Opening the Black Box: Mechanistic Interpretability of Transformer Models in Modular Subtraction

Elon Abergel
Department of Computer Science
Yale University
elon.abergel@yale.edu

Raja Moreno
Department of Mathematics
Yale University
raja.moreno@yale.edu

Daphne Raskin
Department of Computer Science
Yale University
daphne.raskin@yale.edu

Elder Veliz
Department of Statistics
Yale University
elder.veliz@yale.edu

1 Introduction

1.1 Background

Recent advancements in neural networks have led to their successful application in a variety of fields like computer vision(1), natural language processing(2), and medicine(3). As such, these networks have become increasingly being employed when making critical and ethically complicated decisions, despite their black-box nature. This raises significant interest in the field of mechanistic interpretability, which is a subfield of interpretability that seeks to understand how and why models become "good" at solving particular tasks over time. It is widely believed that neural networks solve problems through two main methods: memorization and generalization(4). However, the specific internal processes that enable these models to memorize and generalize remain largely unknown. Mechanistic interpretability researchers seek to understand how models work and verify their accuracy.

"Grokking" is a surprising phenomenon for researchers studying this field. It describes how a model moves from merely memorizing data to generalization after extensive training on a consistent dataset. Importantly, grokking is distinct from the machine learning phenomenon "double descent," in which the model testing error initially decreases as the model complexity increases, then increases reaching a peak (where the model complexity allows it to perfectly fit the training data), and finally decreases again as model complexity continues to grow(5). Grokking, on the other hand, describes a model that generalizes *un*predictably, long after overfitting its training data—when no further significant learning improvements would be expected(6).

1.2 Project Scope

Nanda et al. (2023) (4) describe how a one-layer transformer model learns an unusual algorithm—the "Fourier Multiplication Algorithm"—for adding two integers mod p when trained on specific examples of modular addition. Building on their methods, we aim to apply a similar reverse-engineering approach to understand and discover if, when trained on examples of modular subtraction, the same network architecture learns distinct algorithms for each operation, or if it learns one unified algorithm.

Our project aim is two-fold: (1) to extend the work of Nanda et al. (2023) to modular subtraction and (2) to gain a deeper understanding of grokking within transformer architectures—exploring methods, potential benefits, and inherent limitations of mechanistic interpretability.

2 Overview of Original Paper: Nanda et al. (2023)(4)

2.1 Motivation: The value of explaining *in one's own words*

One of our two main goals for this project is to delve into mechanistic interpretability and better understand the exciting phenomenon that is grokking. To hold ourselves accountable, we have spent many hours discussing and experimenting with the methods described in Nanda et al. To improve our scientific writing and to sharpen our understanding of grokking within transformer architectures and mechanistic interpretability, we have included our own explanation of Nanda et al.'s research methods in the following section. This background serves as context for our extension of Nanda et al.'s paper.

2.2 Grokking and Identification Methods

A classification problem requires three key features for grokking to occur. Firstly, the problem needs to be complicated enough such that the model first overfits to its training data by internally creating a memorizing circuit. Secondly, the problem's generalized solution needs to be complicated enough that it takes time for the model to optimize or align its internal components so they work together successfully, and the model should prefer to find this generalized solution if the problem includes weight decay regularization (as this will bias the model toward such general solutions). Finally, the problem must have limited data such that relying on its memorization components scales in complexity with more data, but relying on generalization components does not.

One challenge of studying mechanistic interpretability for models that grok is to determine how a model transitions from relying on its memorization components to its generalization components. In Nanda et al., the researchers used two types of progress measures to track this transition(4). The first progress measure, "excluded loss," determined the model's reliance on its generalization strategies. To measure this, the model was modified such that it could *not* use the specific key frequencies that were critical to the learned generalization algorithm (the algorithm is described in greater detail in later sections). If the model relied more heavily on its generalization algorithm than its memorization algorithm, then this modification would cause the model's loss to increase significantly. If the model relied more heavily on its memorization algorithm than its generalization algorithm, then this modification would have little effect on the observed loss. On the other hand, "trig loss" was a second progress measure used to determine the model's reliance on memorization by modifying the model such that it could *only* use the 5 relevant key frequencies of the learned generalization algorithm. If the model relied more heavily on its generalization algorithm than its memorizing algorithm, then this modification would only improve the model's performance. Conversely, if the model relied more heavily on its memorizing algorithm than its generalizing algorithm, then this modification would cause the model's loss to increase significantly.

