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 the 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 of the network that perform memorization.

1 INTRODUCTION

Neural networks often exhibit emergent behavior, in which new qualitative capabilities 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. that detect earlier repeated subsequences. 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. Several variants of this task exhibit grokking for small transformers trained with weight decay (Appendix B.2).

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

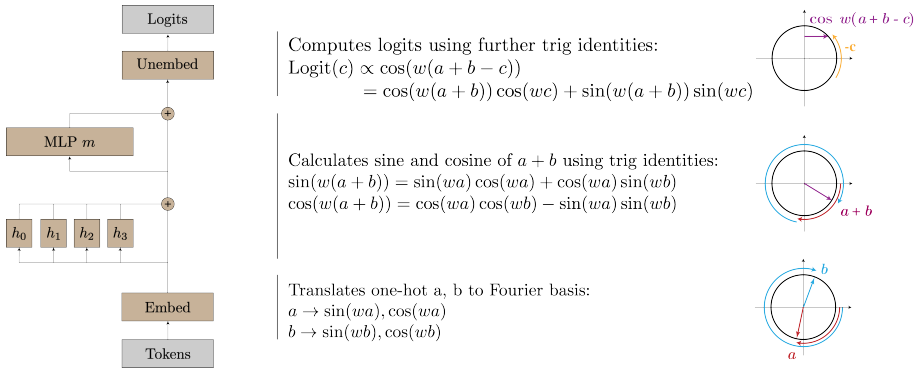

Figure 1: The algorithm implemented by the one-layer transformer for modular addition. Given two numbers a and b, the model projects each point to a corresponding rotation using its embedding matrix. Using its attention and MLP layers, it then composes the rotations to get a representation of $a + b \mod P$. 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 combines 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 the 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 the 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 clean-up, *after* the generalizing mechanism was learned. 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 training 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 arises from an optimization anomaly they dub the slingshot mechanism. Our results show that this mechanism is unnecessary for grokking to occur.

Mechanistic interpretability and progress measures. There is a growing literature that seeks to mechanistically understand neural networks. The style of post-hoc mechanistic interpretability in Section 4 is heavily inspired by the Circuits approach of Cammarata et al.(2020), Elhage et al.

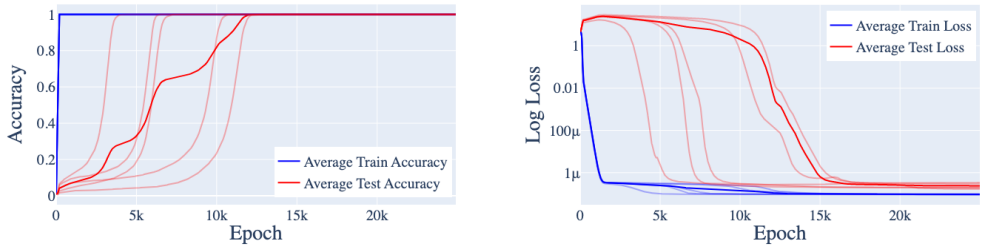

Figure 2: The train and test accuracy (left) and train and test loss (right) of one-layer transformers on the modular addition task described in Section 3, over 5 random seeds. These models consistently exhibit grokking: they quickly overfit early on in training, but then later learn to generalize.

(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 113^2 pairs of numbers mod P). We used full batch gradient descent. We used 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 113^2 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: training 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 part of its computation (Appendix A), 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 B.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, weight decay is crucial: in Appendix C, we demonstrate that weight decay 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.

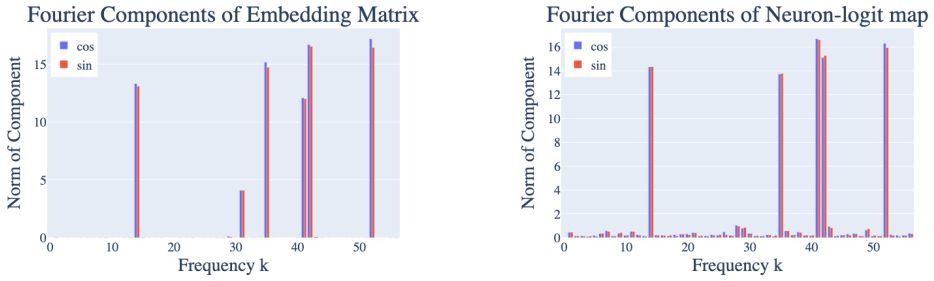

Figure 3: *(Left)* The norms of the Fourier components in the embedding matrix W_E. As discussed in Section 4.1, the sparsity of W_E in the Fourier basis is evidence that the network is operating in a Fourier basis. Of the six non-zero frequencies, five frequencies appear in later parts of the network, 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.

- 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 (see Section B.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*.

We next look into the actual **mechanisms** implemented in the model weights (Section 4.2). The embedding and unembedding matrices are both linear combinations of the key frequencies. Moreover, the unembedding coefficients match those in the cosine addition formula, and projecting the MLP activations in the unembedding directions 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.

