Mechanistic Interpretability and Grokking Phenomenon in Transformer Models

Mann Patel **Che Wei Wu** **Huy Nguyen** **Te Li** **Luis Gil**

1 Introduction

Mechanistic interpretability is a subfield within the realm of Explainable AI. It provides a robust mathematical framework and methodology for reverse engineering a neural network. The goal of our project is to explore algorithmic tasks that exhibit the "Grokking" phenomenon, which is defined as a sudden generalization occurring well after overfitting Power et al. (2022). We aim to fully understand "Grokking" by analyzing the attention head patterns, neuron activations, and circuits of a small model, along with the proposed metric called the "Progress measure" Nanda et al. (2023) By deciphering this sudden emergent phenomenon, we can potentially revolutionize the prevailing methodologies of neural network training. Furthermore, it may shed light on the mysterious Double Descent Belkin et al. (2019) phenomena observed in overparameterized neural networks.

A variety of experiments were performed, ranging from novel attempts to classify prime numbers to modifications of successful experiments involving modular addition. In experiments where the "Grokking" phenomenon is found to exist, the goal is to apply mechanistic interpretation, identify variations from the original mechanism, and discover the new mechanism.

A compelling hypothesis in the field of mechanistic interpretability states that different models learn similar features and circuits when trained on similar tasks Cammarata et al. (2020). Following this line of thinking, we carry out variations of the modular addition task that would result in a differentiated mechanism that retains some characteristics of the original mechanism Nanda et al. (2023) and Chughtai et al. (2023a). The modular addition task described by Nanda et al. takes inputs $x, y \in \{0, ..., P-1\}$ for some prime P and predicts their sum c mod P. This experimental setup [1] is known to "grok" consistently and provides a straightforward analytical basis, allowing it to be extended into a more generalized mathematical form. Taking the form of this arithmetic task as a basis, we exponentiate the input values with small positive integer powers and find that—without many changes to the original transformer architecture–the model also groks consistently. We predict that these variations must maintain some of the trigonometric relationships found in the original mechanism.

In surveying other mathematical tasks that transformer models might be able to learn by generalization, we found literature Charton (2023) stating that—under specific setups—sequence-to-sequence transformers are capable of finding the greatest common denominator (GCD) of two small integers. We attempted to adapt this approach to one that does not require the sequence-to-sequence component of the original experiment. Furthermore, as the mechanistic interpretation of the transformer demonstrates that the model learns a sieving algorithm that discovers and utilizes prime numbers, we were motivated by the potential for transformers to classify numbers as prime and non-prime. Building on the premises of the GCD experiment, we attempted to train a transformer to identify prime numbers below 100.

[1] Our code is available at https://github.com/manncodes/mech-interp

2 Related Work

Before doing interpretability, there were studies about how small transformers, trained from random initialization, can efficiently learn arithmetic operations such as addition, multiplication, and elementary functions like square root, using the next token prediction objective Lee et al. (2023). After "Grokking" a model on arithmetic tasks, research has been done on what set of operations transformers can learn and, more recently, how an operation is performed internally by transformers.

The *universality hypothesis* in mechanistic interpretability states that different models learn similar features and circuits when trained on similar tasks Chughtai et al. (2023b). Two forms of this hypothesis exist: strong and weak. The strong universality hypothesis posits that the exact same features and circuits arise in models trained across similar tasks. The weak hypothesis takes a step back and claims that there are underlying principles to these aforementioned features and circuits to be understood and that models leverage these features somewhat arbitrarily when converging upon a generalizing solution.

The discovery of a mechanism for modular addition by Nanda et al. (2023) through the process of reverse engineering demonstrates a three-part process learned by the model: A mapping of inputs to a Fourier basis for the token embeddings; The use of attention heads and perceptron to linearly transform the appropriate embedded tokens through angle-sum trigonometric identities; An unembed process that takes the output of the perceptron to translate it into the final sum.

Work by Chughtai et al. analyzes a class of group-theoretical problems wherein inputs and outputs exist as a finite group and solutions entail manipulation of an irreducible representation of that group. Here, the authors show strong evidence that if an irreducible representation for the input values of a function exists, the model can learn to emulate any function that is a linear transformation of that representation through manipulation carried out by the transformer's attention heads and perceptron layer, as long as the desired output remains within the original representation group.