Using these progress measures, Nanda et al. revealed that grokking does not happen suddenly, but, rather, is the result of a sequential progress by which the model (a) builds a circuit of components that memorize the data, then (b) builds a circuit to perform its generalizing algorithm, which the model is encouraged to build due to weight decay regularization, and lastly (c) gradually removes its memorization components(4).

A couple of key phases must be identified to determine that a model is grokking. One key phase is the "clean-up" phase during which the components of the model based primarily on memorization are gradually removed. Since the model becomes solely dependent on its generalization circuit components, the restricted loss will go down, but the excluded loss will increase. A second key phase is the model's demonstration of the emergence phenomenon, where models *suddenly* improve significantly. An example of this phenomenon is how models like GPT-3 demonstrate improved ability on numerous tasks like mathematical addition, news article generalization and code-writing —capabilities that smaller models do not exhibit(11). Models that grok demonstrate emergent behavior because even while the model is internally building a generalization circuit, its test performance does not decrease significantly until the model undergoes its "clean-up" phase.

2.3 Grokked Algorithm: Methods and Findings

Using several ad-hoc techniques, which we draw inspiration from, Nanda et al. identified the trigonometric-based "Fourier Multiplication" algorithm that their model learned to generalize on examples of modular addition. Specifically, this algorithm computes $(a + b)\%p$ by initially mapping

numbers a and b onto a unit circle as $\frac{a}{p}$ and $\frac{b}{p}$, respectively. Then, it combines these positions to find their sum on the circle—corresponding to the modular sum. The transformer passes $p = 113$ logits through a softmax function. It then learns to "rotate" these values counterclockwise by c—locating the maximum logit along the x-axis to determine the correct output c.

To identify which parts of their model were actively contributing to the model outputs, Nanda et al.'s team began by examining the various activations within the model: the embedding layer activations, attention patterns, MLP, neuron activations, and output activations to identify which parts of the model were actively contributing to its outputs(4). In the embedding layer, the researchers found that each number n was embedded as $\sin(w \cdot n)$ and $\cos(w \cdot n)$, where w is a weight parameter that adjusts the frequency of the sine and cosine waves, effectively mapping each number to a specific point on the unit circle(4). Nanda et al. asserts that its model projects each point (represented as two one-hot vectors, where each vector is in 114-dimensional space) onto a corresponding rotation using its embedding matrix. In this case, the embedding matrix is used to project each one-hot vector (114-dimensional space) into a subspace of the model dimensions (128-dimensional space).

The researchers used four main pieces of evidence to support their findings. First, they observed a periodic structure in the lookup table embeddings, suggesting that the model used cyclical patterns like trigonometric functions, which are essential in modular arithmetic computations (4). Second, by examining specific neurons, the team found that certain neurons were tuned to represent $(a + b)$ at unusual, specific frequencies, thus aligning neural network mappings with expected arithmetic terms (4). The "key frequencies" identified included $k \in \{14, 35, 41, 42, 52\}$. Third, deletion tests on neurons considered critical by the algorithm (such as those representing arbitrary constants like k in $\cos(14k)$) negatively impacted the model's performance, confirming these frequencies' importance in this computation. Conversely, preserving these neurons maintained low performance loss, validating their importance to the model (4). Finally, the discrete Fourier transform applied to the embedding matrix mapped each frequency's contribution to the outcomes, revealing how inputs were converted into outputs through the model's learned frequency patterns.

Nanda et al. found that their one-layer transformer model used trigonometric and periodic patterns. The methods used in their paper provided a deep understanding of how the model works inside: the model can do more than just memorize; it can apply systematic, rule-based strategies typical of modular arithmetic (4).

3 Data

To explore the mechanistic interpretability of a simple one-layer transformer architecture, we generated $12,769$ examples of modular subtraction. Each of our examples is a (1x3) matrix of the form [a, b, 113], where $a, b \in \{0, 1, 2, \cdots, 112\}$, and our chosen prime number, 113, represents the 'special' token of the equals sign.

Our modular subtraction examples take the form $(a - b)\%p =?$. For instance, take the sample $[21, 31, 113]$ in our dataset. Here, $a = 21$ and $b = 31$, so the model is prompted to solve $(21-31)\%p$, where p is some prime value. For this project, we defined a constant p value of 113. Therefore, the correct output would solve the equation $(21 - 31)\%113 = (-10)\%113 = 103$.