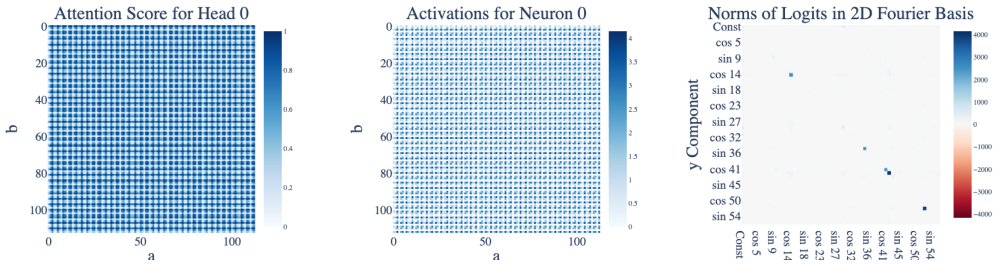

Figure 4: *(Left)* The attention score for head 0 from $=$ to a, as a function of inputs a, b. *(Center)* The activations of MLP neuron 0 given inputs a, b. Both the attention scores and the neuron activations are periodic, as discussed in Section 4.1. *(Right)* The norm of the Fourier components of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

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

Next, we verify that the model uses the Fourier multiplication algorithm by examining its weights. We show that the logits consist of weighted sums of $\cos(w_k(a+b-c))$ for the 5 key frequencies w_k and confirm these results by approximating the logits with such a weighted sum.

The W_L is well approximated by $\cos(w_k c)$ and $\sin(w_k c)$. Our algorithm computes logits using the trig identity $\cos(w_k(a+b-c)) = \cos(w_k(a+b))\cos(w_k c) + \sin(w_k(a+b))\sin(w_k c)$. If the model implements the algorithm, W_L should contain be rank 10, with terms corresponding to $\sin(w_k c)$ and $\cos(w_k c)$ for the key frequencies. 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. To further confirm that W_L implements the trigonometric identity, we examine the dot product of the MLP activations with each of the 10 directions in W_L identified in the previous paragraph. We display the results in the Fourier basis in Table 3: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 $\sin(w_k a)\sin(w_k b)$ (both of which have approximately equal coefficients), and projecting onto the $\sin(w_k c)$ direction gives us terms of the form $\sin(w_k a)\cos(w_k b)$ and $\sin(w_k a)\cos(w_k b)$ (also with approximately equal coefficients).

W_L Direction	Fourier transform of projection of MLP activations onto direction	FVE
$\cos(w_{14}c)$	$2522\cos(w_{14}a)\cos(w_{14}b) - 2466\sin(w_{14}a)\sin(w_{14}b) \approx 2494\cos(w_{14}(a+b))$	93.2%
$\sin(w_{14}c)$	$2490\sin(w_{14}a)\cos(w_{14}b) + 2494\cos(w_{14}a)\sin(w_{14}b) \approx 2492\sin(w_{14}(a+b))$	93.5%
$\cos(w_{35}c)$	$2298\cos(w_{35}a)\cos(w_{35}b) - 2466\sin(w_{35}a)\sin(w_{35}b) \approx 2382\cos(w_{35}(a+b))$	96.8%
$\sin(w_{35}c)$	$2362\sin(w_{35}a)\cos(w_{35}b) + 2364\cos(w_{35}a)\sin(w_{35}b) \approx 2363\sin(w_{35}(a+b))$	96.5%
$\cos(w_{41}c)$	$2533\cos(w_{41}a)\cos(w_{41}b) - 2529\sin(w_{41}a)\sin(w_{41}b) \approx 2531\cos(w_{41}(a+b))$	97.0%
$\sin(w_{41}c)$	$2516\sin(w_{41}a)\cos(w_{41}b) + 2516\cos(w_{41}a)\sin(w_{41}b) \approx 2516\sin(w_{41}(a+b))$	97.0%
$\cos(w_{42}c)$	$3562\cos(w_{42}a)\cos(w_{42}b) - 3872\sin(w_{42}a)\sin(w_{42}b) \approx 3717\cos(w_{42}(a+b))$	96.4%
$\sin(w_{42}c)$	$3829\sin(w_{42}a)\cos(w_{42}b) + 3831\cos(w_{42}a)\sin(w_{42}b) \approx 3830\sin(w_{42}(a+b))$	96.4%
$\cos(w_{52}c)$	$3419\cos(w_{52}a)\cos(w_{52}b) - 3703\sin(w_{52}a)\sin(w_{52}b) \approx 3561\cos(w_{52}(a+b))$	97.4%
$\sin(w_{52}c)$	$3463\sin(w_{42}a)\cos(w_{42}b) + 3463\cos(w_{42}a)\sin(w_{42}b) \approx 3463\sin(w_{42}(a+b))$	98.2%

Table 1: For each of the directions in the unembedding matrix, we project the MLP activations in that direction and then perform a Fourier transform. For brevity, we ignore trivial terms—those with coefficients less than 1.5% of the largest coefficient. We then compute the fraction of variance explained (FVE) if we replace the projection with 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.

Logits are well approximated by a weighted sum of $\cos(w_k(a+b-c))$s. Next, we approximate the output logits as a sum of $\sum_{k\in\{14,35,41,42,52\}} A_k \cos(\frac{2k\pi}{P}(a+b-c))$ and fit the coefficients A_k. When we do this, the resulting approximate logits explains 95% of the variance in the original logits. This is surprising—naively the output logits are a 113^3 dimensional vector, yet we can the logits with just 5 directions predicted by our interpretation. If we evaluate test loss on this approximation to the logits, we actually see a significant *improvement* in loss, from $2.4 \cdot 10^{-7}$ to $4.7 \cdot 10^{-8}$.