In the case of modular arithmetic, the modulus creates a group structure that is cyclical about the chosen prime p. This ensures that the output will have the same group structure as the input as long as inputs are limited to the range $[0, p)$. As the inputs and outputs define a cyclical group, an efficient representation is given by the rotational symmetric group that corresponds to radius-preserving rotations in 2-dimensions. Any linear combination of these rotations will produce a result within the same group. This allows the model weights to essentially search for the best linear combinations that approximate the given mathematical function. For the modular addition problem, these linear combinations replicate the angle-sum identities of trigonometry. However, the attention heads and perceptron are mechanistically unaware that they are linearly combining trigonometric functions; it is simply the most effective linear combination, given the basis it is working with. This motivates the case for the exponentiated modular addition task. Regardless of the exponent used, the modulus maintains the cyclical nature of the inputs and outputs, suggesting a rotationally symmetric mechanism will be found through learning.

In order to not rely too heavily on problems that fall into the aforementioned class of problems, the Greatest Common Divisor problem was found in literature Charton (2023) to have a distinct mechanism underlying its transformer-based solution. Interpretation of the mechanism suggests the model is following the same procedure as described by the *sieve of Eratosthenes* algorithm. Across iterations, where training batches are generated anew with probabilistically distinct inputs, the model learns the powers of the numeric base B it is working in, as well as the powers of the primes that divide B. It classifies numbers into classes of pairs divisible by pairs of these primes. Once it has learned and classified all the multiples of these primes, it begins to categorize the remaining numbers (all of which are prime at that point) into their own class. Successful observation of this the Sieve algorithm was found to rely on the representation chosen for the data (specifically which numeric base the transformer was provided), a sequence-to-sequence approach, and a 4-layer model. Given the multi-step approach required to implement the sieve, achieving similar success in a single or two-layer model may not be conceivable, but working with a simpler model allows for simpler deconstruction of the underlying processes.

3 Experiments and Analysis

3.1 Prime Numbers Classification

3.1.1 Experimental Setup

For the algorithmic task of primality classification of a given number, we create a synthetic dataset using *Sympy's sieve of Eratosthenes*. We use a 1-attention layer decoder-only transformer model with 32 heads and 128 hidden units to classify the prime numbers as $\{0, 1\}$. We try to train the model until it undergoes the Grokking phenomenon or a phase transition.

3.1.2 Observations

We noticed that the input has to be typecasted to `float64` for the model to "grok" or even undergo a phase transition. For other types, such as type `int` and type `long`, the model would showcase overfitting, and the phase transition state would not happen. The model trained with `float32` input has sudden loss spikes that appear to be cyclic in nature, observed before by Thilak et al. (2022). Upon further investigation, we find that this is due to the fact that the model wants to undergo a phase transition but is unable to do so due to numerically unstable gradient values. At 10^{-5} to 10^{-7} loss ranges, very small numbers are fed into log softmax. The loss spikes almost exclusively appear at the onset of a phase transition. This can be further examined by the gradient norm and weight norm of the last layer of the model. This behavior was very prominent when we extrapolated the experiment from finding primality to coprimality and further when we generalized modular addition from two operands to an arbitrary number of operands. Take a look at Weights & Biases report for more details. Notice how gradients change at the onset of a phase transition in the report.

3.1.3 Result

To our extensive experimentation, the transformer model didn't grok for the primality test with infinite training input. To our knowledge, there is a deterministic algorithm primality-proving algorithm called the AKS primality test. The AKS primality test is based on the following theorem:

> Given an integer $n > 1$, the integer n is prime if and only if the polynomial congruence relation $(x + a)^n \equiv x^n + a \mod n$ holds for all a with $1 \leq a \leq \lfloor \sqrt{\phi(n)} \log_2(n) \rfloor$.

To the best of our knowledge, the 1-attention layer decoder-only transformer model is not capable of doing recursive computation. This goes along with the fact that transformers can't reduce fractions to their lowest form. This is the most probable explanation for why they cannot compute GCD, which requires reducing fractions to it's lowest form. GCD only "groks" when the numbers are presented in a base that has many frequent prime factors (For example, base 420 hits 96.8% accuracy on GCD task) trained on a log uniform distribution of training data.

Overall, the lack of inherent structure is why Transformer can't learn Primality (hypotheses: Primality has no corresponding group; Thus transformers can't learn the same Chughtai et al. (2023b)). Primality is a non-local property, meaning there isn't a clear pattern in the binary representation of a prime number that a model can quickly learn. Unlike addition tasks (where the "Grokking" phenomenon occurs), there's no underlying structure to exploit.