To generate our dataset, we used the following code to generate and stack each modular subtraction example. The a_vector is a (113x113)-dimensional vector consisting of 113 zeros, followed by 113 ones, then 113 twos, ..., all the way to 113 elements of 112. The b_vector is a (113x113)-dimensional vector with 113 concatenated copies of the sequence $0, 1, 2, ..., 112$. The $equals\_vector$ is a (113x113)-dimensional vector where each entry is 113 (again, the special "=" token id). Finally, we use torch.stack to stack the a_vector, b_vector, and $equals\_vector$ into one 113^2 x 3 tensor of every possible pairing of |a| |b| |=|.

```
a_vector = einops.repeat(torch.arange(p), "i -> (i j)", j=p)
b_vector = einops.repeat(torch.arange(p), "j -> (i j)", i=p)
equals_vector = einops.repeat(torch.tensor(113), " -> (i j)", i=p, j=p
                                    )
dataset = torch.stack([a_vector, b_vector, equals_vector], dim=1).to(
                                    DEVICE)
```

4 Methodology

To explore grokking and mechanistic interpretability, we trained a simple one-layer transformer architecture on our previously described dataset of modular subtraction examples. The following two subsections describe our dataset and training parameters.

4.1 Dataset Preparation

Our dataset comprises of a $(12,769\text{x}3)$-dimensional matrix consisting of $12,769$ examples. Our transformer model processes each row independently as a sequence of three tokens. After generating our dataset, we created our training and testing set with 30% and 70% of the dataset examples, respectively. We used a fixed random seed from PyTorch to ensure the reproducibility of the dataset shuffling.

4.2 Model Configuration

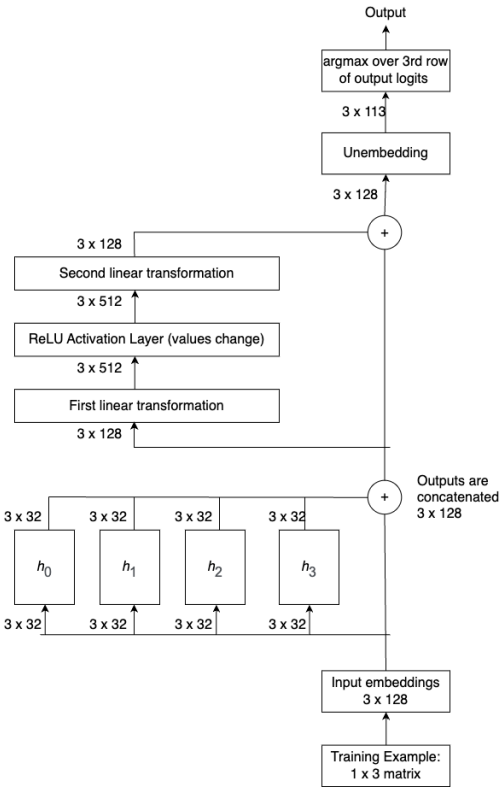
We configured our transformer model for simplicity:

- **Number of Layers**: 1.
- **Number of Attention Heads**: 4.
- **Dimension of Model**: 128 – Applies to both attention heads and the MLP layer.
- **Dimension Per Attention Head**: 32 – Computed as $\frac{128}{\text{number of attention heads}}$.
- **Dimension of MLP**: 512.
- **Activation Function**: ReLU – Introduces non-linearities to enhance model capabilities.
- **Normalization Type**: None – Omitted to maintain model simplicity.
- **Size of Vocabulary**: 114 – Includes 113 unique tokens plus one special token.
- **Output Vocabulary Size**: 113 – Corresponds to the range of output logits.
- **Context Window Size**: 3 – Each sequence processed consists of three tokens.

We used the cross-entropy loss function to train our transformer model.

To optimize our objective function, we used the AdamW optimizer from PyTorch to update the model's weights based on the gradients of the loss function during training.

Figure 1: Our Model Architecture



Our input dataset is a $(12,769\text{x}3)$-dimensional matrix. Each row is an example representing the modular subtraction problem $(a-b)\%p =?$ and takes the form $[a, b, 113]$. Each row is processed independently.

Input Embeddings: Each example has 3 tokens, and each token is represented by a 128-dimensional vector before being processed by the model. Thus, each of the 3 tokens are embedded into a 128-dimensional space.