Taken together, these results confirm that the model computes sums of terms of the form $\cos(w_k(a+b-c)) = \cos(w_k(a+b))\cos(w_k c) + \sin(w_k(a+b))\sin(w_k c)$.

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 113^2 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 attention pattern weights for each of the four attention heads, we find that the attention pattern of the $=$ to a for each of the four heads is 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 B.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 forms 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. We display the histogram in Figure 5.

Maps to the logits are localized by frequency. We next partition these 433 neurons based on which frequency has 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.

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

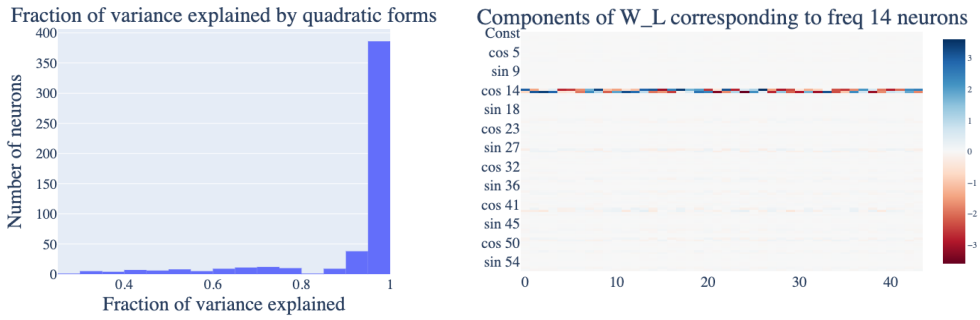

Figure 5: (Left) The vast majority of neurons are well-approximated by quadratic forms on sinusoidal functions of the inputs. (Right) The weights in W_L corresponding to each of the 44 neurons of frequency 14. The non-trivial components correspond to $\sin(w_k)$ and $\cos(_k)$ for $k = 14$.

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.

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 (we leave the other neurons untouched). This increases loss by only 3% in relative terms (from 2.41×10^{-7} to 2.48×10^{-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 5.54×10^{-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 $113 \times 113 \times 113$ logit matrix over all 113^2 pairs of inputs to get the logits in the Fourier basis, and 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* $113^2 - 40$ of the Fourier components outside the five key frequencies; this *improved* performance. Insofar as other frequencies are used, they appear to serve as noise that degrades the 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 1.19×10^{-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 both depend on these directions and depend *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 since 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, 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.

We also examine the ratio of test loss to restricted loss. As restricted loss measures the test loss *after* removing the memorization solution, we expect test loss to be much higher than restricted loss while the generalizing circuit has been formed, but before the memorization solutions have been removed.

Excluded loss. The excluded loss is the opposite of the restricted loss: instead of keeping only the important frequencies w_k, we remove *only* these frequencies from the logits and keep the rest. Intuitively, this metric tracks how much the model's performance on the *training* data comes from the Fourier multiplication algorithm, vs 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 the ℓ_2-norm of the weights during training, since weight decay should push these down once the training loss is near zero.

5.2 PHASES OF GROKKING: MEMORIZATION, CIRCUIT FORMATION, AND CLEAN-UP

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 clean-up phases.

Memorization (Epochs 0k–1.4k). We first observe a decline of both excluded loss and train loss, with test loss and restricted loss both remaining high. In other words, the model memorizes the data, and the frequencies w_k used by the final model are not used.

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 and 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 fact that the ratio of test to restricted loss plateaus and then sharply decreases.

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 this hypothesis via ablation—in the absence of weight decay, grokking fails to occur.

In Appendix B.2 and C, 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 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

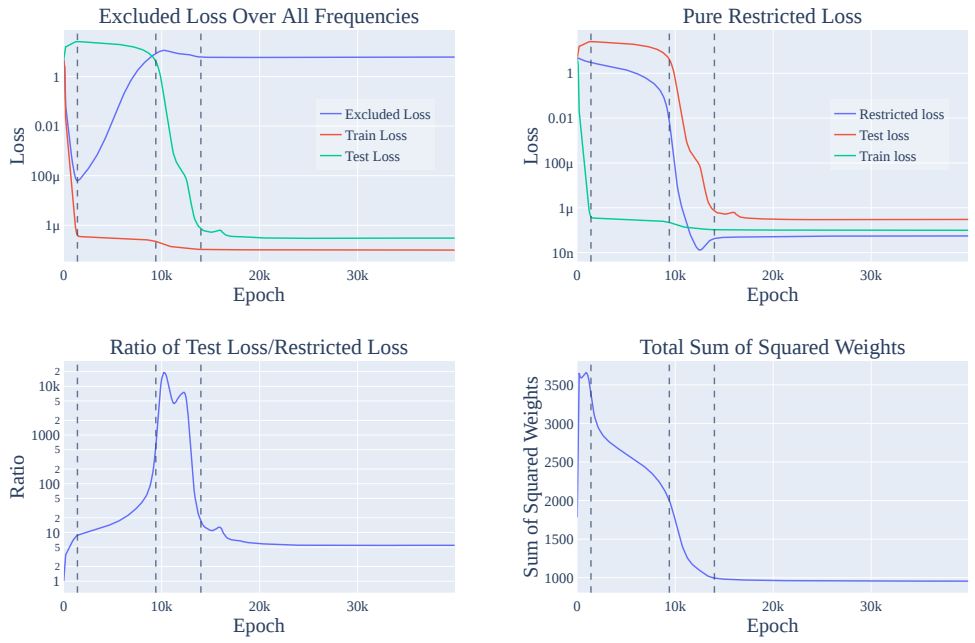

Figure 6: How each of the progress measures in Section 5.1 changes over the course of training. The lines delineate the 3 phases of training, memorization, circuit formation and cleanup (and a final stable phase) *(Top Left)* Excluded loss increases during circuit formation, while train and test loss remain flat. *(Top Right)* The restricted loss begins declining before test loss declines. *(Bottom Left)* The ratio of test loss/restricted loss increases during circuit formation, then returns to baseline during clean-up *(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.

