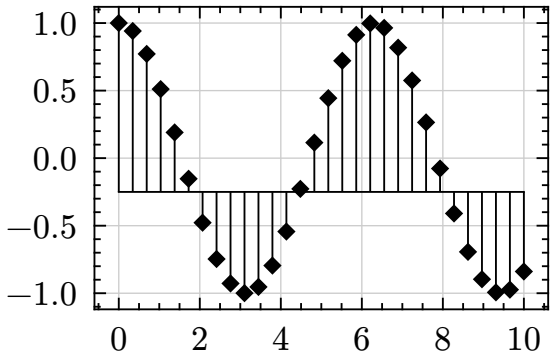
Mechanistic Interpretability on (multi-task) Irreducible Integer Identifiers

Noah Syrkis and Anders Søgaard

Abstract. This paper investigates how neural networks solve multiple related mathematical tasks simultaneously through mechanistic interpretability. A transformer model is trained on 29 parallel tasks, each predicting remainders when dividing two-digit base-113 numbers by all primes less than 113. This setup spans task complexity from binary classification (division by 2) to 109-way classification (largest prime less than 113). Further, the model, trained using gradient filtering to accelerate generalization, achieves perfect accuracy across all tasks. Embedding analysis, singular value decomposition, and Fourier analysis of neuron activations reveal complex internal representations. The model initially solves simpler tasks (modulo 2, 3, 5, and 7) before developing a shared strategy for the remaining tasks. The increased number of active frequencies in neuron activations suggests a phase where additional circuits form during generalization, some facilitating learning but not present in the final solution. Findings also confirm that amplifying slow-moving gradients significantly accelerates generalization. This study shows that multi-task learning shapes the development of internal mechanisms in deep learning models, offering insights into how these models internalize algorithms across tasks. Future research could automate circuit discovery and explore variations in multi-task learning setups. The project repository is available at https://github.com/syrkis/miiii.



1 | Introduction

Recent years have seen deep learning (DL) models achieve remarkable proficiency in complex computational tasks, including protein structure prediction [1], strategic reasoning [2], and natural language generation [3]—areas previously thought to be the

exclusive domain of human intelligence. Traditional (symbolic) programming allows functions like $f(x, y) = \cos(a \cdot x) + \sin(b \cdot y)$ to be implemented in code with clear typographical isomorphism—meaning the code's structure directly mirrors the mathematical notation. For example, in the language Haskell: `f x y = cos(a * x) + sin(b * y)`. In contrast, DL models are inherently sub-symbolic, meaning that the models' atomic constituents (often 32-bit floating-point numbers centered around 0) do not map directly to mathematical vocabulary. For reference, Appendix D shows a DL-based implementation of the aforementioned function. Indeed, the increasing prevalence of DL can be understood as a transition from symbolic to sub-symbolic algorithms.

Precursors to modern DL methods learned how to weigh human-designed features [4], with later works learning to create features from data to subsequently weigh [5], [6]—in combination with tree search strategies, in the case of games [7]. Very recent DL work has even eliminated tree search in the case of chess, mapping directly from observation space to action space [8]. Pure DL methods are thus becoming ubiquitous but remain largely inscrutable, with recent works still attempting to define what interpretability even means in the DL context [9]. Given the sub-symbolic nature of DL models, it is unsurprising that their interpretation remains difficult.

Mathematically, DL refers to a set of methods that combine linear maps (matrix multiplications) with non-linearities (activation functions). Formally, all the potential numerical values of a given model's weights W can be thought of as a hypothesis space $\mathcal{H}$. Often, $\mathcal{H}$ is determined by human decisions (number of layers, kinds of layers, sizes of layers, etc.). $\mathcal{H}$ is then navigated using some optimization heuristic, such as gradient descent, in the hope of finding a W that "performs well" (i.e., successfully minimizes some loss $\mathcal{L}$ often computed by a differentiable function with respect to W) on whatever training data is present. This vast, sub-symbolic hypothesis space, while enabling impressive performance and the solving of relatively exotic[1] tasks, makes it challenging to understand how any one particular solution actually works (i.e., a black box algorithm).

The ways in which a given model can minimize $\mathcal{L}$ can be placed on a continuum: on one side, we have overfitting, remembering the training data, (i.e. functioning as an archive akin to lossy and even lossless compression); and on the other, we have generalization, learning the rules that govern the relationship between input and output (i.e. functioning as an algorithm).

When attempting to give a mechanistic explanation of a given DL model's behavior, it necessarily entails the *existence* of a mechanism. Mechanistic interpretability (MI) assumes this mechanism to be general, thus making generalization a necessary (though insufficient) condition. Generalization ensures that there *is* a mechanism/algorithm

[1]Try manually writing a function in a language of your choice that classifies dogs and cats from images.

present to be uncovered (necessity); however, it is possible for that algorithm to be so obscurely implemented that reverse engineering, for all intents and purposes, is impossible (insufficiency). Various forms of regularization are used to incentivize the emergence of algorithmic (generalized) and interpretable, rather than archiving (overfitted) behavior [10], [11], [12].

As of yet, no MI work has explored the effect of multi-task learning, the focus of this paper. Multitask learning also has a regularizing effect [13]. Formally, the set of hypotheses spaces for each task of a set of tasks (often called environment) is denoted by $\mathcal{H} \in \mathbb{H}$. When minimizing the losses across all tasks in parallel, generalizing W's are thus incentivized, as these help lower loss across tasks (in contrast to memorizing W's that lower loss for one task). A W derived from a multi-task training process can thus be thought of as the intersection of the high-performing areas of all $\mathcal{H} \in \mathbb{H}$.

In this spirit, the present paper builds on the work of Nanda et al. (2023), which trains a transformer [15] model to perform modular addition, as seen in Eq. 1. The task is denoted as $\mathcal{T}_{\text{nanda}}$ throughout the paper.

$$(x_0 + x_1) \bmod p, \quad \forall x_0, x_1 < p, \quad p = 113 \tag{1}$$

The task of this paper focuses on predicting remainders modulo all primes q less than p, where x is interpreted as $x_0 p^0 + x_1 p^1$, formally shown in Eq. 2, and is referred to as $\mathcal{T}_{\text{miiii}}$:

$$\left(x_0 p^0 + x_1 p^1\right) \bmod q, \quad \forall x_0, x_1 < p, \quad \forall q < p, \quad p = 113 \tag{2}$$

$\mathcal{T}_{\text{miiii}}$ differentiates itself from $\mathcal{T}_{\text{nanda}}$ in two significant ways: *1)* it is non-commutative, and *2)* it is, as mentioned, multi-task. These differences present unique challenges for mechanistic interpretation, as the model must learn to handle both the order-dependent nature of the inputs and develop shared representations across multiple modular arithmetic tasks. Further, as $\mathcal{T}_{\text{miiii}}$ is harder than $\mathcal{T}_{\text{nanda}}$ the model can be expected to generalize slower when trained on the former. Therefore, Lee et al. (2024)'s recent work on speeding up generalization by positing the model parameters gradients through time can be viewed as a sum of *1)* a slow varying generalizing component (which is boosted), and *2)*, a quick varying overfitting component (which is suppressed), is (successfully) replicated to make training tractable.

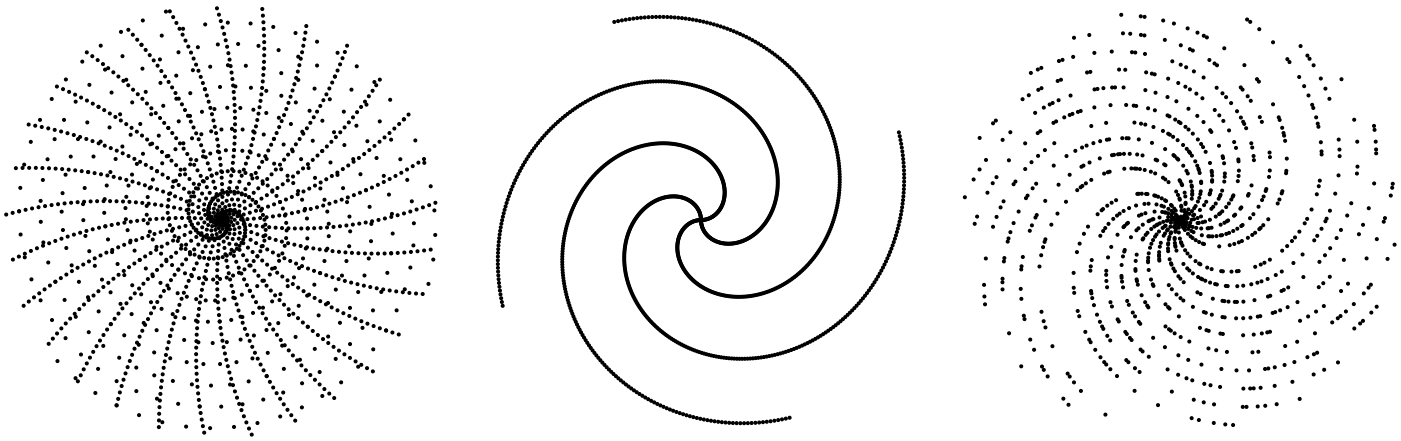


Figure 2: Visualizing natural numbers less than 12 769 in polar coordinates $(n, n \bmod 2\pi)$. Left: union of numbers with remainder 0 mod 17 and 23 (see the two spirals). Middle: numbers with remainder 0 mod 11. Right: prime numbers. It is shown here to encourage the reader to think in periodic terms.

More generally, modular arithmetic on primes is a particularly useful task for MI as it ensures uniformity among the output classes, allows for comparison with other MI work [14], and, from a number-theoretic point of view, primes contain mysteries ranging from the trivially solved—are there an infinite number of primes?—to the deceptively difficult—can all even numbers larger than 4 be described as the sum of two primes? The latter, known as Goldbach's Conjecture, remains unsolved after centuries. The choice of using every prime less than the square root of the largest number of the dataset also serves the following purpose: to test if a given natural number is prime, it suffices to test that it is not a multiple of any prime less than its square root—the set of tasks trained for here, can thus be viewed in conjunction as a single prime detection task (primes are the only samples whose target vector contains no zeros, since it is not a multiple of any of the factors q). There are about $\frac{n}{\ln(n)}$ primes less than n.