Multi-Head Attention Layer: Since our model uses 4 attention heads, each of the tokens' 128-dimensional embedding is split among them. Each attention head operates on a portion of the embedding vector, and thus the 128-dimensional embeddings split across 4 attention heads results in each attention head getting an input of 3 tokens x 32-dimensional partial embedding. After each attention head processes, the outputs are concatenated (along the dimensional axis corresponding to the embeddings), resulting in a single output matrix of the original embedding dimension per token: $3\text{x}128$.

MLP Layer: To add some complexity to our model, we project the $3\text{x}128$ matrix into higher-dimensional space, treat the data with a ReLU function, and then project the resulting $3\text{x}512$ matrix back into $3\text{x}128$-space.

Un-Embedding: We map the $3\text{x}128$ matrix space back into the target vocabulary size (113) so that each entry in the output matrix can be interpreted as logits corresponding to each of the 3 tokens in the input example. We only care about the third row of this output matrix, which returns the output logits following the equals-sign.

5 Implementation Details

To train our model, we used full batch training rather than stochastic gradient descent. Although this approach demands more memory and computational resources per update than SGD (since the entire dataset is used to compute the gradient for each update rather than a random subset of the data), we used full batch training because it made training smoother and more stable.

We used the cross-entropy loss function to train our transformer model, as this loss function is optimized for multi-class classification problems. To implement this, we calculated the log of the softmax probabilities for the logits. Then, we selected the log probabilities that corresponded to the true class labels and calculated the mean of the negative of those correct class log probabilities across all samples. This final step averaged the negative log probabilities of the correct classes, and by negating the mean, we ensured that we could increase the chances of correctly classifying the data by minimizing the loss function.

We used the AdamW optimizer from PyTorch to update the model's weights. In our configuration, we used a learning rate of 0.001 and a weight-decay value of 1 (set high to bias the model toward generalizing). Penalizing model complexity prevented the model from permanently overfitting on the training data. To control the exponential decay rates of the moving averages of historical gradients and squared gradients, we set our beta values to $(0.9, 0.98)$. We trained for $25,000$ epochs.

To keep our model as simple as possible, we disabled the gradients for the biases and did not include a normalization layer in our model architecture. For increased precision, we used float-64 datatypes

for our output logits, as opposed to float-32 datatypes, which are often used. Empirically, this made the loss curve smoother.

6 Results

Our model grokked subtraction approximately $22,500$ epochs into training (Figure 2). It took roughly half that number of epochs—$9,400$—to grok addition. We hypothesize the model took significantly longer to grok a general algorithm for subtraction because modular subtraction is not a commutative operation (i.e., generally $(a-b)\%p \neq (b-a)\%p$), whereas modular addition is commutative (i.e., $(a+b)\%p = (b+a)\%p$), so addition had a training dataset that was effectively twice as dense as subtraction.

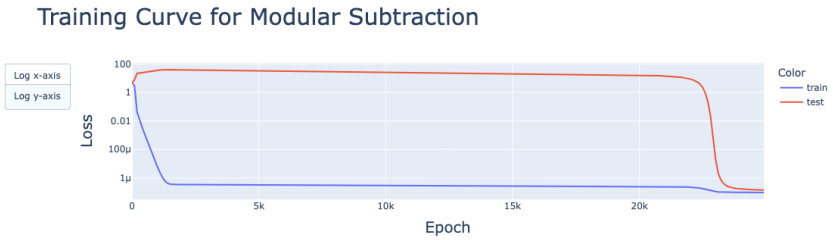


Figure 2: Training Curve for Modular Subtraction.

Our hypothesis was that Nanda et al.'s model, retooled for subtraction, would grok an algorithm for modular subtraction heuristically similar to the algorithm it grokked for modular addition. Our experiments corroborate this hypothesis. Note: For modular addition, Nanda et al.'s team found the embedding matrix W_E is sparse in the Fourier basis, only having significant nonnegligible norm at 6 frequencies. Of those frequencies, only 5 appeared to be significant, corresponding to $k \in \{14, 35, 41, 42, 52\}$. For modular subtraction, we also found our embedding matrix W_E to be sparse in the Fourier basis. However, the significant frequencies to our model include $k \in \{1, 5, 34\}$ (described in greater detail below.) Patterned after Nanda et al., we claim that our network:

1. Maps two one-hot encoded tokens a, b to $\sin(w_k \cdot a)$, $\cos(w_k \cdot a)$, $\sin(w_k \cdot b)$, and $\cos(w_k \cdot b)$ using the embedding matrix W_E where $w_k = \frac{2\pi k}{p}$ for key frequencies $k \in \{1, 5, 34\}$.
2. Computes $\cos(w_k(a-b))$ and $\sin(w_k(a-b))$ via trigonometric identities in the attention and multilayer perceptron layers:
$$\cos(w_k(a-b)) = \cos(w_k \cdot a)\cos(w_k \cdot b) + \sin(w_k \cdot a)\sin(w_k \cdot b)$$
$$\sin(w_k(a-b)) = \sin(w_k \cdot a)\cos(w_k \cdot b) - \cos(w_k \cdot a)\sin(w_k \cdot b)$$
3. For all output logits c, computes:
$$\cos(w_k((a-b)-c)) = \cos(w_k(a-b))\cos(w_k \cdot c) - \sin(w_k(a-b))\sin(w_k \cdot c)$$
via $W_L = W_{unembed}W_{out}$. Note that $\cos(w_k((a-b)-c))$ is maximized when $\cos(w_k((a-b)-c)) = 1$ (e.g., when $c = a-b$).
4. And, adds the $\cos(w_k((a-b)-c))$ together across all key frequencies to yield constructive interference over the c^* that corresponds to $c^* = (a-b) \mod p$, making the logit corresponding to c^* large, and destructive interference over other candidate cs, making their corresponding logits small.

Our prior is that, if we observe model behavior closely paralleling the behavior of Nanda et al.'s modular addition model, the algorithm it has grokked corresponds to the subtraction algorithm outlined above.

Note: Refer to our Jupyter notebook, `subtraction_grok.ipynb`*, for the code used in this analysis.*

Our first clue that our model groks our specified algorithm lies in the properties of the embedding matrix W_E. The singular value decomposition (SVD) of W_E reveals that it predominantly occupies

a six-dimensional subspace of $\mathbb{R}^{128}$ (the model's dimensionality). Notably, only six singular values exhibit meaningful magnitude (Figure 3), suggesting that the effective rank of W_E is approximately six.

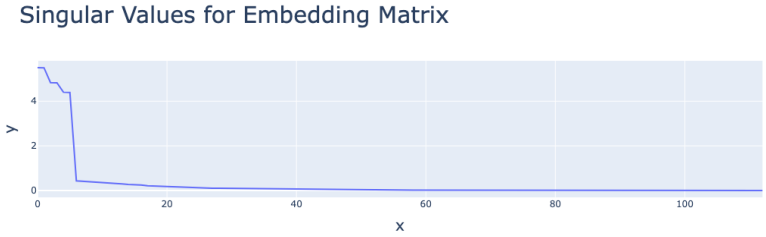


Figure 3: Singular Value Decomposition of W_E.

The SVD decomposition of W_E is illuminating, but it does not immediately tell us the *functional significance* of the principal components. To understand the nature of the six-dimensional subspace to which W_E maps, we transition from spatial to frequency domain analysis. We discover that if we embed W_E into the discrete Fourier basis, the six components corresponding to $\sin$ and $\cos$ functions of the key frequencies yield significantly larger norms (Figure 4).

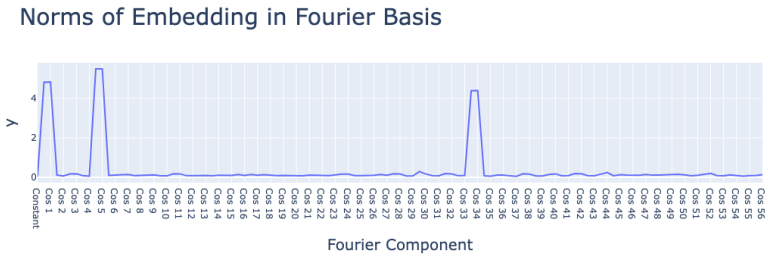


Figure 4: Norms of Embedding in Fourier Basis.

Indeed, we discover that the rows in Figure 5, which correspond to the activation over $\sin$ and $\cos$ of *all* possible frequencies only exhibit strong activation—indicated by strong coloration—over the *key* frequencies $k \in \{1, 5, 34\}$. This clear pattern highlights the model's prioritization of these frequencies, confirming their central role in the computational strategy for modular subtraction. Moreover, this supports the hypothesize that the earlier-identified six-dimensional subspace is dedicated to the $\sin$ and $\cos$ transformations at these three key frequencies. Overall, these findings imply W_E efficiently encodes the trigonometric components needed for our proposed algorithm.