3.2 GCD

In our replication of the literature Charton (2023), we modified the original approach by using a fixed dataset instead of an infinite generated dataset. We trained an encoder-decoder Transformer and structured GCD calculations as a supervised translation task, where pairs of inputs and their labels, $((a, b), \gcd(a, b))$, with $a, b \in \mathbb{N}$, were randomly generated. These pairs were encoded as sequences of digits in base 10, with a sign token serving as a separator. Our evaluation used a stratified test set with GCD values uniformly distributed between 1 and 100. Contrary to the original paper, we did not observe the Grokking phenomenon for base 10, where the model initially learns the products of small powers of primes dividing in base B. This phenomenon, characterized by a sequential understanding of small powers of prime non-divisors leading to a sudden boost in test accuracy, was not evident in our experiments. Instead, our model consistently demonstrated overfitting throughout the training process. Following a suggestion from the original paper, we sampled a and b from a log-uniform distribution, a method purported to enhance the occurrence of the Grokking phenomenon. However, under this altered setting, the phenomenon remained elusive in our observations. Due to limited computational resources, both settings were trained for base 10 each with 2000 epochs. Here is the link to Weights & Biases report for further details regarding our experiments.

3.3 Exponentiated Modular Addition

3.3.1 Experimental Setup

We extended the form of the modular arithmetic function to an exponentiated functional form,

$$z = (x^a + y^b) \bmod p \tag{1}$$

for $a, b, x, y \in \mathbb{N}$ and $p \in \mathbb{P}$. The value $p = 113$ was primarily used across experiments, but other values of p were explored, including $p = 101$ and $p = 103$. The range of x and y within the dataset is chosen according to the value of p: $x, y \in [0, p)$. Due to restrictions on the size of the output of $x^a + y^b$, the values of a and b are limited to the range $[1, 9]$. The ground truth values z are used as labels.

Similar to primality classification, we use a one-layer, decoder-only ReLU transformer with 4 attention heads, a 128-dimension token embedding, and 512 hidden units in the MLP. Bias terms are disabled from the model configuration to facilitate interpretability in the downstream analysis. The learning set was limited to 30% of the entire dataset, from randomly chosen pairs (x, y) and labels z. Tests were run on varying epoch length, but typically range between 30K and 40K epochs. Please check this link to see more on different experiments we performed.

3.3.2 Observations

Not all (a, b)-pair experiments were found to grok under the initial choice of $p = 113$. However, the prototypical grokking pattern was observed for various symmetric $(a = b)$ and asymmetric $(a \neq b)$ exponent pair choices (see Table 1 in Appendix). We notice that, when grokking does occur, the speed of grokking increases as the values of a and b increase. The drop in the number of epochs needed before grokking is most significant in the test runs between $a, b = 1$ to $a, b = 3$.

We also observed that for values of a and b larger than 1, the distribution of z values is pivotal for the phase change transition to happen. We found that most choices of symmetric values for (a, b) will grok, except for $a, b = 7$. Examining this outlier, we found that the distribution of label values is uneven, with many values in the $[0, p)$ not represented in the output. On the other hand, for pairs like $a, b = 3$, the function outputs all values in the range $[0, p)$ equally.

For other values of a and b that did not grok, the distribution of remainders is uneven, having a large number of (x, y) pairs that produce outputs with value $z = 0$ and/or various other potential values of z are not represented. Changing the value of p such that the distribution of z is evenly distributed results in successful grokking. If $p = 113$, grokking does not happen for $a, b = 7$, while the choice of $p = 101$ results in the model successfully grokking. For the model to generalize, it would seem that the result of $x^a + y^b \bmod p$ has to have a uniform distribution of remainders. Applying different values of p to experiments that fail to grok results in successful generalization when the distributions of the residues are more uniform.

3.3.3 Result

First, our observations suggest that the underlying mechanism has changed relative to the original functional form of modular addition. The only mechanism that was found unchanged was the mechanism pertaining to the unembed phase, in which the algorithm maps the result of z to a number in the input vocabulary. This is in line with the strong universality hypothesis, which states that specific mechanisms are reused across related tasks. All results z would have to be mapped from the its representation on a circular group to the real numbers (i.e. reverse Fourier transform), due to modular operation. We see that key frequencies in the Fourier basis as still found in the unembed matrix (see Appendix: W_logits in the Fourier Basis).

Attempts to identify a simple representation of the basis were inconclusive across all novel choices of (a, b) except for $(a, b) = (2n, 2n)$, for $n \in \mathbb{N}$ (suggesting some regular structure). Even in these cases, the basis is spread out widely but evenly (equal amplitude) across the majority of frequencies, instead of being sparse in a few key values (suggesting some regular structure). In other cases, there is no simple functional structure that is apparent (See appendix: Embedding in Fourier Basis). Moreover, it is clear that angle-sum identities are no longer being formed as evidenced by the norms of neuron activations (See appendix: Norms of neuron activations by Fourier Component). Instead, the algorithm depends on transforming the representations of x and y separately then combining those results in the unembed phase.