training and test losses (instead, both decline sharply some number of epochs into training). Finally, in Section C.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 believe that these results show the promise of using mechanistic interpretability to define continuous progress measures. Below, we discuss some of directions for future work.

Larger models and realistic tasks In this work, we studied the behavior of small transformers on a simple algorithmic task, that can be solved with a single circuit. On the other hand, larger models use many circuits implemented on orders of magnitude more neurons to solve significantly more challenging tasks. In addition, the models we study are very overparameterized relative to the task, which is likely untrue for realistic tasks.

Automated mechanistic analysis The analysis reported in this work required significant amounts of manual human effort to produce, and our progress metrics are specific to one network on one particular algorithmic task. Automating the analysis would allow it to scale to more, 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.

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.

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.

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 the specific case of our model.

We denote our hypermeters 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 (ie 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^i (queries), W_K^i (keys), W_V^i (values), W_O^i (attention output) (the 4 weight matrices of head i 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, ie 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.

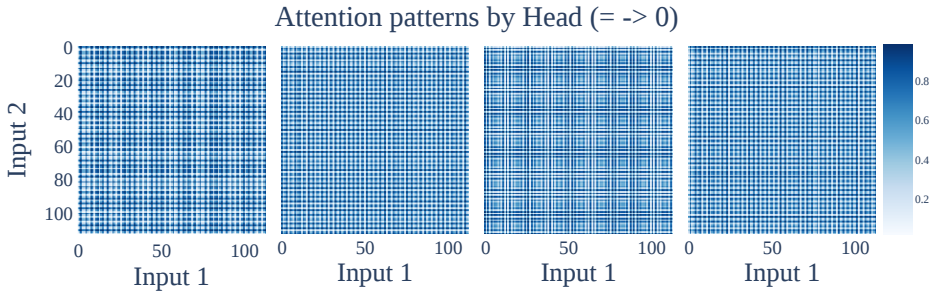

Figure 7: Attention Patterns for each head, from position = to a, as a heatmap over the inputs. All four attention heads exhibit striking periodicity.

Head	k	C	D	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 $0.5 + C(\cos(w_k a) - \cos(w_k b)) + D(\sin(w_k a) - \sin(w_k b))$, and give the coefficients and fraction of variance explained for this approximation.

B Supporting evidence for mechanistic analysis of modular arithmetic networks

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

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

B.1.1 Periodicity in the activations of other attention heads

In Figure 7 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, and 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.

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

We can further show that the periodicity of the attention heads has a striking form—A_0^j is well approximated by $0.5 + C(\cos(w_k a) - \cos(w_k b)) + D(\sin(w_k a) - \sin(w_k b))$, for some frequency w_k and constants C and D (which may differ for each head). Note further that this simplifies to $0.5 + E(\cos(w_k(a + \theta)) - \cos(w_k(b + \theta)))$ for some constants E and θ. We show the coefficients and fraction of variance explained in table 3

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

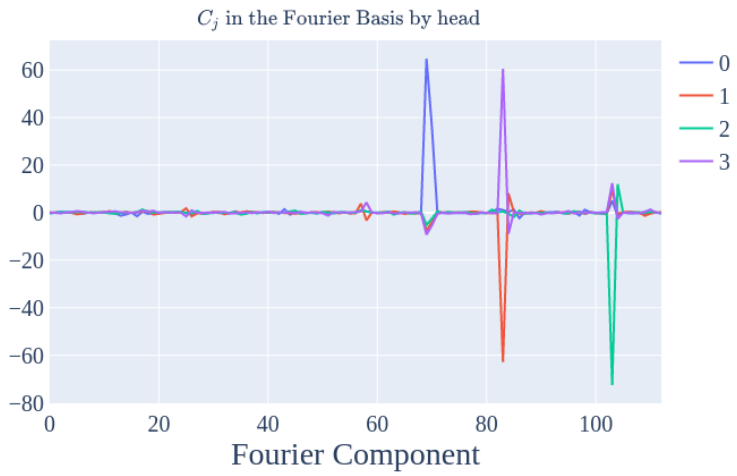

Figure 8: We plot the attention pattern weights, C_j in the Fourier Basis, for each head j. We observe significant sparsity, with almost all of each term being associated with a single frequency. Note: components are given in the order 0:constant, 1:$\cos w_1$, 2:$\sin w_1$, etc. Giving 113 total for 56 frequencies

$C_j = W_E^T {W_K^j}^T W_Q^j x_2^{(0)} \in \mathbb{R}^p$ 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 8. 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.

We validate that this is all that is going on, by intervening to replace 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 $0.5 \pm D_j(\cos(w_{k_j}(a+\theta_j)) - \cos(w_{k_j}(b+\theta_j)))$ - 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 7.