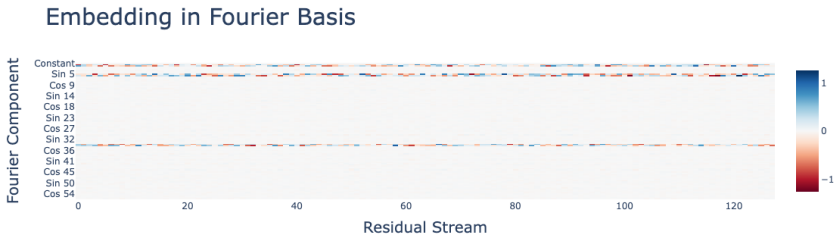


Figure 5: Embedding in Fourier basis, the result of multiplying the Fourier basis matrix by W_E.

Therefore, we claim the embedding matrix W_E is pivotal in transforming the input space into the trigonometric domain, which is necessary for our specified algorithm hypothesis to hold. Specifically, W_E maps our two one-hot encoded tokens a and b to the trigonometric functions of their respective

projections: i.e., $\sin(w_k \cdot a)$, $\cos(w_k \cdot a)$, $\sin(w_k \cdot b)$, and $\cos(w_k \cdot b)$ for key frequencies $k \in \{1, 5, 34\}$ and $w_k = \frac{2k\pi}{p}$. (This is the first step of our proposed algorithm.)

We further validate that the algorithm relies on these key frequencies by calculating two loss metrics: excluded loss and restricted loss (Figure 6).

Excluded loss is a measure of the model's performance when the key frequencies are specifically ablated. By ablation, we mean that we subtract off the components of the neuron activations in the MLP that are in the direction of $\sin(w_k(a-b))$ and $\cos(w_k(a-b))$ for our key frequencies k, suppressing the model along those frequencies. We found that the excluded loss initially decreases as the model starts memorizing the training data; however, as the model transitions to generalizing based on the key frequencies, the loss steadily increases. This aligns with our expectations; by removing the very frequencies that the model relies on for generalization, we inherently degrade is performance, which shows the significance of the key frequencies in the model's learning process.

Conversely, restricted loss is a measure of the model's performance when all frequencies *except* the key frequencies are ablated. We observe that the restricted loss remains relatively constant and below the test loss during the initial training phase. Notably, as the model approaches the grok and therefore begins to generalize effectively, the restricted loss drops *with* the test loss. This is also expected, as the restricted-loss model does not have enough complexity to overfit to the training data. Restricted loss drops in tandem with test loss when the larger model sheds its memorized values and generalizes, demonstrating that the model generalizes along the key frequencies as predicted.

This analysis with excluded and restricted loss analyses therefore reinforces the significance of key frequencies in guiding the model from simple memorization to robust generalization.

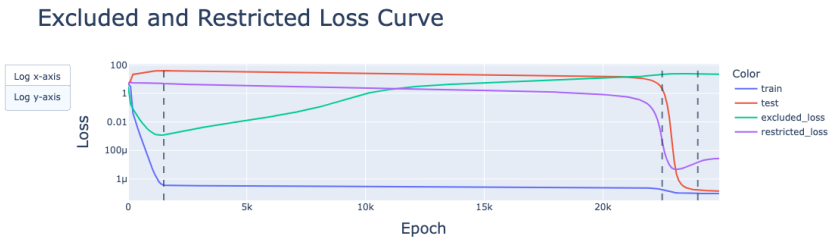


Figure 6: Excluded and restricted loss.

We now proceed to analyzing the neuron activations in the last layer of the MLP—focusing on how the activations along various frequencies contribute to the overall response of each neuron. Importantly, the fraction of activation attributed to any given frequency is only significantly large for the key frequencies $k \in \{1, 5, 34\}$, indicating a focused response to these specific components of the input data (Figure 7).

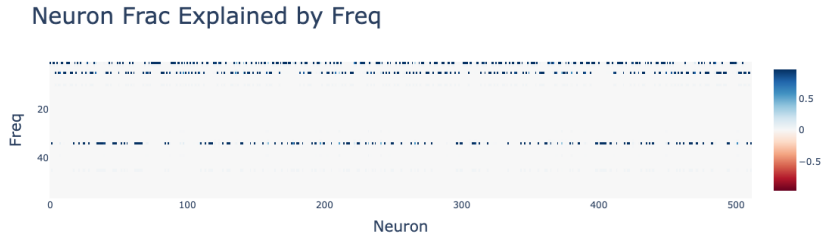


Figure 7: Fraction of Neuron Activation in W_L Explained by Frequency.