3.4 Sorting

3.4.1 Experimental Setup

We conducted an experiment with a single-layer attention-only Transformer model, focusing on two distinct tasks: sorting lists of both fixed and variable lengths. Our training and testing sequences began with a special START token, followed by an unsorted list, a MID token, and then the sorted list. The model achieves near-perfect validation accuracy within 50,000 epochs for both tasks without any signs of overfitting. This was consistent even when dealing with lists of up to 100 items.

3.4.2 Observations

In our analysis, we observed a consistent pattern across the attention heads for both tasks. They showed a marked tendency to focus on the tokens immediately following the MID token, with the intensity of attention decreasing in proportion to the distance from the MID token. A significant observation was the attention heads' apparent reliance on a copy mechanism for output prediction. This was evidenced by a pronounced diagonal line in the *OV circuit* Elhage et al. (2021) heatmap, where values were considerably higher than in other areas. We further quantified this observation by examining the positive eigenvalues of the *OV circuit*, or the "copying score," which was around 0.6 to 0.9. This score corroborates our earlier observations about the leveraged mechanism.

Integrating a feedforward network enhances the complexity of the mechanism, leading to a significant reduction in its tendency to duplicate information. However, upon examining the weights of the neurons, it appears that many are redundant. Employing just a single attention head is adequate for accomplishing the task. Additionally, our findings indicate that while positional embedding isn't essential for tasks with or without repetition, it does facilitate faster convergence. Another interesting observation is that with the same hyperparameters, the model performs better on sorting the variable length list than the fixed length list.

We also trained a model on data with lists containing some repeating tokens with similar results. The main difference was that some attention was paid to the same tokens as the previous token. This was modulated by positional embeddings: the attention from tokens at the end of the list is smaller than the tokens at the beginning, even when they have the same values.

4 Conclusion & Future Work

Our exploration of the capability of small transformer models to learn and generalize across a variety of mathematical problem types met with mixed results but agreed with general hypotheses in the field of mechanistic interpretability. Our exponentiated modular addition experiment, as well as our sorting experiment, saw the small transformer model find efficient solutions. Attempts to solve the GDC task in a 1-2 layer model were not successful and begs the question of whether a smaller model than that which is given in the original paper can achieve the same task. Similarly, attempts to find a transformer model that can generalize (or at least do better than chance) for prime categorization were unsuccessful, at least given the limited model size we chose to work with.

As shown in Vaswani et al. (2017), transformers are great for tasks such as translation or summarization. These tasks utilize their attention mechanism, especially long-range dependencies, to find separate patterns in parallel. Local consistent patterns across wide ranges of input are similarly reinforced during learning, making natural language models successful in tasks involving local consistency, even at a small model scale. We can see that only one attention head is required to deal with the relative magnitude between two numbers in sorting. When these local consistencies form a group-theoretical logic, the model weights are led to a simplifying representation. When compared to tasks with non-local relationships (i.e. prime classification), our successes differ significantly from that of modeling "grouped operations." This lends credence to the weak hypothesis of universality, wherein self-consistent underlying principles are learned as opposed to converging upon a global solution.

The Transformer is very efficient in natural language tasks due to the attention mechanism, but even at scale, the model still cannot emulate model-checking algorithms properly Madusanka et al. (2023). It has been noted that neural networks would fall short in mathematical tasks that require generalizing beyond a fixed vocabulary (i.e. range of numbers), as numerical representations would fail to generalize beyond the training data range Trask et al. (2023), given that symbolic representation and interpolation is known to be achievable, but not extrapolation.

For future analysis, we want to explore more different group-mathematical operations in order to build further upon our hypothesis. Further, a revisiting of exponentiated modular addition would also be required to learn about the underlying mechanism of the Transformer model. We want to get an answer on how it was able to extract patterns and experience ”Grokking” on the same exponent numbers. It would also be interesting to see if a smaller transformer model can solve GDC. This would allow us to gauge how large a model needs to be before it is capable of generating certain structures for more advanced algorithmic tasks. Similarly, it would be interesting to explore a more complex model configuration to see if mechanisms for prime categorization can emerge in such a setting.

References

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. In *Proceedings of the National Academy of Sciences*, 2019. URL https://www.pnas.org/doi/full/10.1073/pnas.1903070116.

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.

Francois Charton. Can transformers learn the greatest common divisor? In *N/A*, 2023. URL https://arxiv.org/pdf/2308.15594v1.pdf.

B. Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. *International Conference on Machine Learning*, 2023a. doi: 10.48550/arXiv.2302.03025. URL https://arxiv.org/abs/2302.03025v2.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *N/A*, 2023b. URL https://arxiv.org/pdf/2302.03025.pdf.

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.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv: 2307.03381*, 2023. URL https://arxiv.org/abs/2307.03381v1.