To provide insight into the periodic structure of these remainders for natural numbers less than 12 769 (and motivate thinking in rotational terms), Figure 2 visualizes various modular patterns in polar coordinates $(n, n \bmod 2\pi)$. One could imagine tightening and loosening the spiral by multiplying 2π by a constant to align multiples of a given number in a straight line (imagining this is encouraged).

2 | Background and related work

Multiple papers describe the use of deep learning to detect prime numbers [17], [18], [19]. None are particularly promising as prime detection algorithms, as they do not provide speedups, use more memory, and are less accurate than traditional methods.

However, in exploring the foundations of deep learning, the task of prime detection is interesting, as it is a simple task that is difficult to learn, and is synthetic, meaning that the arbitrary amounts of data are generated by a simple algorithm.

2.1 | Mechanistic Interpretability (MI)

MI is a relatively new field focused on reverse-engineering the internal mechanisms of neural networks. Lipton (2018) explored different definitions of interpretability in this context. MI can be contrasted with other forms of interpretability, such as feature importance analysis; while feature importance measures correlations between inputs and outputs (e.g., red pixels correlating with "rose" classifications), MI aims to understand how the model actually processes information (i.e., the mechanism).

Methods and tools used so far in MI include: Activation visualization across ordered samples; Singular value decomposition of weight matrices; Ablation studies to identify critical circuits. Conmy et al. (2023) even successfully automate circuit[2] discovery. Many reverse engineering methods from other fields, such as computational neuroscience or signal processing, almost certainly have their uses here as well.

In spite of deep learning's practical successes, uncertainty remains about its theoretical underpinnings, echoing the interpretability debate. Recent work attempts to place different DL architectures and concepts in a either geometric [21], information theoretic [22], or even category theoretic [23] context. However, no unified theory has emerged. Much interesting deep learning research thus focuses on practical, simple, or algorithmic tasks with known solutions and architectures. For example, grokking [24], the (relatively) sudden generalization after overfitting, as elaborated later, is a recent and practical discovery.

2.1.1 | Case study: modular addition

One such practical discovery is made by Nanda et al. (2023). A single layer transformer model with ReLU activation function was trained to perform modular addition ($\mathcal{T}_{\text{nanda}}$). Nanda et al. (2023)'s analysis of their trained model exemplifies MI methodology. They discovered that: *1)* The embedding layer learns trigonometric lookup tables of sine and cosine values as per Eq. 3; *2)* The feed-forward network combines these through multiplication and trigonometric identities (Eq. 4), and *3)* The final layer performs the equivalent of argmax (Eq. 5).

$$x_0 \to \sin(wx_0), \cos(wx_0) \tag{3.1}$$

[2] In the context of MI, "circuit" refers to a subgraph of a neural network that performs a particular function.

$$x_1 \to \sin(wx_1), \cos(wx_1) \tag{3.2}$$

$$\sin(w(x_0 + x_1)) = \sin(wx_0)\cos(wx_0) + \cos(wx_0)\sin(wx_1) \tag{4.1}$$

$$\cos(w(x_0 + x_1)) = \cos(wx_1)\cos(wx_1) - \sin(wx_0)\sin(wx_1) \tag{4.2}$$

$$\text{Logit}(c) \propto \cos(w(x_0 + x_1 - c)) \tag{5.1}$$

$$= \cos(w(x_0 + x_1))\cos(wc) + \sin(w(x_0 + x_1))\sin(wc) \tag{5.2}$$

2.2 | Generalization and grokking

Power et al. (2022) shows generalization can happen "[...] well past the point of overfitting", dubbing the phenomenon "grokking". The phenomenon is now well established [14], [25], [26]. Nanda et al. (2023) shows that a generalized circuit "arises from the gradual amplification of structured mechanisms encoded in the weights," rather than being a relatively sudden and stochastic encounter of an appropriate region of $\mathcal{H}$. The important word of the quote is thus "gradual".

By regarding the series of gradients in time as a stochastic signal, Lee et al. (2024) proposes decomposing the signal. Conceptually, Lee et al. (2024) argues that in the case of gradient descent, the ordered sequence of gradient updates can be viewed as consisting of two components: *1)* a fast varying overfitting component, and *2)* a slow varying generalizing components. The general algorithm explaining the relationship between input and output is the same for all samples, whereas the weights that allow a given model to function are unique for all samples. Though not proven, this intuition bears out in that generalization is sped up fifty-fold in some cases.

This echoes the idea that generalized circuits go through *gradual* amplification [14]. To the extent that this phenomenon is widespread, it bodes well for generalizable DL in that the generalizing signal that one would want to amplify might exist long before the model is fully trained and could potentially be boosted in a targeted way by the method described by Lee et al. (2024).

Perhaps the most widespread loss functions used in deep learning are mean cross-entropy Eq. 6.1 (for classification) and mean squared error Eq. 6.2 (for regression).

$$L_{\text{MCE}} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{p_{ij}} \ln\left(\frac{1}{\hat{y}_{p_{ij}}}\right) \tag{6.1}$$

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.2}$$

These have various computational and mathematical properties that make them convenient to use, though they have been shown to struggle with generalizing out-of-distribution data [27], [22].

2.3 | Multi-task learning in deep learning

As stated, multi-task learning has been shown to have a regularizing effect [13], [28] as the hypothesis W that performs well across all of the hypothesis spaces $\mathcal{H} \in \mathbb{H}$ is more likely to be general. Viewed information theoretically, this concept is reminiscent of Shannon (2001)'s asymptotic equipartition property [30], or even more generally, the law of large numbers, which states that the more samples we have of a distribution, the closer our *estimates* are to its underlying properties will align with the *true* underlying properties.

In the context of $\mathcal{T}_{\text{miiii}}$, multi-task learning is done by having the last layer output predictions for all tasks in parallel. Thus, whereas $\mathcal{T}_{\text{nanda}}$ outputs a single one-hot 1×113 vector for each of the potential remainders, $\mathcal{T}_{\text{miiii}}$, as we shall see, outputs a $1 \times q$ vector for each prime $q < p$ (i.e., 29 output-task vectors when $p = 113$). The embeddings layer and the transformer block are thus shared for all tasks, meaning that representations that perform well across tasks are incentivized.

2.4 | Transformer architecture

Transformers combine self-attention (a communication mechanism) with feed-forward layers (a computation mechanism). The original transformer-block [15] used extensive regularization—layer norm [10], dropout, weight decay, and residual connections are all integral components of the original architecture, though recent years have seen simplifications yielding similar performance [31], [32].

Input tokens are embedded into a d-dimensional space using learned token and positional embeddings:

$$z = \text{TokenEmbed}(x) + \text{PosEmbed}(\text{pos}) \tag{7}$$

Each transformer block comprises multi-head attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(Q\frac{K^T}{\sqrt{d_k}}\right)V \tag{8}$$

where Q, K, and V are linear projections of the input. Attention heads are combined through addition rather than concatenation (a transformer specific detail to align with Nanda et al. (2023)). This is followed by a feed-forward network with ReLU activation:

$$\mathrm{FFN}(z) = \mathrm{ReLU}(zW_{\text{in}})W_{\text{out}} \tag{9}$$

mapping from $d \longrightarrow 4d \longrightarrow d$ dimensions, before finally:

$$\hat{y} = zW_{\text{unembed}} \tag{10}$$

Each component includes residual connections and dropout.

3 | Methods

How exactly a given model implements an algorithm is a non-trivial question—even modular addition is implemented in a relatively obscure way [14], as per Eq. 3, Eq. 4, and Eq. 5.

This investigation probes the fundamental algorithmic structures internalized by a transformer model trained on a set of basic prime number-related modular arithmetic tasks with slight variations in complexity. This approach provides insights into how and why specific algorithmic patterns emerge from seemingly straightforward learning processes.

As stated, the setup here differentiates itself from $\mathcal{T}_{\text{nanda}}$ in two crucial ways: *1)* It is non-commutative; and *2)* It is multitask.

3.1 | Tasks

Stated plainly: the task $\mathcal{T}_{\text{miiii}}$ predicts the remainder when dividing a two-digit base-p number by each prime factor q less than p. The set of prime factors we construct tasks for is thus $\{q\} = \{q \in \mathbb{P} : q < p\}$. For $p = 113$, this yields 29 parallel tasks, one for each prime less than p. Each task predicts a remainder in the range $[0, q - 1]$. This means smaller primes like 2 and 3 require binary and ternary classification, respectively, while the largest prime less than p, 109, requires predictions across 109 classes. The tasks thus naturally vary in difficulty: predicting $\bmod 2$ requires distinguishing odd from even numbers (which in binary amounts to looking at the last bit) while predicting $\bmod 109$ involves making a selection between many relatively similar classes. From an information-theoretical perspective, the expected cross entropy for an n-class problem is $\ln(n)$, which has implications for the construction of the loss function, further discussed in Section 3.4.

Additionally, a baseline task $\mathcal{T}_{\text{basis}}$ was constructed by shuffling the y-labels of $\mathcal{T}_{\text{miiii}}$, and a task ablation test $\mathcal{T}_{\text{masked}}$ was constructed by masking away the four simplest tasks $q \in \{2, 3, 5, 7\}$.

3.2 | Data

Input Space (X) Each input $x \in X$ represents a number in base p using two digits, (x_0, x_1), where the represented number is $x_0 p^0 + x_1 p^1$. For example, with $p = 11$, the input space consists of all pairs (x_0, x_1) where $x_0, x_1 < 11$, representing numbers up to $11^2 - 1 = 120$. This yields a dataset of 121 samples. Figure 3 visualizes this input space, with each cell representing the value $x_0 p^0 + x_1 p^1$.