To verify that our model actually learns and applies the last step of our specified algorithm, we investigate the cosine similarity metrics—measuring the alignment between the logits predicted by our algorithm and those produced by the model. Over the course of training, there is an increase in

cosine similarity between the predicted logits and the model-produced logits, indicating increasingly strong alignment with the theoretical predictions. Concurrently, the cosine similarity between the residual (the difference between the model-produced logits and the algorithm-approximated logits) decreases, suggesting that the model's outputs are increasingly explained by our algorithmic approach (Figure 8).

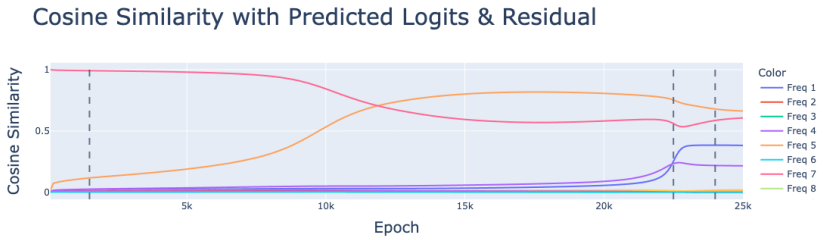


Figure 8: Cosine Similarity with Predicted Logits and Residual.

Note that the prominent orange (frequency 5), blue (frequency 1), and purple (frequency 34) lines represent how aligned the model-produced logits are with the logits predicted by evaluating the formula at each key frequency. The red (residual) line represents the similarity between the model-produced logits and the residuals. At the beginning of training, almost nothing about the model-produced logits can be explained by the algorithm-predicted logits at the key frequencies. But by the end of training, the cosine similarity between the model-predicted logits and the residual logits decreases significantly, while the cosine similarity between the logits predicted by evaluating our algorithm at the key frequencies increases. This finding supports step three of our proposed algorithm.

The evidence supporting from steps 1 through 3, coupled with the observed trends in cosine similarity, strongly suggests that our subtraction model synthesizes the results in a manner analogous to the addition model previously studied. This conclusion supports our hypothesis that the model adopts a consistent algorithmic strategy for handling modular subtraction, further demonstrated by its ability to generalize effectively to unseen data using the learned key frequencies.

7 Discussion

Our goals for this project were two-fold: to extend the work of Nanda et al (2023) (4) to modular subtraction, and to gain a deep understanding of grokking within transformer architectures (investigative methods, benefits, and limitations).

We come away from this project with a much deeper understanding of grokking, transformer architectures, and mechanistic interpretability. Through weekly team discussions, individual study, and our research, we gained the ability to chart how training data flows through our transformer model, identify key frequencies and principal components of our model's learned algorithm, and gain experience making sense of the inner mechanisms of a toy model. This was an exciting project for us; we enjoyed extending cutting-edge research, delving into the math of transformer architecture, and communicating our work through this report.

Through our research project, we also read other recent papers in the field of mechanistic interpretability that aided our research process by providing us with helpful background in relevant topics. One such paper was "Towards Automated Circuit Discovery for Mechanistic Interpretability," a paper where researchers divided neural networks into smaller components by identifying specific subgraphs or circuits for different tasks (7). They used methods like activation patching and changing the dataset to see how different network parts affected the model's overall behavior, allowing them to understand how the components fit together. Their algorithm, the Automated Circuit Discovery and Control (ACDC) algorithm, is useful and interesting in that it allows component identification within large networks easier, by breaking those networks into smaller components (7). Another paper of interest was the "Scale Alone Does Not Improve Mechanistic Interpretability in Vision Models," which discussed how making neural networks bigger does not make them easier to understand. The paper shows that human checks and new automated tools can help clarify such models (8). For example,

new research like "Brain-Inspired Modular Training" (BIMT) uses a fresh way to train where neurons are set up in a geometric space and taught with a special loss function to keep connection costs low. This method helps build more modular and straightforward networks, enhancing their interpretability. This approach is especially good at making the structure of models simpler and clearer across different tasks, which also makes them easier to understand (9). This approach ties into the insights from "Mechanistic Interpretability for AI Safety," which stresses the importance of understanding the internal workings of AI to ensure they align with human values and are safe for deployment (10).