Tharindu Madusanka, Ian Pratt-hartmann, and Riza theresa Batista-navarro. Identifying the limits of transformers when performing model-checking with natural language. In *N/A*, 2023. URL https://aclanthology.org/2023.eacl-main.257.pdf.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *N/A*, 2022. URL https://arxiv.org/abs/2201.02177.

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. URL https://arxiv.org/abs/2206.04817v2.

Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *N/A*, 2023. URL https://ar5iv.labs.arxiv.org/html/1808.00508.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *N/A*, 2017. URL https://arxiv.org/abs/1706.03762.

5 Appendix

The AKS algorithm is as follows:

Algorithm 1 AKS Primality Test

```
1:  procedure AKSPrimalityTest($n$)
2:      if $n$ is a perfect power then
3:          output composite
4:      $r \leftarrow 1$
5:      while $r < \log^2(n)$ do
6:          if $\gcd(r, n) \neq 1$ then
7:              skip this $r$
8:          for $a \leftarrow 2$ to $\min(r, n-1)$ do
9:              if $a|n$ then
10:                 output composite
11:         if $n \leq r$ then
12:             output prime
13:         for $a \leftarrow 1$ to $\lfloor \sqrt{\phi(r)} \log_2(n) \rfloor$ do
14:             if $(X+a)^n \neq X^n + a \mod X^r - 1, n$ then
15:                 output composite
16:         output prime
```

This is deterministic algorithm that is used to find primality of the number. This was the original algorithmic task that the white box transformers was suppose to, we further investigate that subroutines of the primality algorithm: modulo operation and GCD operation are grokkable or not.

(a, b)	**Prime** p	**Grokked**	**Grokked Epoch**
$(1, 1)$	113	Yes	18100
$(3, 3)$	113	Yes	10000
$(3, 4)$	113	Yes	70000
$(5, 4)$	113	Yes	5000
$(5, 5)$	113	Yes	3000
$(7, 7)$	113	No	N/A
$(7, 7)$	101	Yes	2500
$(9, 9)$	113	Yes	1600
$(4, 5)$	113	Yes	1100
$(8, 9)$	113	Yes	600

Table 1: Results of whether model grokked on $(x^a + y^b) \bmod p$

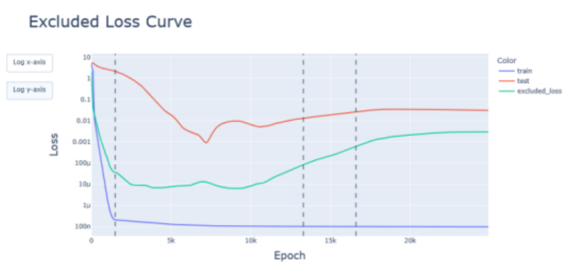


Figure 2: Progress Measure of Excluded Loss

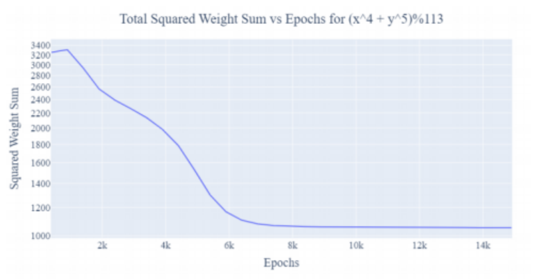


Figure 3: Total Sum of Squared Weights indicate progress in grokking

Figure 1: Progress Measures

For the above figure: We can see that the total sum of squared weights, goes from high(when the model is memorizing and overfitting) to low(when the model after grokking, generalized and now the weights are sparse).