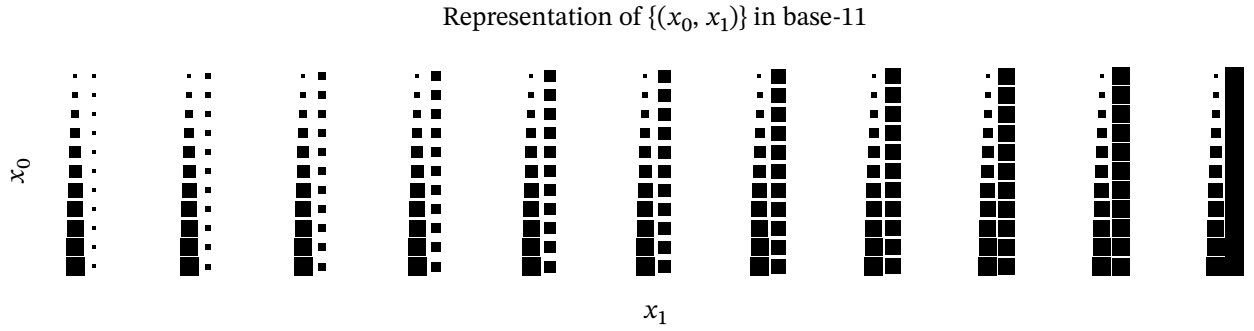


Figure 3: Visualizing X (for a small dataset where $p = 11$). Each cell represents the tuple (x_0, x_1). The top left shows 0 $(0, 0)$, and the bottom right shows 120 $(10, 10)$—both in base-11

Output Space (Y) For each input x, a vector $y \in Y$ contains the remainder when dividing by each prime less than p. For $p = 11$, this means predicting the remainder when dividing by 2, 3, 5, and 7. Each element y_i ranges from 0 to $q_i - 1$ where q_i is the i-th prime. Figure 4 visualizes these remainders, with each subplot showing the remainder pattern for a specific prime divisor. For comparison, the rightmost plot shows the output space of [14]'s modular addition task.

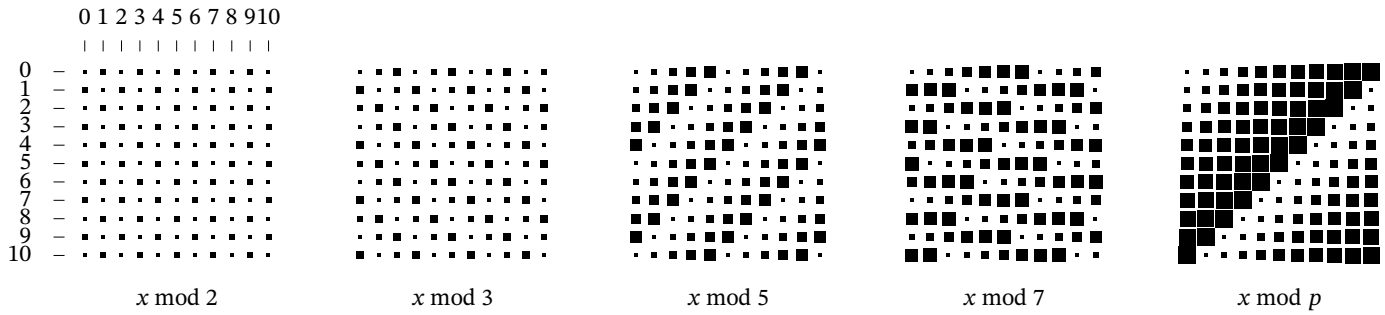


Figure 4: Visualizing tasks in Y (for $p = 11$). x_0 and x_1 vary on the two axis, with the remainder modulo $q \in \{2, 3, 5, 7\}$ indicated by the square size. Note the innate periodicity of the modulo operator.

3.3 | Model

The model follows the original transformer architecture [15] with several key design choices aligned with recent work on mechanistic interpretability [14], [16]: biases are disabled, and layer normalization is not used. The model consists of three main components: an embedding layer, transformer blocks, and an output layer. All weights are initialized following He et al. (2015). The model processes vectors of the kind seen in Eq. 11, writing the eventual result to the last position.

$$\begin{bmatrix} x_0 & x_1 & \hat{y} \end{bmatrix} \tag{11}$$

3.4 | Training

Hyper parameter optimization was conducted using Optuna [34], searching over Table 1.

dropout	λ	wd	d	lr	heads
$0, \frac{1}{2}, \frac{1}{5}, \frac{1}{10}$	$0, \frac{1}{2}, 2$	$0, \frac{1}{10}, \frac{1}{2}, 1$	$128, 256$	3e-4, 1e-4	4, 8

Table 1: Hyperparameter search space for training.

The model is trained using AdamW [35] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ following Nanda et al. (2023). To handle the varying number of classes across tasks (from 2 classes for mod 2 to 109 classes for mod 109), a modified (weighted) mean cross-entropy (Eq. 6.1) loss is created, correcting for the difference in the expected loss within each task. Note that $\mathbb{E}[L_{\text{MCE}}] = \ln\left(\frac{1}{q}\right)$, where q is the number of classes within the task in question. Correcting for this, the loss function becomes as shown in Eq. 12.3.

$$L_{\mathcal{T}_{\text{miiii}}} = \sum_{q \in \{q\}} \frac{L_{\text{MCE}_q}}{\ln(q)} \tag{12.1}$$

$$= \sum_{q \in \{q\}} \frac{\sum_{i=1}^{n} \sum_{j=0}^{q-1} y_{qij} \ln\left(\hat{y}_{qij}\right)}{n \ln(q)} \tag{12.2}$$

$$= \sum_{q \in \{q\}} \sum_{i=1}^{n} \sum_{j=0}^{q-1} \frac{y_{qij} \ln\left(\hat{y}_{qij}\right)}{n \ln(q)} \tag{12.3}$$

To accelerate generalization, gradient filtering as per Lee et al. (2024) is implemented and replicated.

$$g_t = \nabla_\theta L + \lambda(\alpha e_{t-1} + (1-\alpha)g_{t-1}) \tag{13}$$

where e_t is the exponential moving average of gradients with decay rate $\alpha = 0.98$, and λ controls the influence of the slow-varying component.

The training uses full batch gradient descent with the entire dataset of p^2 samples (12 769 when $p = 113$). The model is evaluated on a held-out validation set after each epoch, tracking per-task accuracy and loss. As the setup used in $\mathcal{T}_{\text{nanda}}$, training was done on thirty percent of the total dataset, with the remaining used for validation (1000 samples) and testing (remaining). Further, as $\mathcal{T}_{\text{miiii}}$ involves the learning of 29 (when $p = 113$) tasks rather than 1, and due to each task's non-commutativity, a larger hidden dimension of 256 was added to the hyper parameter search space, as well as the potential for 8 heads ($\mathcal{T}_{\text{nanda}}$ was solved with a hidden dimension of 128, and 4 heads). The number of transformer blocks was kept at 1, as this ensures consistency with $\mathcal{T}_{\text{nanda}}$ (and as full generalization was possible, as we shall see in the Section 4).

Training was done on a NVIDIA GeForce RTX 4090 GPU, with Python3.11 and extensive use of "JAX 0.4.35" and its associated ecosystem. Neuron activations were calculated at every training step and logged for later analysis.

3.5 | Visualization

Much of the data worked with here is inherently high dimensional. For training, for example, we have n steps, two splits (train/valid) about $\frac{p}{\ln(p)}$ tasks, and two metrics (accuracy and loss). This, along with the inherent opaqueness of deep learning models, motivated the development of a custom visualization library, `esch`[3], to visualize attention weights, intermediate representations, training metrics, and more. To familiarize the reader with visualizing the inner workings of a trained model, an essential plot type for the reader to keep in mind is seen in Figure 5. As there are only 12 769 samples when $p = 113$, all samples can be fed at once to the model. Inspecting a specific activation thus yields a $1 \times 12\,796$ vector v, which can be reshaped as a 113×113 matrix, with the two axes, x_0 and x_1, varying from 0 to 112, respectively. The top-left corner then shows the given value for the sample $(0 \cdot p^0 + 0 \cdot p^1)$, and so on.

[3]https://github.com/syrkis/esch

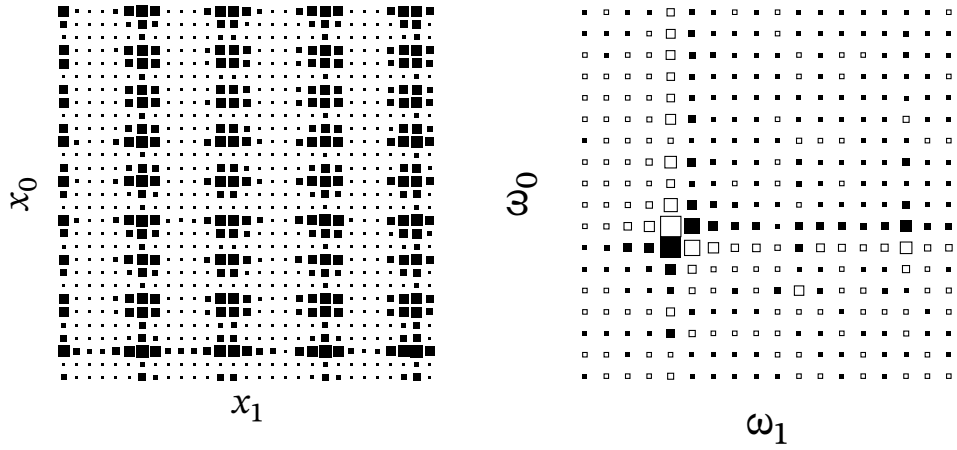


Figure 5: Plotting a neuron: (left) The activation of a particular neuron as x_0 and x_1 varies from 0 to p. (right) The same processed with a fast Fourier transform to active frequencies (ω).

Note that in `esch` plots, when appropriate, only the top leftmost 37×37 slice is shown so as not to overwhelm the reader.

