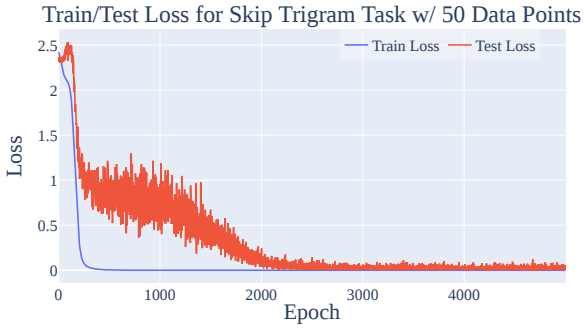

Figure 32: The train and test loss for the skip trigram task, trained on 512 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.

occur when both train and test loss improve together without the initial divergence, as shown in many of the figures in this paper, for example Figures 2 and 16.

The core issue is that the model has two possible solutions: memorization (with low train loss and high test loss) and a generalization (with low train loss *and* low test loss). In our case, the Fourier Multiplication Algorithm is the generalization solution. Intuitively, with very little training data, the model will overfit and memorize. With more training data, the model must generalize or suffer poor performance on both train and test loss. Since neural networks have an inductive bias favoring "simpler" solutions, memorization complexity scales with the size of the training set, whereas generalization complexity is constant. The two must cross at some point! Yet, the surprising aspect of grokking is the abrupt shift during training, when the model *switches* from memorization to generalization.

The other component of grokking is phase transitions - the phenomena where models trained on a certain task develop a specific capability fairly rapidly during a brief period of training, as shown for the case of induction heads forming in transformer language models in Olsson et al. (2022) and our results in Appendix D.3. That is, rather than slowly forming that capability over training, the model rapidly goes from being bad at it to being good at it. One interpretation of a phase transition is that there's some feature of the loss landscape that makes the generalising solution harder to reach - rather than a smooth gradient for the model to follow, it instead initially finds it difficult to make progress, but then crosses some threshold where it can rapidly make progress.

Therefore, grokking occurs with phase transitions, limited data, and regularization. Models exhibit phase transitions despite having enough training data to avoid overfitting. Regularization (weight decay in our case) favors simpler solutions over complex ones. The model has enough data to marginally prefer generalization over memorization. The phase transition indicates that generalization is "hard to reach" while the model has no problems with memorization. But as it memorizes, the network becomes more complex until the weight decay prevents further memorization then moves towards equilibrium. The gradient to memorize balances the gradient towards smaller weights. With generalization, the model is incentivized to both memorize and simplify. Strikingly, it is capable of both while maintaining a somewhat constant training performance in this circuit formation phase. Next, as the model approaches generalization, the memorization weights are removed in the cleanup phase. The cost from complexity outweighs the benefit from lower loss. Due to the phase transition during this training period, as model's progress towards generalization accelerates, the cleanup rate sharpens as well.

A model that learns a perfect solution and is trained with weight decay has competing incentives: larger weights (for more extreme logits and thus lower loss) and smaller weights (from weight decay). So for any solution and any level of weight decay, there will always be a level of train loss where these two forces equilibrate. Thus, memorization is not necessarily a "simpler" solution than generalization. The key is that generalization will have smaller weights *holding train loss fixed*. In

fact, weight decay should be expected to equilibrate at a slightly lower train loss in generalization, since the base solution is simpler. This matches what we observe in practice. [6]

E.2 Hypothesis: Phase Transitions are inherent to composition

A promising line of work in the growing field of mechanistic interpretability suggests that models form *circuits* (Cammarata et al., 2020) – clean interpretable algorithms formed by subnetworks of the model, such as curve detectors (Cammarata et al., 2020) in image classification networks and induction heads (Elhage et al., 2021; Olsson et al., 2022) in LLMs. This is surprisingly true! A circuit represents the model learning an algorithm, a fundamentally discrete thing; each step in the algorithm only makes sense if the other steps are present. But neural networks are fundamentally continuous, trained to follow gradients towards lower loss and struggle to jump to new optima without following a smooth gradient. So how can a model learn a discrete algorithm?

As a concrete example, let's consider the case of induction heads in LLMs. There is a subnetwork of a next-token prediction autoregressive language model that learns to continue repeated subsequences. It detects whether the current token occurred earlier in the context. If so, it predicts the same token after that previous occurrence will also come next. The circuit consists of a previous token head, which attends to each previous token and copies the context of the previous token to the current token, and an induction head which attends to the token *after* a previous occurrence of the current token. The induction head composes with the previous token head by forming a query vector representing the current token and a key vector representing the previous token head's output using K-Composition, the context of the previous token. It attends to a token where this query and key match.

This circuit significantly improves loss but only in the context of the other heads present. Before either head is present, no gradient encourages the formation of either head. At initialization, we have neither head, so gradient descent should never discover this circuit. Naively, we might predict that neural networks will only produce circuits analogous to linear regression, where each weight will marginally improve performance as it continuously trains. And yet in practice, neural networks indeed form such sophisticated circuits, involving several parts interacting in non-trivial, algorithmic ways. So how can this be?