B.1.3 Periodicity in the activations of additional neurons

In Figure 9, 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.

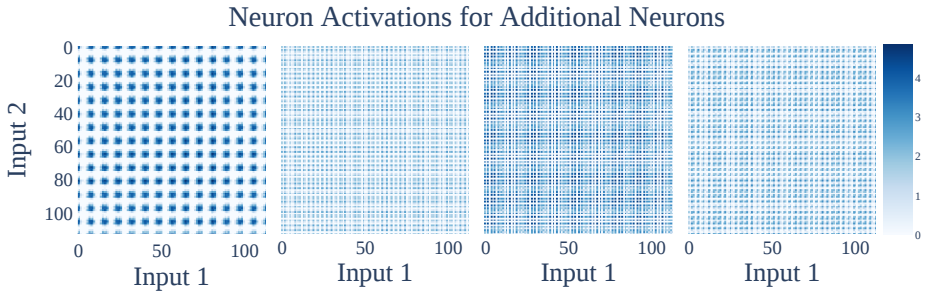

Figure 9: Plots of neuron activations for MLP neurons 1, 2, 3 and 5. As with Neuron 0, all of the activation patterns are periodic in both inputs.

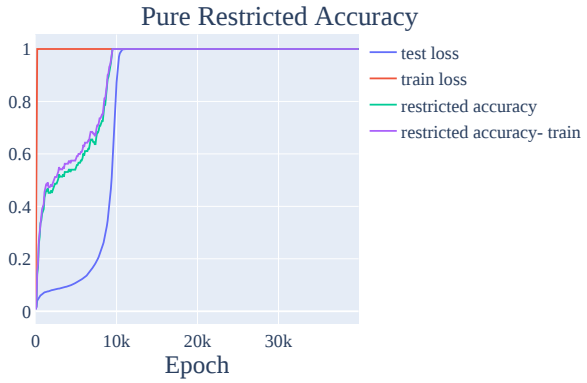

Figure 10: Accuracy when restricting Fourier Components to the five key frequencies. As with restricted loss, this shows that the model figures out how to generalise modulo deleting noise earlier than deleting noise

B.1.4 Additional grokking figures for this architecture

In Figure 10, 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 11, 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 **??**, 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.

B.2 Additional graphs from different runs

B.2.1 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.

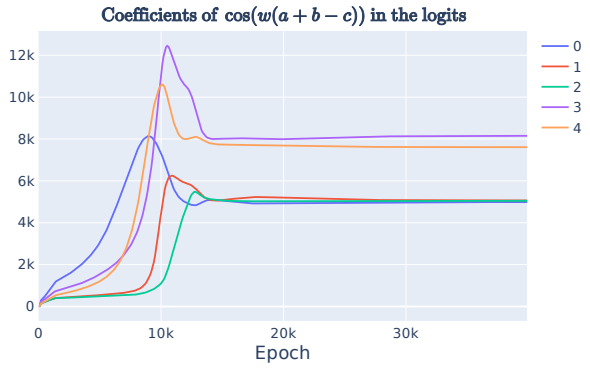

Figure 11: The coefficients of $\cos(w(a+b-c))$ in the logits over the model's training. As with the metrics in the paper, this shows a nice interpolation and growth of each cos term.

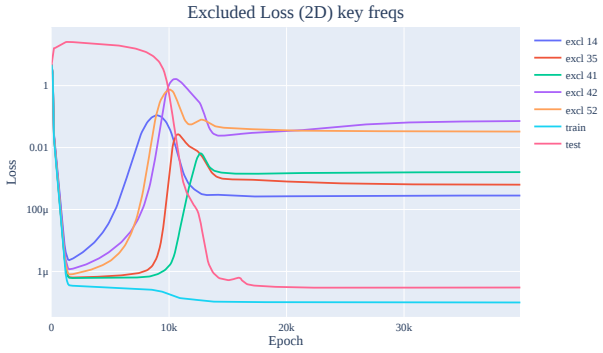

Figure 12: The excluded loss if we exclude each of the five key frequencies. As with the excluded loss results in Section 5.1, this hsows that the model interpolates between memorising and generalising.

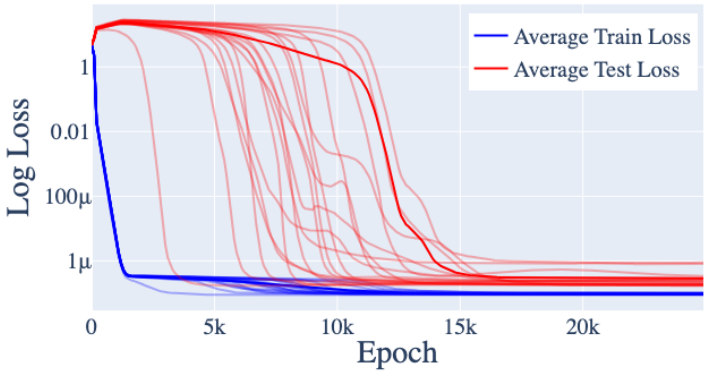

Figure 13: Training and test losses for a 1-layer transformer with training data fractions from 0.05 to 0.95, stepped by 0.05, inclusive. While the majority of the runs exhibit grokking, runs with 90% or 90% of the data do not exhibit grokking, and broadly speaking higher fractions of the data lead to the test loss declining sooner.