3.6 | Mechanistic interpretability process

Recall that a combination of linear products is itself a linear product. Therefore, as a mechanistic interpretability rule of thumb, one should look at the outputs of the non-linear transformations. In our case, that will be the attention weights and the intermediate representations within the transformer block's feed-forward output (which follows the ReLU activation). Additionally, the embedding layers will be inspected using Fourier analysis and singular value decomposition. As mentioned in Section 2.1, our interpretability approach combines activation visualization with frequency analysis to understand the learned algorithmic patterns. Following Nanda et al. (2023), we analyze both the attention patterns and the learned representations through several lenses:

3.6.1 | Attention visualization

Using `esch`, the custom visualization library, to visualize attention weights and intermediate representations. The library allows for the visualization of attention patterns across different layers, as well as the visualization of intermediate representations at each layer. These visualizations provide insights into the learned patterns and help identify potential areas of improvement.

3.6.2 | The fast Fourier transform

As periodicity is established by Nanda et al. (2023) as a fundamental feature of the model trained on $\mathcal{T}_{\text{nanda}}$, the fast Fourier transform (FFT) algorithm is used to detect which frequencies are in play. Note that any square image can be described as a sum of 2d sine and cosine waves varying in frequency from 1 to the size of the image divided by 2 (plus a constant). This is a fundamental tool used in signal processing. The theory is briefly outlined in Appendix B for reference. This analysis helps identify the dominant frequencies in the model's computational patterns. Recall that a vector can be described as a linear combination of other periodic vectors as per the discrete Fourier transform.

The default basis of the one-hot encoded representation of the input is thus the identity matrix. This can be projected into a Fourier basis by multiplying with the discrete Fourier transform (DFT) matrix visualized in Appendix BA.

4 | Results and analysis

4.1 | Hyper-parameter optimization

The best-performing hyper-parameters for training the model on $\mathcal{T}_{\text{miiii}}$ are listed in Table 2. Notably, the model did not converge when $\lambda = 0$, confirming the utility of the gradient amplification method proposed by Lee et al. (2024) in the context of $\mathcal{T}_{\text{miiii}}$.

dropout	λ	wd	d	lr	heads
$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{3}$	256	3×10^{-4}	4

Table 2: Result of hyper-parameter search over $\mathcal{T}_{\text{miiii}}$.

4.2 | Model Performance

Figure 6 show the training and validation accuracy on $\mathcal{T}_{\text{miiii}}$ over time. The model achieves a perfect accuracy of 1 on the validation set across all 29 tasks. The cross-entropy loss in Figure 7 echoes this. In short—and to use the terminology of Power et al. (2022)—the model "grokked" on all tasks. Interestingly, tasks corresponding to modulo 2, 3, 5, and 7 generalized in succession, while the remaining 25 tasks generalized around

epoch 40 000 in no particular order. This might suggest that the model initially learned solutions for the simpler tasks and later developed a more general computational strategy that allowed it to generalize across the remaining, more complex tasks.

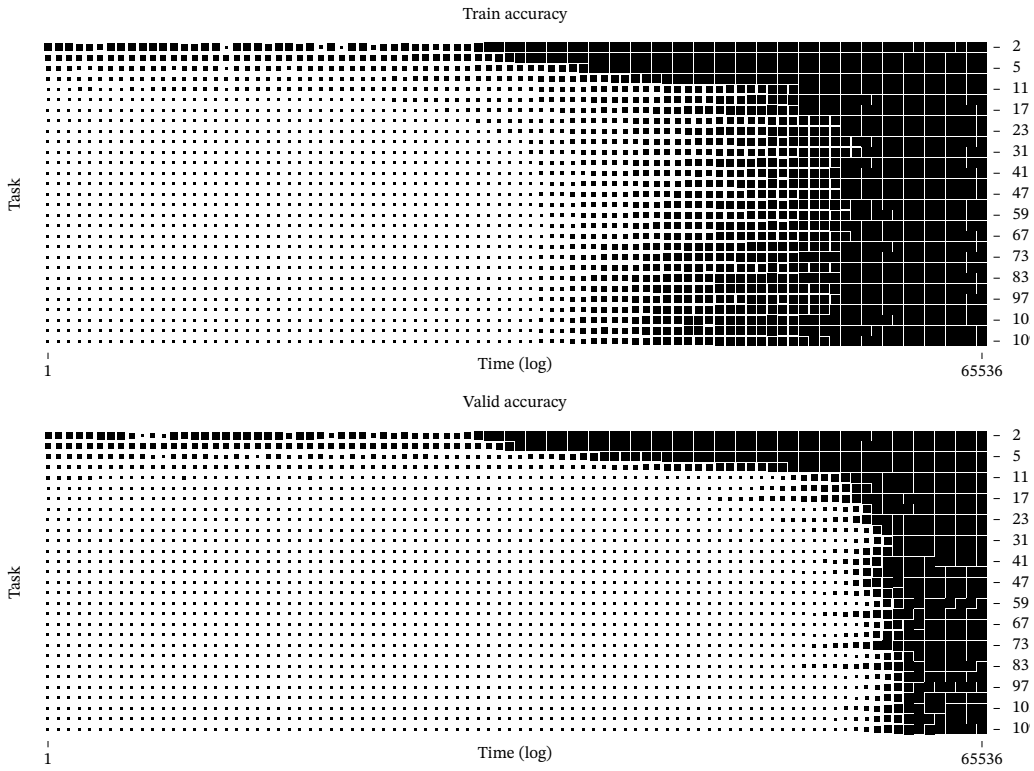


Figure 6: Accuracy training "curves": Training (top) and validation (bottom) accuracy over time (x-axis in log-scale). We see grokking occur on all tasks, first for $q \in \{2, 3, 5, 7\}$ in that order, and then the remaining 25 in no particular order.

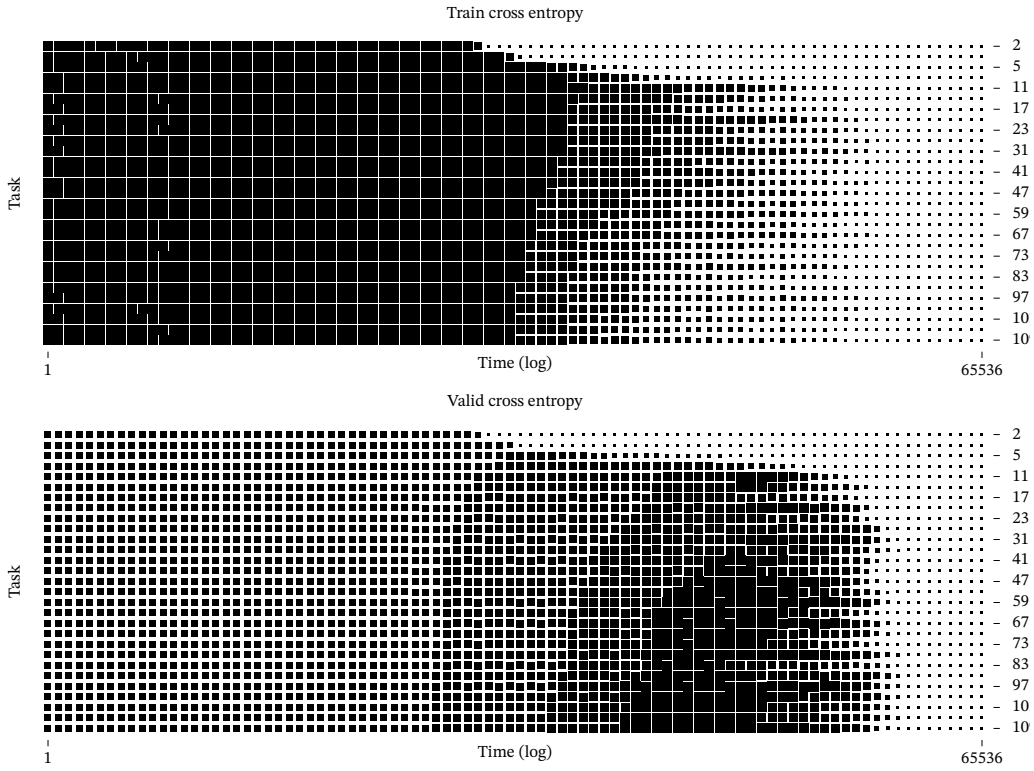


Figure 7: Cross-entropy (Eq. 6.1) loss on training (top) and validation (bottom) over time (note the log scale on the x-axis).

4.3 | Embeddings

Positional embeddings play a crucial role in transformers by encoding the position of tokens in a sequence. Figure 8 compares the positional embeddings of models trained on $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$.

For $\mathcal{T}_{\text{nanda}}$, which involves a commutative task, the positional embeddings are virtually identical, with a Pearson correlation of 0.95, reflecting that the position of input tokens does not significantly alter their contribution to the task. In contrast, for $\mathcal{T}_{\text{miiii}}$, the positional embeddings have a Pearson correlation of −0.64, indicating that the embeddings for the two positions are different. This difference is expected due to the non-commutative nature of the task, where the order of x_0 and x_1 matters ($x_0 \cdot p^0 \neq x_0 \cdot p^1$). This confirms that the model appropriately encodes position information for solving the tasks.

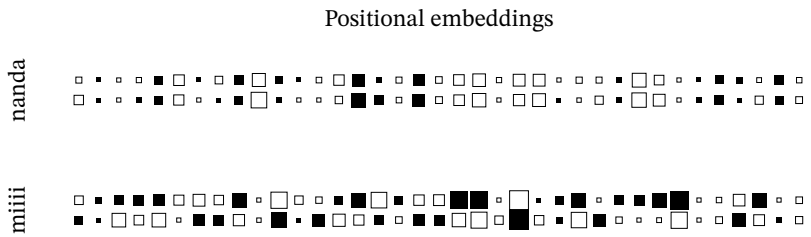


Figure 8: Positional embeddings for (x_0, x_1) for models trained on $\mathcal{T}_{\text{nanda}}$ (top) and $\mathcal{T}_{\text{miiii}}$ (bottom). Pearson's correlation is 0.95 and −0.64 respectively. This reflects the commutativity of $\mathcal{T}_{\text{nanda}}$ and the lack thereof for $\mathcal{T}_{\text{miiii}}$. Hollow cells indicate negative numbers.