One significant limitation of mechanistic interpretability, and specifically Nanda et. al's research (4) and our own, is that we use incredibly simple models to solve simple tasks to enable the human researchers (us) to understand what is going on. Thus, the techniques employed in our research would not necessarily work well for real-world tasks. Furthermore, the interpretability techniques used in this paper are constrained by the number and nature of features they can handle. Our model was trained on a task with 113 distinct features, a relatively small and managable number. However, real-world tasks, like visual classification or language generation, involve thousands of features interacting in complex and subtle ways. This complexity makes it challenging to apply similar mechanistic interpretability techniques effectively.

Additionally, though grokking is an interesting phenomenon, it mainly occurs in over-trained and over-parameterized models trained on very small datasets where memorization is feasible (Section 2.2 reflects further on the types of problems that lend themselves to grokking). Such conditions are rare in practical scenarios where models are trained on large datasets, such as the three billion tokens for training models like GPT-3 (8). Also, although certain measures can indicate when a model begins to grok, they cannot reliably predict when this transition will occur (7). This unpredictability and the absence of a clear method to forecast these changes reduce the practicality of using grokking as a benchmark for model performance or reliability in real-world applications.

Initially, our project ambitiously aimed to also explore how transformer models learn modular multiplication and division, alongside our study on subtraction. However, due to resource and time constraints, this part of our project was infeasible. Preliminary observations revealed that the transformer models exhibited large norms across *all* frequencies in the domain space when dealing with multiplication and division tasks—suggesting that the learning mechanisms for these operations might differ significantly from those for modular addition and subtraction. Given these initial findings, future research should focus on delving deeper into how one-layer transformers process multiplication and division. Access to more powerful computational resources will be crucial for this exploration, enabling a more detailed analysis that could uncover new insights into the learning dynamics and internal representations specific to these more complex arithmetic operations.

References

[1] Zhao, X. and Wang, L. and Zhang, Y. et al., “A review of convolutional neural networks in computer vision," *Artif Intell Rev* 57, 99 (2024). https://doi.org/10.1007/s10462-024-10721-6

[2] Xu, C. and McAuley, J., "A Survey on Dynamic Neural Networks for Natural Language Processing," *arXiv*, (2023). https://arxiv.org/abs/2202.07101

[3] Weiss, R. and Karimijafarbigloo, S. and Roggenbuck, D. and Rödiger, S., "Applications of Neural Networks in Biomedical Data Analysis," *Biomedicines*, 2022 Jun 21;10(7):1469. https://doi.org/10.3390/biomedicines10071469

[4] Nanda, N. and Chan, L. and Lieberum, T. and Smith, J. and Steinhardt, J., “Progress measures for grokking via mechanistic interpretability," *arXiv*, (2023). https://arxiv.org/abs/2301.05217

[5] Nakkiran, P. and Kaplun, G. and Bansal, Y. and Yang, T. and Barak, B. and Sutskever, I., "Deep Double Descent: Where Bigger Models and More Data Hurt," *arXiv*, (2019). https://arxiv.org/abs/1912.02292

[6] Power, A. and Burda, Y. and Edwards, H. and Babuschkin, I. and Misra V., *Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets*, (2022) https://arxiv.org/abs/2201.02177

[7] Conmy, A. and Mavor-Parker, A.N. and Lynch, A. and Heimersheim, S. and Garriga-Alonso, A., *Towards Automated Circuit Discovery for Mechanistic Interpretability (NeurIPS)*, (2023). https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf

[8] Zimmermann, R.S. and Klein, T. and Brendel, W., *Scale Alone Does not Improve Mechanistic Interpretability in Vision Models*, (2023) https://proceedings.neurips.cc/paper_files/paper/2023/file/b4aadf04d6fde46346db455402860708-Paper-Conference.pdf

[9] Liu, Z., Gan, E., and Tegmark, M., *Seeing Is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability*, (2023) https://www.mdpi.com/1099-4300/26/1/41

[10] Bereska, L. and Gavves, E., *Mechanistic Interpretability for AI Safety: A Review*, (2023) https://arxiv.org/pdf/2404.14082

[11] Zhang, M. and Li, J., *A commentary of GPT-3*, Fundamental Research, Volume 1, Issue 6, (2021), Pages 831-833, ISSN 2667-3258, https://doi.org/10.1016/j.fmre.2021.11.011. https://www.sciencedirect.com/science/article/pii/S2667325821002193