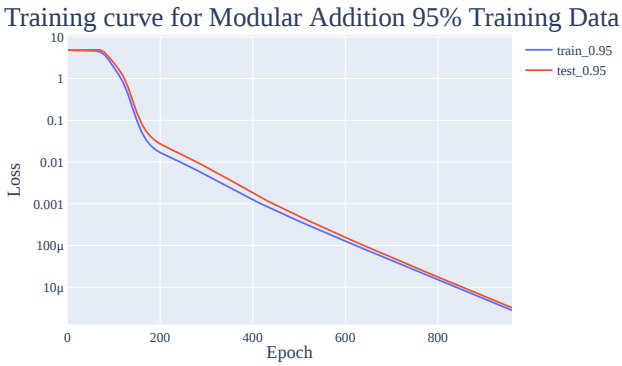

Figure 14: Training Curve for Modular Addition, 95% as training data. Interpretation: Shows a phase change in both train and test

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 113^2 pairs of inputs that the model sees during training), as shown in Figure 13. Smaller amount lead to slower grokking, but sufficiently large fractions of data ($\geq 85\%$) lead to immediate generalization, as shown in Figure 14.

2 Layer Transformers As shown in Figure 15, 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 16, 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. See and Figure 17 and Figure 18. For $P = 53$, 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

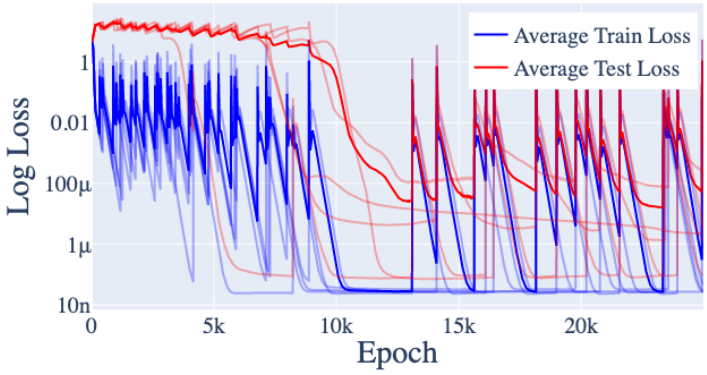

Figure 15: Training and test loss for a 2-layer version of the original architecture. Average across 5 random seeds is in bold.

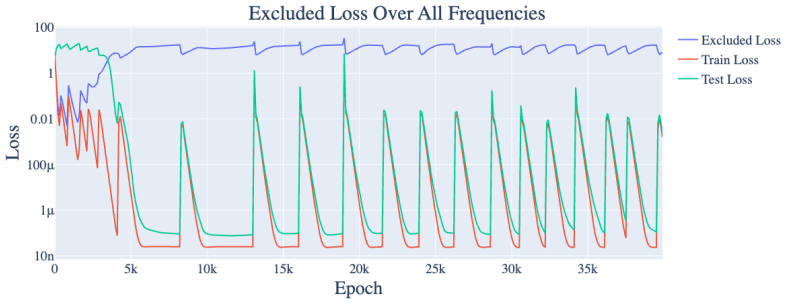

Figure 16: Training, test, and full excluded loss for a 2-layer version of the original architecture. One random seed chosen for readability.

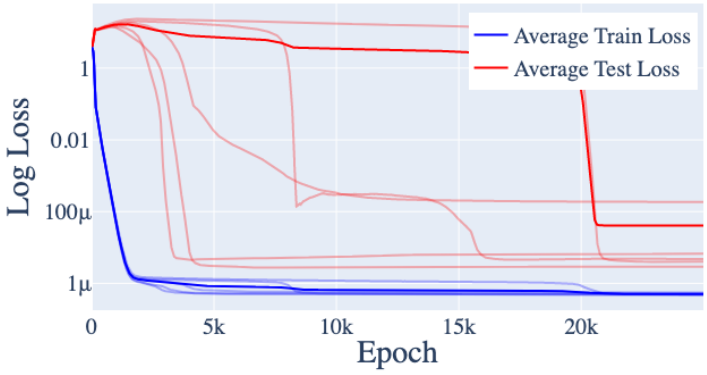

Figure 17: The training and test losses for $p = 53$ and all other hyperparameters except weight decay (which we set to $\gamma = 5$) the same as the main training run discussed in the paper. The averages are bold, and all contributing runs are partially transparent. Note that grokking occurs.

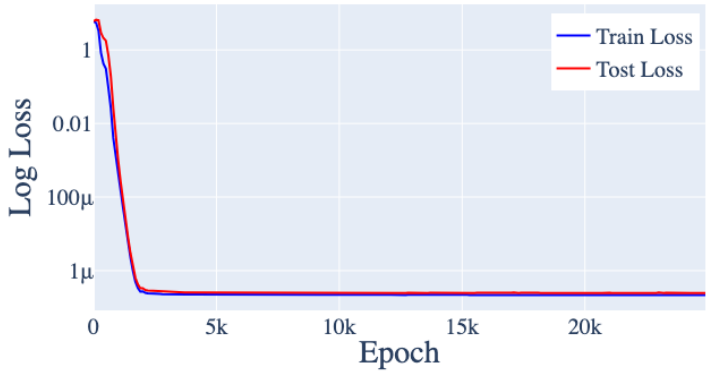

Figure 18: The training and test losses for $p = 401$ and all other hyperparameters the same as the main training run discussed in the paper. Note that Grokking doesn't occur, even across a variety of weight decays.