Recall that a matrix $\mathbf{M}$ of size $m \times n$ can be decomposed to its singular values $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$ (with the transpose being the complex conjugate when $\mathbf{M}$ is complex), where $\mathbf{U}$ is $m \times m$, $\mathbf{\Sigma}$ an $m \times n$ rectangular diagonal matrix (whose diagonal is represented as a flat vector throughout this paper), and $\mathbf{V}^{\mathbf{T}}$ a $n \times n$ matrix. Intuitively, this can be thought of as rotating in the input space, then scaling, and then rotating in the output space.

Figure 9 displays the singular values of the token embeddings learned for $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$. The singular values for $\mathcal{T}_{\text{miiii}}$ are more diffuse, indicating that a larger number of components are needed to capture the variance in the embeddings compared to $\mathcal{T}_{\text{nanda}}$. This suggests that the token embeddings for $\mathcal{T}_{\text{miiii}}$ encode more complex information, reflecting the increased complexity of the multi-task learning scenario.

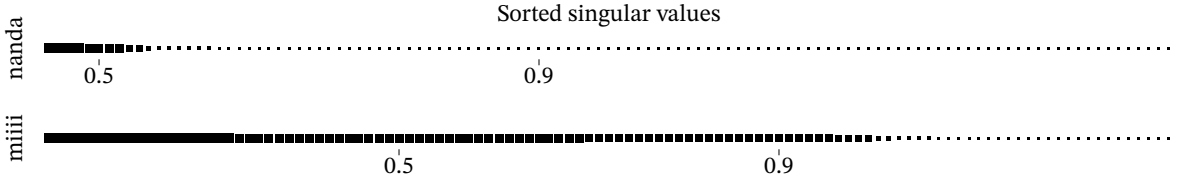


Figure 9: First 83 of 113 singular values (truncated for clarity) of U for $\mathcal{T}_{\text{nanda}}$ (top) and $\mathcal{T}_{\text{miiii}}$ (bottom). The ticks indicate the points where 50% and 90% of the variance is accounted for. We thus see that for $\mathcal{T}_{\text{miiii}}$, the embedding space is much more crammed.

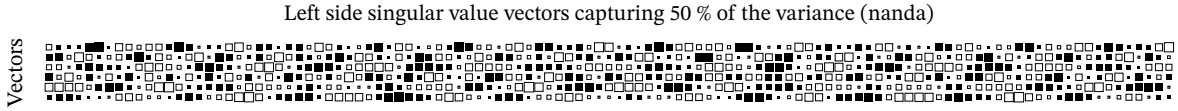


Figure 10: $\mathcal{T}_{\text{nanda}}$'s most significant (cutoff at 0.5 as per Figure 9) singular vectors of U from the singular value decomposition. Note this looks periodic!

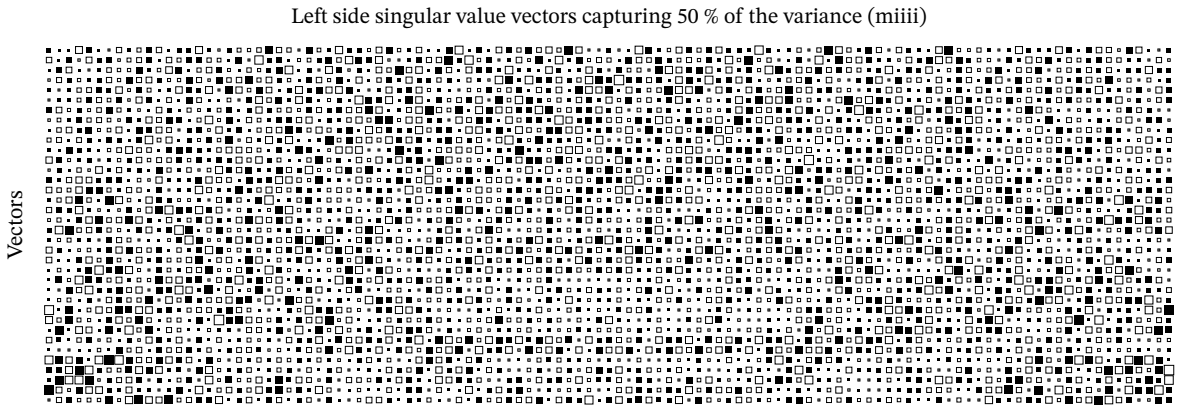


Figure 11: $\mathcal{T}_{\text{miiii}}$'s most significant vectors of U. Note that, like in Figure 10, we still observe periodicity, but there are more frequencies in play, as further explored in Figure 13.

Figure 10 and Figure 11 present the most significant singular vectors of U for $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$, respectively. Visual inspection shows periodicity in the top vectors for both models, but the $\mathcal{T}_{\text{miiii}}$ model requires more vectors to capture the same amount of variance, consistent with the diffuse singular values observed in Figure 9.

To further understand the structure of the token embeddings, we applied the Fast Fourier Transform (FFT). Only a few frequencies are active for $\mathcal{T}_{\text{nanda}}$ as seen in Figure 12, consistent with the model implementing a cosine-sine lookup table as described in Nanda et al. (2023).

For the $\mathcal{T}_{\text{miiii}}$ model, we observe a broader spectrum of active frequencies (Figure 13). This is expected due to the model having to represent periodicity corresponding to 29 primes.

Comparing with $\mathcal{T}_{\text{basis}}$ in figure Figure 14, the periodicity is understood to be a structure inherent to the data picked up by the model.

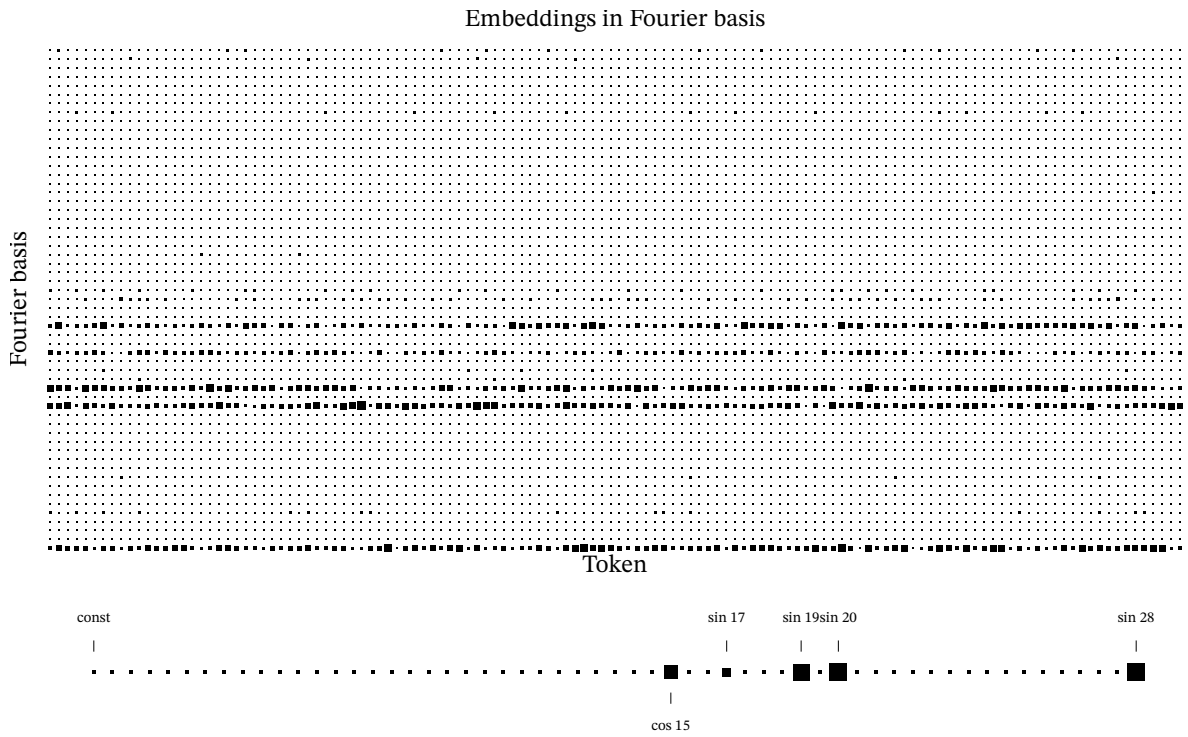


Figure 12: $\mathcal{T}_{\text{nanda}}$ tokens in Fourier basis: Note how all tokens are essentially linear combinations of the five most dominant Fourier basis vectors. The sparsity echoes the findings in Figure 9 that very few directions in the embedding space are used.

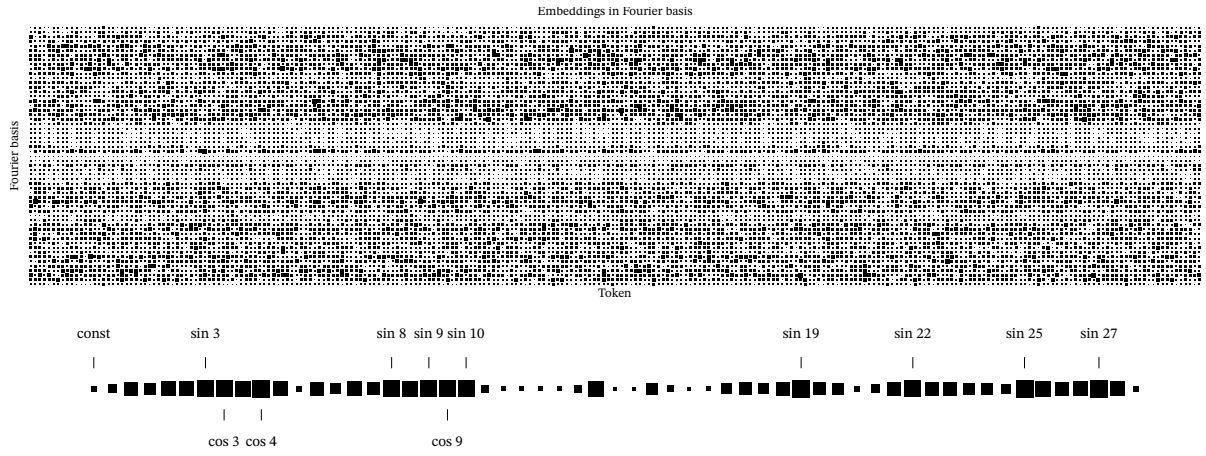


Figure 13: The periodicity in the $\mathcal{T}_{\text{miiii}}$ embeddings involves a much larger fraction of the Fourier basis, echoing the multiple tasks and their innate difference in frequency (recall that all tasks are performed on unique primes q).