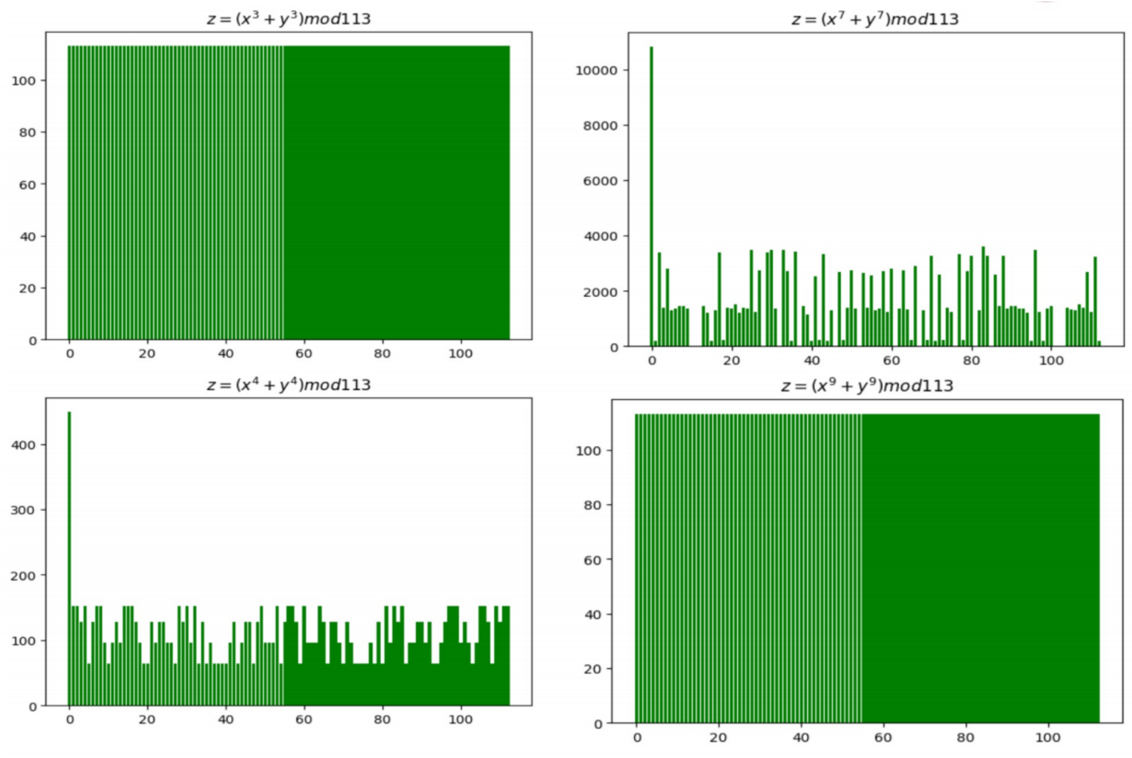
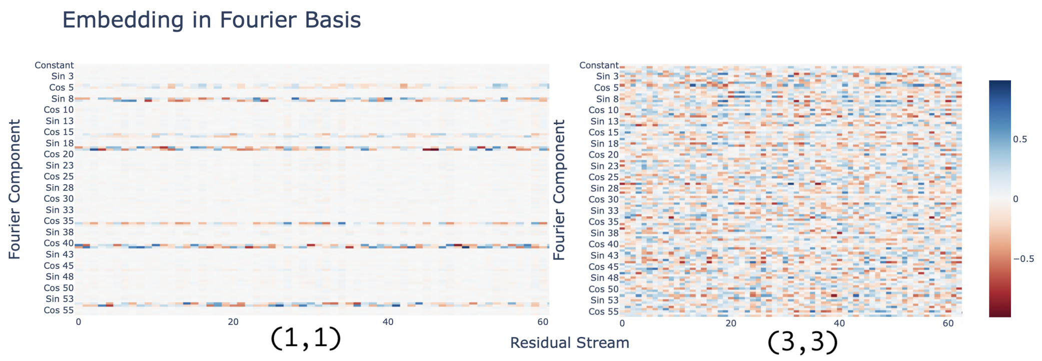


Figure 2: Distributions of the remainders z



For figure 2: The result $z = (x^a + b^y)\%p$ are observed to find the correlations between values (a, b) that grokked and values that didn't. As showcased, z for values that grokked has an even distribution, while the values that did not grokked have a very uneven distribution, with emphasis on 0.

5.1 Prior Hypothesis

In Neel's paper with the task of $(a + b) \mod 113$, the model was found to independently uncover the Discrete Fourier Transform. This result indicates an early understanding of trigonometric concepts during the training phase. My hypothesis, largely based on observing the model's immediate and accurate generalization for cases where m equals n, suggests that this occurs early in training, likely as the model begins to grasp trigonometry. I propose that the model extends its trigonometric knowledge to the Polar Coordinate System. Notably, in situations where m and n are both 4, the model showed instant and precise generalization. In more complex scenarios like $m = n = 9$, the model initially overfitted but then quickly adapted and generalized perfectly. This was particularly surprising as the model performed better in this more complex setting than in a simpler one ($m = 4$, $n = 3$).