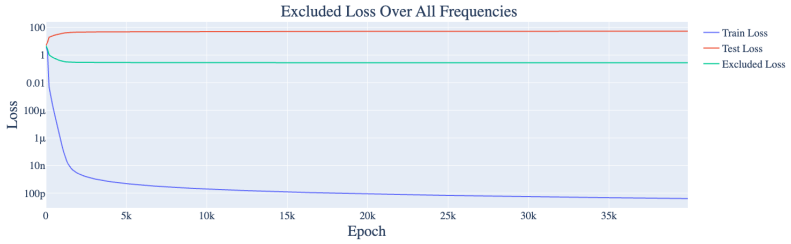

Figure 19: Training, test, and full excluded loss for a 1-layer version of the original architecture without weight decay. One random seed chosen for readability. Note that not having weight decay prevents grokking.

speculate that this is because the memorization solution is significantly smaller (since there are only 53^2 total pairs), thereby requiring more aggressive weight decay for the generalizing solution to be favored.

For $P = 401$, we could not get grokking, even if we varied the weight decay parameter. Instead, the model just immediately learns the generalizing solution (Figure 18). 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 113^2 pairs to 30% of 401^2 pairs), thereby favoring the generalizing solution from the start.

C Additional results on grokking

C.1 Both weight decay and limited data are necessary for grokking

As discussed in Section 6 and Appendix B.2, the weight decay and the amount of data seem to have a strong effect on whether grokking occurs. To confirm this, in this section we experiment with removing weight decay and varying the amount of data on 1-layer transformers. In Figure 19, we give the training, test, and full excluded loss for a typical training run with $\lambda = 0$ (that is, 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.

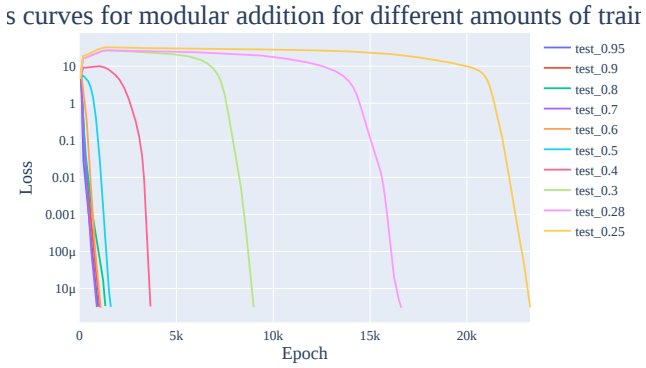

Figure 20: The test loss for 1-layer transformers on modular addition, with various fractions of the full data encountered during training. Though all the training losses are approximately the same—that is, they memorise at the same rate, smaller fractions of data take longer to grok.

In Figure 20, we show the test loss curves for models trained on various fractions of the data. (The training curves are ommited as they are identical and quickly converge to 0.) Though all the training losses are approximately the same—that is, they memorise at the same rate, smaller fractions of data take longer to grok.

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

As noted in Section B.2, our 2-layer transformers exhibit significant sling shot mechanisms 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 casues many gradients to round to 0, this 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.

C.3 Additional evidence from other algorithmic tasks

In this section, we 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.
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 23. Here, 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 (since there are 10^10 such pairs). Interestingly, the various small bumps in Figure 23 correspond to the model learning how to calculate each of the 6 tokens in the output. Figure 22 shows the breakdown of the loss per token.

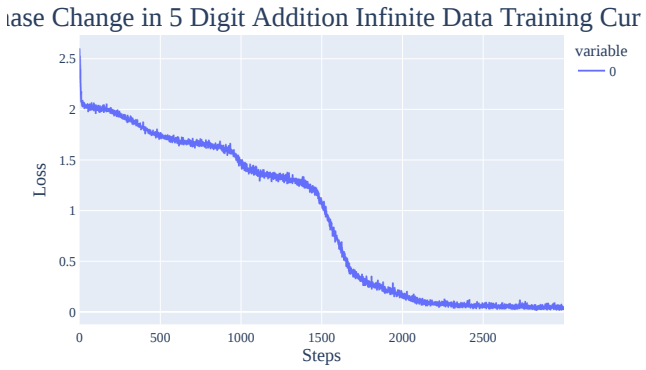

Figure 21: The training/test loss for 5 Digit Addition trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs. There are several phase changes, corresponding to the model learning how to correctly predict each of the 6 tokens of the output.

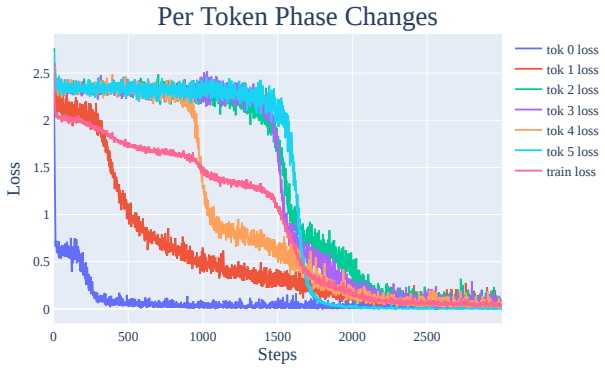

Figure 22: The loss *per token* for 5 Digit Addition, trained with randomly generated data at each step. This reveals that the bumps in in Figure 21 correspond to 6 distinct phase changes, corrsponding to when the model learns to predict one of the 6 tokens of the output.