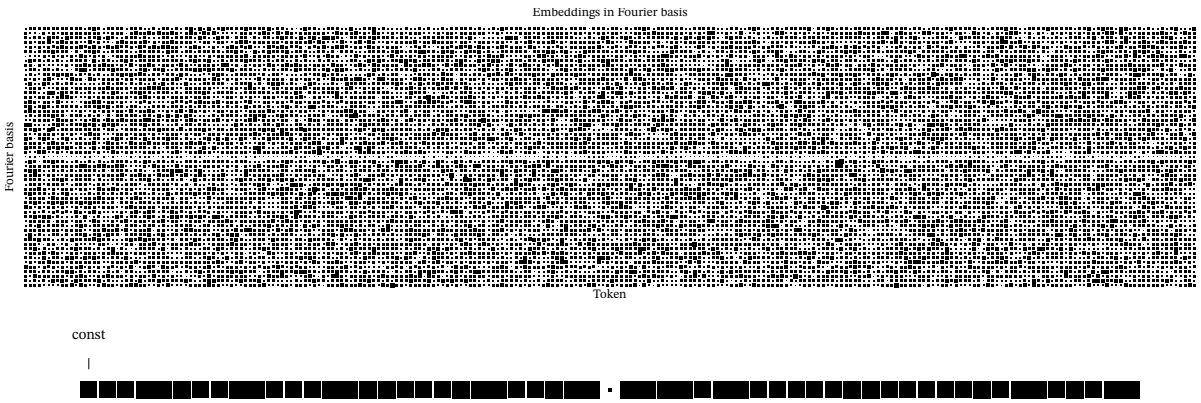


Figure 14: Embeddings for $\mathcal{T}_{\text{basis}}$ in Fourier basis have no periodicity. The periodicity is indeed an artifact of the modulo operator.

4.4 | Analysis of Neuron Activations and Frequencies

To understand the internal mechanisms developed by the model, we analyzed the neuron activations after the output weight matrix W_{out} for the model trained on $\mathcal{T}_{\text{miiii}}$. Figure 15 shows that these activations exhibit periodic patterns with respect to (x_0, x_1). This periodicity aligns with the modular arithmetic nature of the tasks, mirroring Nanda et al. (2023) ($\mathcal{T}_{\text{nanda}}$).

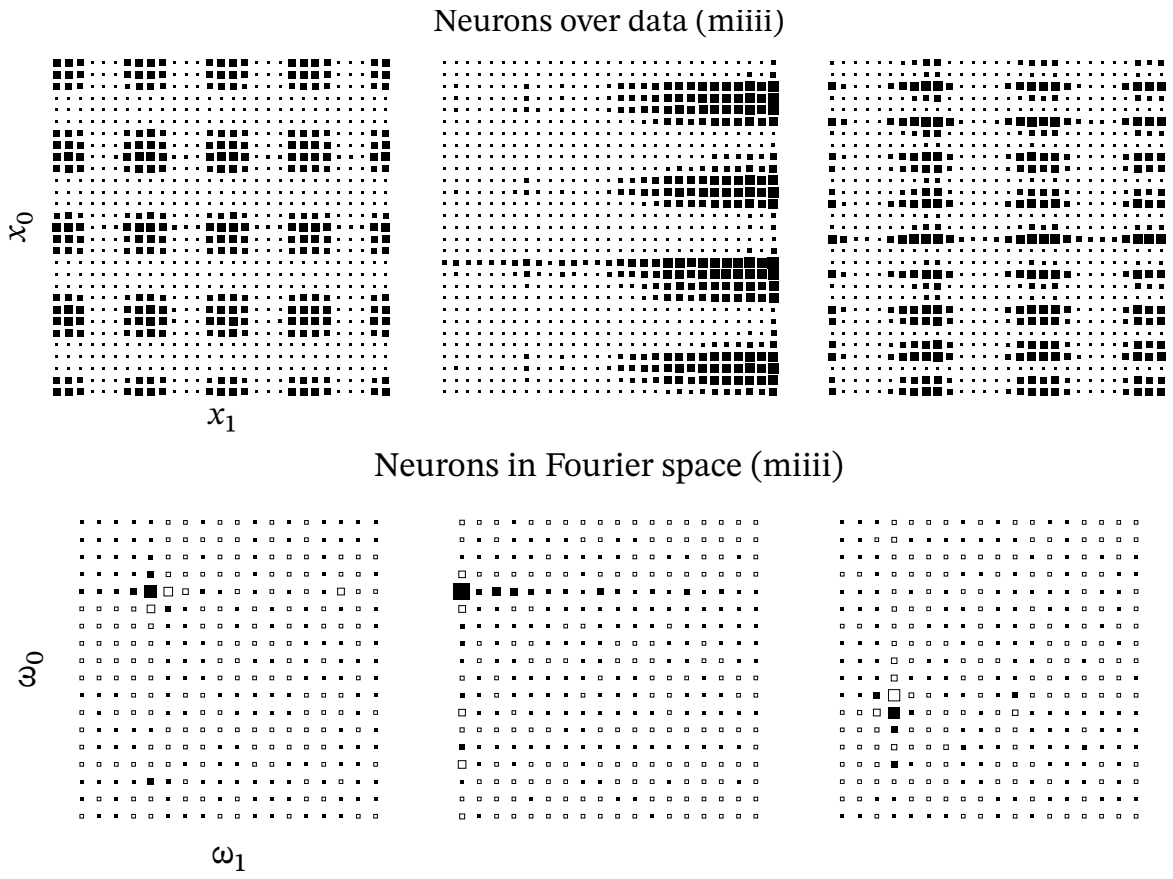


Figure 15: We plot the activation of the first three neurons of the activations immediately following ReLU in Eq. 9 as x_0 and x_1 vary (top). Note we only show the top 37×37 corner of the full 133×113 sample matrix. Here too we see periodicity, confirmed by a Fourier transform (bottom). Neurons are reactive to highly particular frequencies in their input domains.

For comparison, Figure 16 shows the neuron activations for a model trained on $\mathcal{T}_{\text{basis}}$. These activations do *not* exhibit periodicity, confirming that the observed periodic patterns in the models trained for $\mathcal{T}_{\text{miiii}}$ and $\mathcal{T}_{\text{nanda}}$, too, are indeed a result of the modulo operations inherent in the tasks.

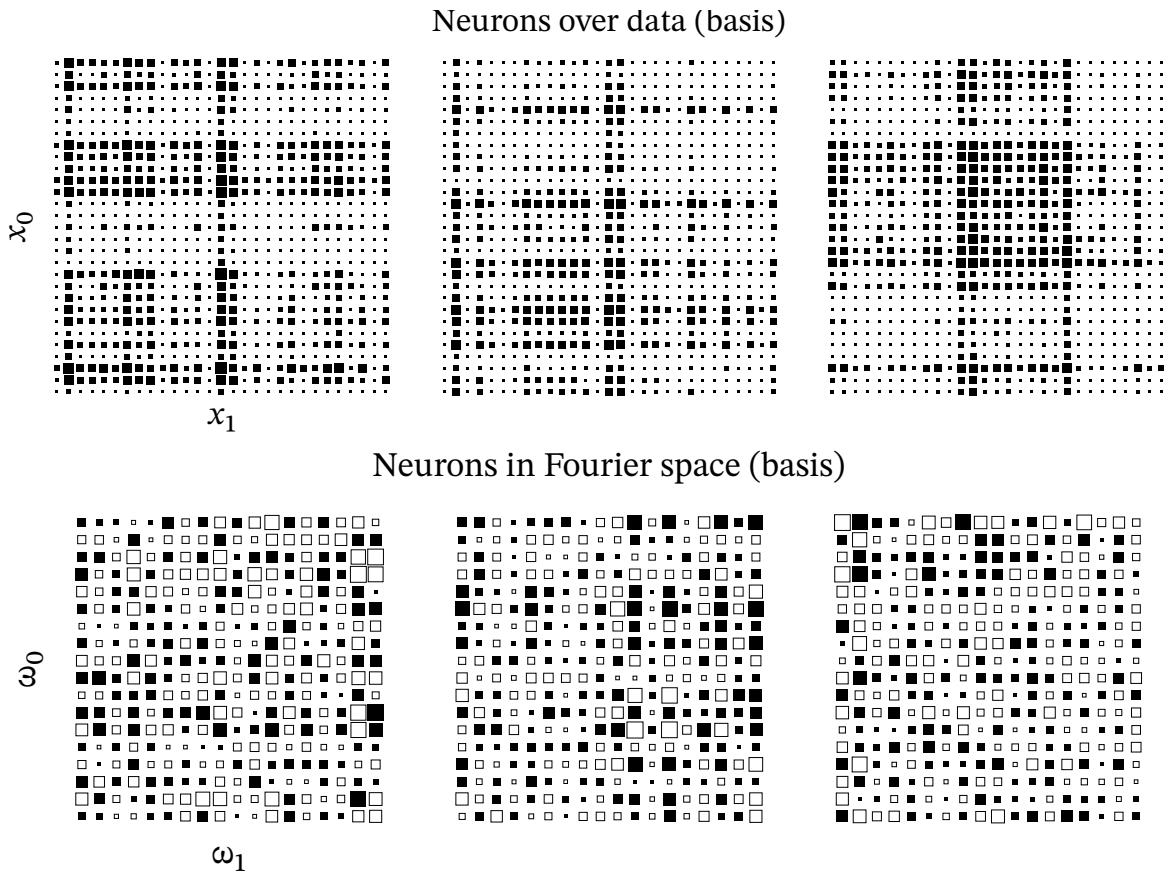


Figure 16: Neuron activations for model trained on $\mathcal{T}_{\text{basis}}$. As in Figure 14, no periodicity is observed for the baseline.