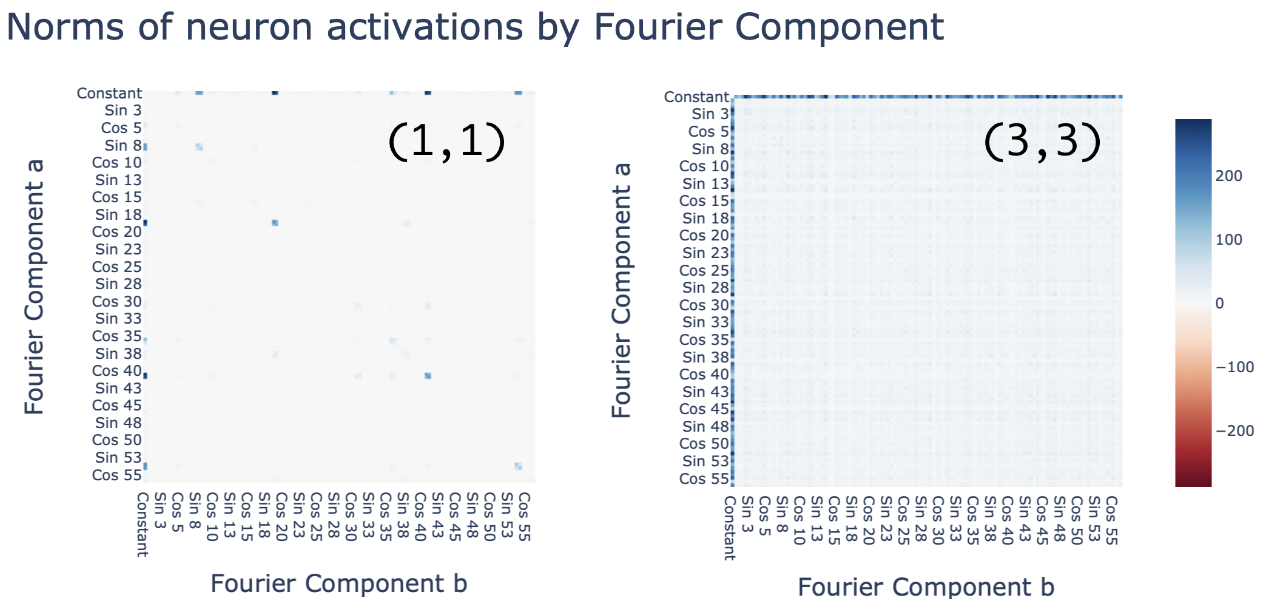
5.1.1 Cases

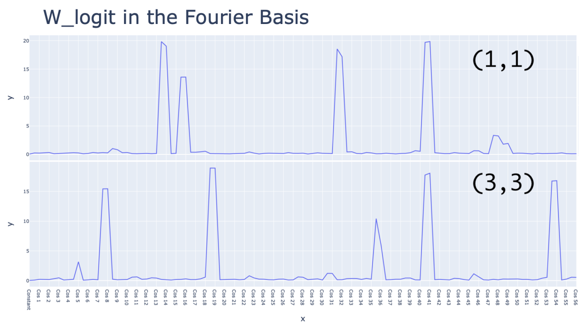
1. **There is a geometric elegance when m equals n for the expression $(a^m + b^n)$.** This elegance arises from how the expression, in these cases, maps onto the Polar Coordinates. Specifically, it leverages the rotational symmetry inherent in complex numbers. De Moivre's formula, which extends Euler's formula, aptly describes these rotations in the complex plane. It suggests that the model intuitively understands and applies complex number theory, particularly the geometric aspect of rotating points in the complex plane, to efficiently predict these expressions.

2. **When m is an integer multiple of n ($m = kn$), the computation of $(a^m + b^n)$** also has a geometric interpretation. This scenario unfolds within the domain of complex numbers, where the powers of complex numbers exhibit specific geometric patterns. In these cases, the repeated multiplication implied by the exponentiation can be visualized as scaled rotations in the complex plane.

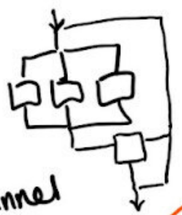
3. **In scenarios where m and n do not align with the special conditions of being equal ($m = n$) or one being an integer multiple of the other ($m = kn$), computing $(a^m + b^n)$** poses significant challenges. Without the geometric clarity provided by rotational symmetry in the complex plane (as seen in the $m = n$ or $m = kn$ cases), the model lacks a straightforward complex number method for prediction. Algebraically, the task becomes increasingly complex with larger exponents, as no simplifying symmetries or patterns emerge to ease computation. From my observations, the absence

of these specific conditions leads to slower generalization and difficulty in accurately predicting $(a^m + b^n)$, unlike in cases where m and n adhere to these special conditions.





Concrete (ish) proposal v0



① Break down NN into chunks based on a priori knowledge of architecture
  ↳ e.g. {seperate attention heads, MLPs}

② See each chunk as an information processing channel that looks at some subset of the global state information

in Transformer: residual stream

Key questions:
- → what info does the channel look at
- → how is the info encoded by the channel
- → what is the true channel capacity

possible link to existing model complexity measures / RLCT in SLT etc

How do we figure this out??
- → look at input/output activations of this chunk
- → look at gradients (input/output Jacobian of this model part)
- → find optimal principal basis for looking at the channel
- → could involve some nonlinear transform of global state
  - ↳ e.g. could view channel as operating on the top n fourier transform frequency components (example, not specifically this)
- → possibly second order information / Hessians / approximation (highest curvature directions wrt loss (at minimum) most important?)