A few possible explanations:

- **Lottery tickets (Frankle & Carbin, 2018):** Initially, each layer of the network is the superposition of many partial circuit components, and the output of each layer is the average of the output of each component. The full output of the network is the average of many different circuits, with significant interference from non-linear interaction. Some of these circuits are systematically useful to reducing loss, but most aren't. Gradients for useless circuits will have zero mean, while gradients for useful circuits will have non-zero mean, with a lot of noise. SGD reinforces relevant circuits and suppresses useless ones, so circuits will gradually form.
- **Random walk:** The network wanders randomly around the loss landscape until it encounters a half-formed previous token head and induction head that somewhat compose. This half-formed circuit becomes useful for reducing loss, so gradient descent completes the circuit.
- **Evolution:** A similar mystery arises from how organisms develop sophisticated machinery, like the human eye. Each part is only useful in the context of other parts. A compelling explanation is a component first developed that was somewhat useful in its own right, like a light-detecting membrane. It was reinforced as a useful component. Then, later components developed depending on the first, like the lens of the eye.

Evolution is a natural explanation, However, based on our toy tasks, it cannot be the whole story. In the repeated subsequence task, we have a sequence of uniform randomly generated tokens, apart

[6] One subtlety: the grokking phenomena is often incorrectly summarized as "the model learned to generalize even after achieving zero loss." Zero loss does not exist with cross-entropy loss. Although the model achieves perfect *accuracy*, it is trained to optimize loss not accuracy. This means the model is *always* incentivized to further improve. In particular, the easiest way to improve performance with perfect accuracy is by scaling up the logits. This lowers the temperature and pushes the softmax closer to an argmax.

from a repeated subsequence at an arbitrary location, e.g. 7 2 8 3 1 9 3 8 3 1 9 9 2 5 END. This means all pairs of tokens are independent, apart from pairs of equal tokens in the repeated subsequence. In particular, this means that a previous token head can never reduce loss for the current token. The previous token will always be independent of the next token. So a previous token head is only useful in the context of an induction-like head that completes the circuit. Likewise, an induction head relies on K-composition with a previous token head and so cannot be useful on its own. Yet the model eventually forms an induction circuit!

A priori, the random walk seems insufficient on its own. An induction circuit is relatively complicated, representing a small region in model space. So a random walk is unlikely to stumble upon it. Concretely, in our modular addition case, progress measures show significant hidden progress pre-grokking, indicating the model did not stumble upon the solution by chance.

Thus, the lottery ticket hypothesis seems the most explanatory. An induction head is useless without a previous token head but might be slightly useful when composing with a head that uniformly attends to prior tokens, since part of its output will include the previous token! Nevertheless, we suspect that all explanations contribute to the entire picture. This seems most plausible if the uniform head just so happens to attend a bit more to the previous token via a random walk.

Returning to phase transitions, the lottery ticket-style explanation suggests that we might expect phase transitions as circuits form. Early in circuit formation, each part of the circuit is rough, so the effect on the loss of improving any individual component is weak, meaning gradients will be small. As each component develops, other components will become more useful, meaning all gradients will increase together non-linearly. As the circuit nears completion, we should expect an acceleration in the loss curve for this circuit, resulting in a phase transition.

F Further discussion on using mechanistic interpretability and progress measures for studying emergent phenomena

While we find approach of using mechanistic interpretability to define progress measures relatively promising, there remains significant uncertainty as to how scalable existing mechanistic interpretability approaches really are. Broadly speaking, depending on the success of future mechanistic interpretability work, we think there are three methods through which mechanistic interpretability and progress measures can help with understanding and predicting emergent phenomena:

1. If mechanistic interpretability can be scaled to large models to the level where we can understand the mechanisms behind significant portions of their behavior, we could perform the same style of analysis as was done in this work. We believe it's currently unclear as to whether or not mechanistic interpretability will successfully scale to large models to this extent (or even if there exist human-understandable explanations for all of their sophisticated behavior). That being said, in cases where mechanistic interpretability does recover human-understandable mechanisms, we could simply use the parts of the mechanism as progress measures.
2. If future mechanistic interpretability can only recover parts of the mechanism of larger models and can only generate comprehensive understanding of the mechanisms of smaller models, we might still be able to use our understanding from smaller models to guide the development measures that track parts of the behavior of the larger model. We find this scenario relatively plausible, as existing mechanistic interpretability work already allows us to recover fragments of large model behavior and understand these fragments by analogy to smaller models. For example, Olsson et al. (2022) use this approach to understand the emergence of in-context learning in medium-sized language transformers.
3. Even if mechanistic interpretability fails to recover understandable mechanisms at all on large models, we might still be able to derive progress measures that don't require human understanding. For example, if we end up with automated mechanistic interpretability (that nonetheless still fails to recover human-understandable mechanisms), we might be able to use the outputs of those opaque processes.

   Another approach is task-independent progress measures: if we can discover progress measures that don't depend on the task, perhaps using many small, interpretable models as testbeds, we might be able to apply these progress measures to large models.

That being said, we think the future work outlined in 6 is necessary to successfully apply our approach to predict and understand emergent behavior in existing large language models, and so remain cautiously optimistic.