The analysis of active frequencies *through training* using the Fast Fourier Transform (FFT) is illustrated in Figure 17, with the core findings showing a spike in frequency activation around epoch 16 384 visible in Table 3. The top plot shows the different frequencies of the transformer block's feed-forward neurons evolving as the model learns. The bottom plot displays the variance of frequency activations and the number of frequencies exceeding a significance threshold $\omega > \mu + 2\sigma$ (i.e., which spots like the ones of the bottom row of Figure 15 are active). Initially, a handful of frequencies become dominant as the model generalizes on the first four tasks. As training progresses and the model begins to generalize on the remaining tasks, more frequencies become significant, suggesting that the model is developing more complex internal representations to handle the additional tasks.

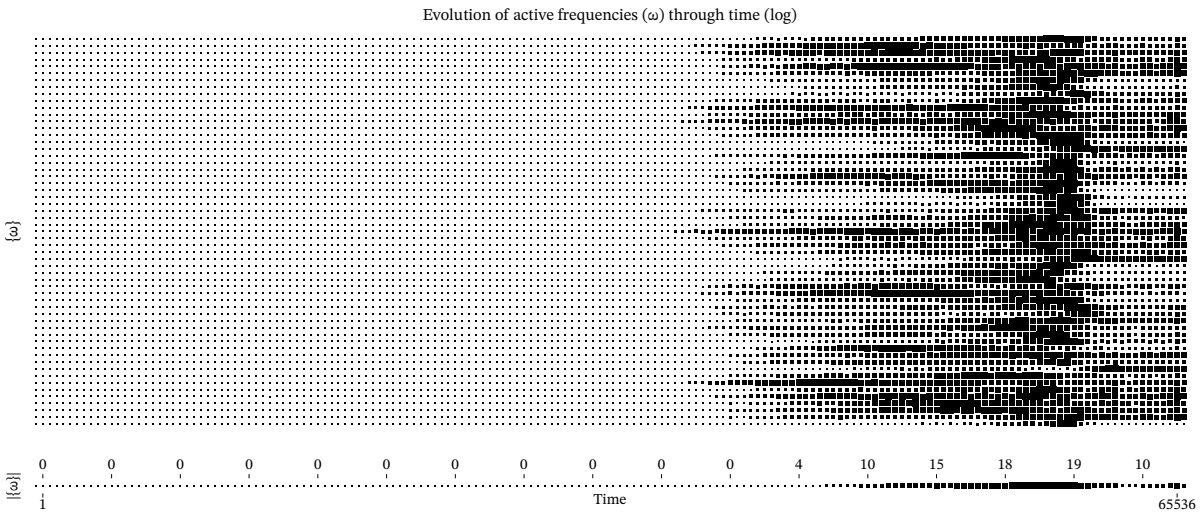


Figure 17: Top: Frequency dominance is averaged over neurons through training (average activation of frequencies as shown in Figure 15). We see four distinct phases: *1)* no significant frequencies, *2)* frequencies gradually emerging, *3)* a wall of frequencies, and *4)* frequency count similar to phase 2. Bottom: total number of active frequencies at the corresponding time step. A frequency ω is active when $\omega > \mu + 2\sigma$ (a frequently used signal processing default).

epoch	256	1024	4096	16384	65536
$\lvert\omega\rvert$	0	0	10	18	10

Table 3: Number of active frequencies on $\mathcal{T}_{\text{miiii}}$ over epochs.

Figure 18 shows the L2 norms of gradients through time for the different weight matrices of the model trained on $\mathcal{T}_{\text{miiii}}$. The gradient norms provide insights into how different parts of the model are being updated during training. Like with Nanda et al. (2023), the attention layer converges quickly, echoing their finding that it does not contribute much to solving their modular arithmetic task.

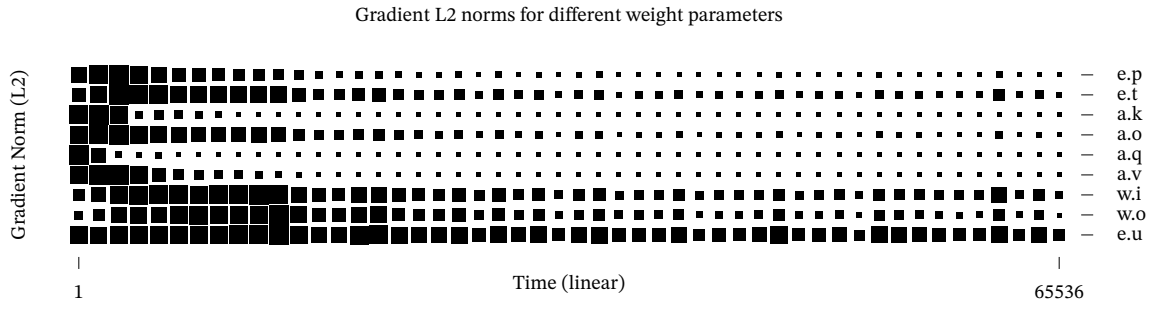


Figure 18: L2 norms of gradients over time for the different weight matrices of the model trained on $\mathcal{T}_{\text{miiii}}$. Row order corresponds to how deep in the model the weight is used. This shows when during training what parts of the model are updated. `e.*` are embeddings Eq. 7, `a.*` attention layer weights Eq. 8, and `w.*` weights of the feed-forward module Eq. 9 (`e.u` being the final un-embedding layer.)

These results demonstrate that more frequencies are involved when training on $\mathcal{T}_{\text{miiii}}$ compared to $\mathcal{T}_{\text{nanda}}$. The increased frequency components reflect the need for the model to encode multiple periodic patterns corresponding to the various modular arithmetic tasks.

Combining the analysis of embeddings and the transformer block neurons, we see that:
1. A lot more frequencies are in play for $\mathcal{T}_{\text{miiii}}$ than in $\mathcal{T}_{\text{nanda}}$.
2. Neurons remain highly reactive to a very small set of frequencies.
3. The periodicity is an artifact of the modulo group by analysis of $\mathcal{T}_{\text{basis}}$

4.5 | Attention Patterns

Figure 19 shows that, in contrast to the model trained on $\mathcal{T}_{\text{nanda}}$, where attention heads may focus jointly on both input tokens in a periodic fashion, the attention heads for the $\mathcal{T}_{\text{miiii}}$ model focus exclusively on one digit or the other. This behavior could be due to the non-commutative nature of the task, where the position of each digit significantly affects the outcome. Nanda et al. (2023) concludes that the attention mechanism does not contribute significantly[4] to the solving of $\mathcal{T}_{\text{nanda}}$, and will thus also not be explored further here.

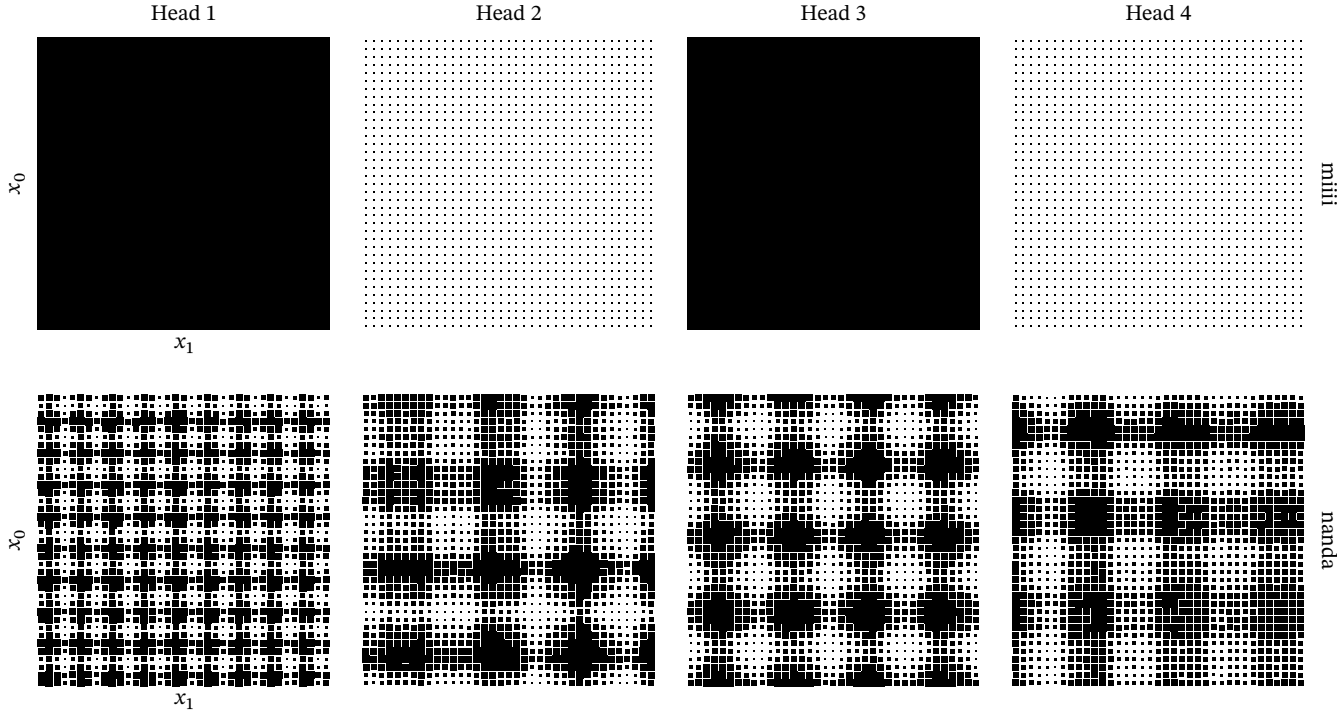


Figure 19: Attention from $\hat{y}$ to x_0 for the four attention heads in the $\mathcal{T}_{\text{miiii}}$ model. The attention heads tend to focus on one digit, reflecting the non-commutative nature of the task.

[4]Neel Nanda has also stated (in a YouTube video) that a multi layer perceptron rather than a transformer-block would probably have been more appropriate for his setup. The transformer is used here due to the non-commutativity of $\mathcal{T}_{\text{miiii}}$, and to stay close to Nanda's work.