To do this you want to look at the channel's input-output behavior on the data distribution and not just capture a bunch of outputs and find an encoding that "seems interpretable"

We do step ② taking into account the full training distribution (to optimally compress model)

The channel is trying to encode information optimally for further processing and not for you to directly read off and interpret → not necessarily sparse!

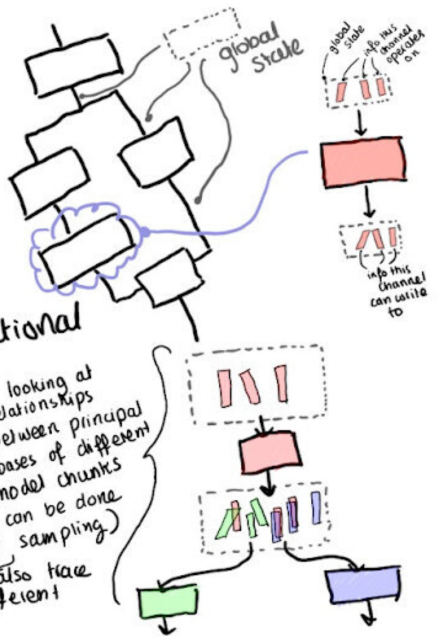
Extension:
- do all this at different levels of initial coarse graining to see different subchannel decompositions
- Do step 1 in a way that also uses algebraic information (instead of just knowledge of architecture) (segment weight space in a principled way)

③ We should now have a coarse model compression consisting of a bunch of function approximations that read and write from the same global state, and a set of principal bases / "subchannels" of this global state for each function/channel that tell us what subset of the info is being operated on
(fuzzy terminology sorry)

↓ could be a set of optimal input/output bases for each channel (so a connection between channels can be seen in terms of the output of one or the input of the other)

⇒ We now sample this compressed computational graph using our knowledge of the data distribution to figure out a mapping to concepts we care about

see which pathways / connections activated for different types of data / tasks



looking at relationships between principal bases of different model chunks (this can be done without sampling)

but then we can also trace paths through on different subdistributions

(massive)

- NNs can be represented as more compressed, modular computational graphs

That are
- easier to "interpret"
  ‖
  read:
  → gain more confidence on function behavior as compared to black-box testing or unprincipled gray-box testing
  → secondary: understand / conceptualize function being implemented

objects that can be used for heuristic arguments

possible outcome: humans cannot understand but we can build machines / algos that can interpret the compressed graph with more guarantees

Key choice: how to represent variables and functions in the code? /state

Why may compression be easier than you think?

→ we know the shape of the training distribution / data and have a bunch of existing abstractions / compressions for that data!

→ we impose many constraints and a strong prior on the shape of the function being implemented via the NN architecture / parameterization

choice of activation function, network not fully densely connected, intrinsic parallelism, e.g. attention heads

→ we can probe internals of models to see intermediate representations, get gradients via backprop, etc

→ the world is modular - it's useful to think in terms of higher-level modular abstractions & concepts

→ modular (parallelized or / composition of functions that incrementally improve performance in series) algorithms are much easier to fin via any form of greedy search, e.g. SGD

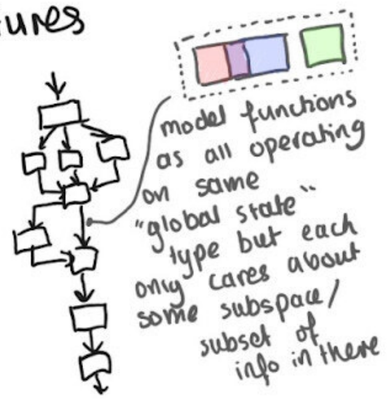
analogous to binary → code decompiling

Why is code more interpretable?
What makes code "good"?
↳ a composition of a bunch of functions that have nice type signatures

Information theoretic framing.

Model is composed of a bunch of information processing channels - we want to approximate what information they encode and how → need to see both inputs & outputs to do this! We also want to use knowledge of input distribution, not all poss. inputs!

flaws of dictionary learning


model functions as all operating on same "global state" type but each only cares about some subspace / subset of info in there

Notable Weights and Biases reports:

1. Reproduction of Neel Nanda's paper with parameter and weight visualization throughout the training: Link
2. Attempt to replicate results from Charton (2023): Link
3. W&B report for Grokking in Modular Addition with Exponents: Link