However, grokking does occur when we restrict the model to only see 700 data points, as shown in Figure 23

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. Figure **??**, where the model sees new data at every step exhibits no grokking, while Figure 25, where we restrict the model to only see 512 data points, exhibits clear grokking.

Skip trigram As with the previous tasks task, we find that restricting the amount of data is necessary and sufficient for grokking on the skip trigram task. Figure 26, where the model sees new data at every step exhibits no grokking, while Figure 27, where we restrict the model to only see 512 data points, exhibits clear grokking.

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

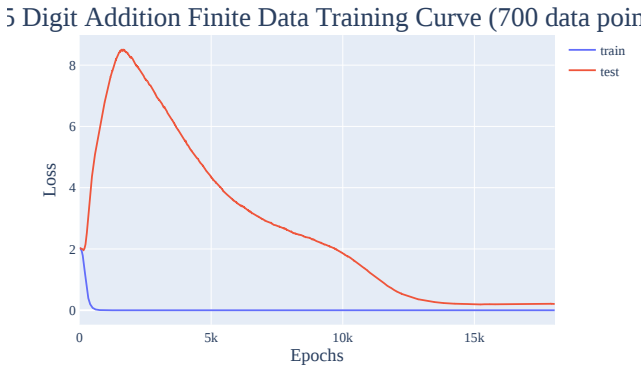

Figure 23: The train and test loss for 5 Digit Addition trained on 700 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.

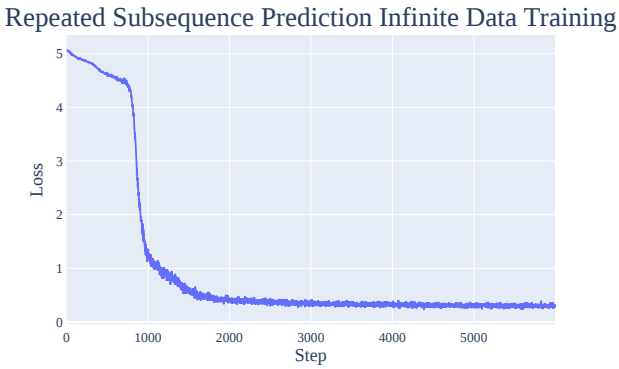

Figure 24: The training/test loss for repeated subsequences trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs. There sharp phase change corresponds to the model forming induction heads. (Olsson et al., 2022)

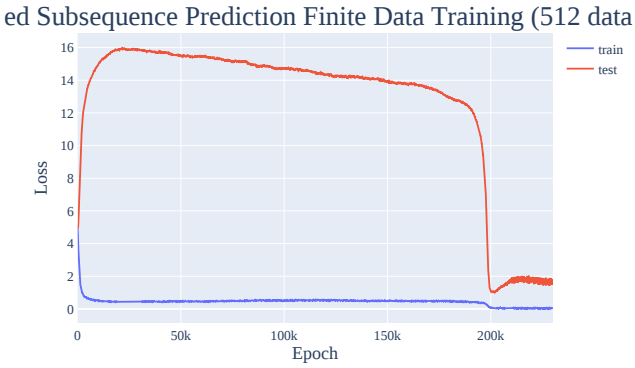

Figure 25: The train and test loss for the repeated subsequence task, trained on 512 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.

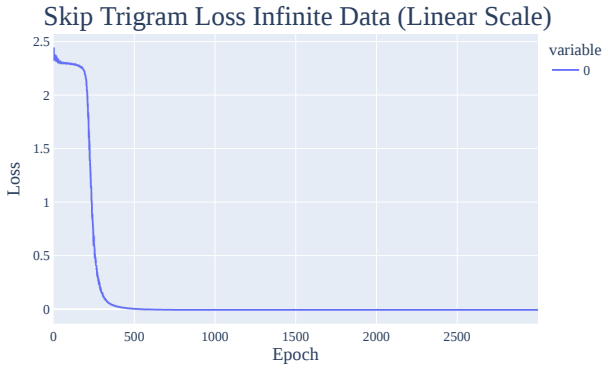

Figure 26: he training/test loss for the skip trigram task, trained on randomly generated data. Note that training and test loss coincide, as the model does not see repeated pairs. The sharp phase change corresponds to the network learning all of the skip trigrams.

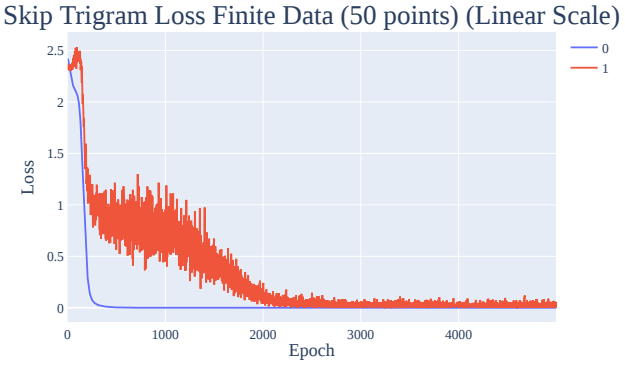

Figure 27: The train and test loss for the skip trigram task, trained on 512 data points. Unlike the infinite, randomly generated data case, this shows both a sharp phase change and clear train test divergence.