Overall, our results demonstrate that the model effectively learns to solve multiple modular arithmetic tasks by developing internal representations that capture the periodic nature of these tasks. The analysis of embeddings and neuron activations provides insights into how the model generalizes from simpler to more complex tasks, possibly through the reuse and integration of learned circuits. Interestingly, as will be discussed in Section 5, there are four significant shifts in the number of frequencies active in the neurons.

Lastly, as can be seen in Appendix C, when dropout was disabled, the model's performance diverged on the validation set (overfitting), even with other kinds of regularization (multi-task, l2, etc.). Appendix A shows the accuracy through training for $\mathcal{T}_{\text{masked}}$. The masking of $q \in \{2, 3, 5, 7\}$, does *not* delay grokking on the remaining tasks notably, the spiking after generalization to easy tasks ($q \in \{11, 13, 17, 19\}$) remains.

5 | Discussion

Recall that, when viewed individually, the sub-tasks of $\mathcal{T}_{\text{miiii}}$ differ from $\mathcal{T}_{\text{nanda}}$ only in commutativity (making the task harder) and in prime since $q < p$ (making the task easier for smaller q's like $\{2, 3, 5, 7\}$—though less so as q approaches p). Figure 8 indicates that the model learns to account for the commutativity by using positional embeddings.

During training, the changes in the number of active neuron frequencies ω in the feed-forward layer (see Figure 17) echo the loss and accuracy progression seen in Figure 6 and Figure 7. Further, as the model groks on the primes $q \in \{2, 3, 5, 7\}$, a handful of frequencies become dominant, similar to the original model trained on $\mathcal{T}_{\text{nanda}}$.

We thus have that: *1)* the model learns to correct for commutativity, and *2)* the model is marred by periodicity as per both the token embedding (Section 4.3) and feed-forward layer analysis (Section 4.4). Combining these facts, we can assume the learned mechanism to be extremely similar (and perhaps identical when ignoring commutativity) to the one outlined by Nanda et al. (2023) in Eq. 3, Eq. 4, and Eq. 5.

As the remaining 25 tasks are learned, we see a temporary spike in the number of active frequencies, disappearing again as the model generalizes fully (reaches perfect accuracy). The observation that the number of active frequencies before and after the remaining 25 tasks are learned are the same indicates a reuse of circuitry. However, the fact of the spike suggests that additional circuits form during the generalization process. Viewed in the context of Lee et al. (2024)'s method for boosting slow-varying generalizing components a question emerges: are there circuits that only facilitate the development of generalization, but are not present in the generalized mechanisms?

Real life gives plenty of examples of phenomena that make itself obsolete (e.g., a medicine that fully eradicates an illness and is thus no longer needed). Viewed this way, the spike suggests we might divide the set of circuits in two: *1)*; those useful for the mechanism, and *2)*, those useful for *learning* the mechanism.

6 | Future work

A logical next step would be to explore the validity of the notion that some circuits help learning and others help solve the problem. This might yield insight on how to improve Lee et al. (2024)'s grokking speedup heuristic. Aspects of the circuit discovery workflow could be automated with the methods outlined by Conmy et al. (2023).

Additionally, making variations on $\mathcal{T}_{\text{miiii}}$ is also likely to be a good avenue for discovery: Divisibility rather than remainder could be predicted; Experiments training for more epochs could be conducted with larger values of p; A more in depth mapping of the shared circuitry could be done, for example attempting to see what types of ablations break which tasks—for example, how can performance be degraded on one grokked task, without affecting the others?.

The code associated with this paper is available as a PyPI package (`pip install miiii`) to facilitate the exploration of these questions (as well as replication of the findings at hand).

7 | Conclusion

This paper explores the impact of multi-task learning on mechanistic interpretability by training a transformer model on a non-commutative, multi-task modular arithmetic problem $\mathcal{T}_{\text{miiii}}$. The model successfully generalizes across all tasks, learning complex internal representations that capture the unique periodic nature of modular arithmetic across multiple primes. Analysis reveals that while the model reuses and integrates circuits for simpler tasks, additional circuits may form during training to facilitate generalization to more complex tasks.

These findings highlight that multi-task learning influences the emergence and complexity of internal mechanisms, posing challenges for mechanistic interpretability but also offering insights into how models internalize algorithms. Understanding these dynamics is important for advancing interpretability and reliability in deep learning systems. Future work includes exploring the distinction between circuits that aid in learning versus those that contribute to the final mechanism/solution and investigating

how variations in task design impact the development of internal representations. Advancing the understanding of how deep learning models handle multiple tasks contributes to the broader goal of making these models more interpretable and reliable.

References

[1] J. Jumper *et al.*, "Highly Accurate Protein Structure Prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

[2] E. Dinan *et al.*, "Human-Level Play in the Game of \emph{Diplomacy} by Combining Language Models with Strategic Reasoning," *Science*, vol. 378, no. 6624, pp. 1067–1074, Dec. 2022, doi: 10.1126/science.ade9097.

[3] A. Radford and K. Narasimhan, "Improving Language Understanding by Generative Pre-Training," 2018.

[4] C. E. Shannon, "Programming a Computer for Playing Chess," *Computer Chess Compendium*, pp. 2–13, 1950, doi: 10.1007/978-1-4757-1968-0_1.

[5] G. Tesauro, "TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play," 1993.

[6] D. Silver *et al.*, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[7] C. B. Browne *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, doi: 10.1109/TCIAIG.2012.2186810.

[8] A. Ruoss *et al.*, "Grandmaster-Level Chess Without Search," no. arXiv:2402.04494. Feb. 2024. doi: 10.48550/arXiv.2402.04494.

[9] Z. C. Lipton, "The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability Is Both Important and Slippery.," *Queue*, vol. 16, no. 3, pp. 31–57, Jun. 2018, doi: 10.1145/3236386.3241340.

[10] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," no. arXiv:1607.06450. arXiv, Jul. 2016. doi: 10.48550/arXiv.1607.06450.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[12] A. Krogh and J. Hertz, "A Simple Weight Decay Can Improve Generalization," in *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, 1991.

[13] J. Baxter, "A Model of Inductive Bias Learning," no. arXiv:1106.0245. arXiv, Jun. 2011. doi: 10.48550/arXiv.1106.0245.

[14] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt, "Progress Measures for Grokking via Mechanistic Interpretability," no. arXiv:2301.05217. arXiv, Oct. 2023.

[15] A. Vaswani *et al.*, "Attention Is All You Need," no. arXiv:1706.03762. arXiv, Dec. 2017. doi: 10.48550/arXiv.1706.03762.

[16] J. Lee, B. G. Kang, K. Kim, and K. M. Lee, "Grokfast: Accelerated Grokking by Amplifying Slow Gradients," no. arXiv:2405.20233. Jun. 2024.

[17] L. Egri and T. R. Shultz, "A Compositional Neural-network Solution to Prime-number Testing," 2006.

[18] S. Lee and S. Kim, "Exploring Prime Number Classification: Achieving High Recall Rate and Rapid Convergence with Sparse Encoding," no. arXiv:2402.03363. arXiv, Feb. 2024.

[19] D. Wu, J. Yang, M. U. Ahsan, and K. Wang, "Classification of Integers Based on Residue Classes via Modern Deep Learning Algorithms." Apr. 2023. doi: 10.1016/j.patter.2023.100860.

[20] A. Conmy, A. N. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso, "Towards Automated Circuit Discovery for Mechanistic Interpretability," no. arXiv:2304.14997. arXiv, Oct. 2023. doi: 10.48550/arXiv.2304.14997.

[21] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velǐcković, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," no. arXiv:2104.13478. arXiv, May 2021. doi: 10.48550/arXiv.2104.13478.

[22] S. Yu, L. Sanchez Giraldo, and J. Principe, "Information-Theoretic Methods in Deep Neural Networks: Recent Advances and Emerging Opportunities," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, Montreal, Canada: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 4669–4678. doi: 10.24963/ijcai.2021/633.

[23] B. Gavranović, P. Lessard, A. Dudzik, T. von Glehn, J. G. M. Araújo, and P. Velǐcković, "Categorical Deep Learning: An Algebraic Theory of Architectures," no. arXiv:2402.15332. arXiv, Feb. 2024. doi: 10.48550/arXiv.2402.15332.

[24] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets," no. arXiv:2201.02177. arXiv, Jan. 2022. doi: 10.48550/arXiv.2201.02177.

[25] A. I. Humayun, R. Balestriero, and R. Baraniuk, "Deep Networks Always Grok and Here Is Why," no. arXiv:2402.15555. Jun. 2024. doi: 10.48550/arXiv.2402.15555.

[26] B. Wang, X. Yue, Y. Su, and H. Sun, "Grokked Transformers Are Implicit Reasoners: A Mechanistic Journey to the Edge of Generalization," no. arXiv:2405.15071. May 2024. doi: 10.48550/arXiv.2405.15071.

[27] H. J. Jeon and B. Van Roy, "An Information-Theoretic Framework for Deep Learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3279–3291, Dec. 2022.

[28] A. Maurer, "Bounds for Linear Multi-Task Learning."

[29] C. E. Shannon, "A Mathematical Theory of Communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, Jan. 2001, doi: 10.1145/584091.584093.

[30] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Hoboken, N.J: Wiley-Interscience, 2006.

[31] B. He and T. Hofmann, "Simplifying Transformer Blocks," no. arXiv:2311.01906. arXiv, Nov. 2023. doi: 10.48550/arXiv.2311.01906.

[32] M. Hosseini and P. Hosseini, "You Need to Pay Better Attention," no. arXiv:2403.01643. Mar. 2024. doi: 10.48550/arXiv.2403.01643.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," no. arXiv:1502.01852. arXiv, Feb. 2015. doi: 10.48550/arXiv.1502.01852.

[34] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

[35] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," no. arXiv:1711.05101. arXiv, Jan. 2019. doi: 10.48550/arXiv.1711.05101.

Appendix

A First four tasks masked

The phenomenon of learning 4 tasks, then spiking, and then learning all tasks is also present when masking away $q \in \{2, 3, 5, 7\}$, though more subtly.

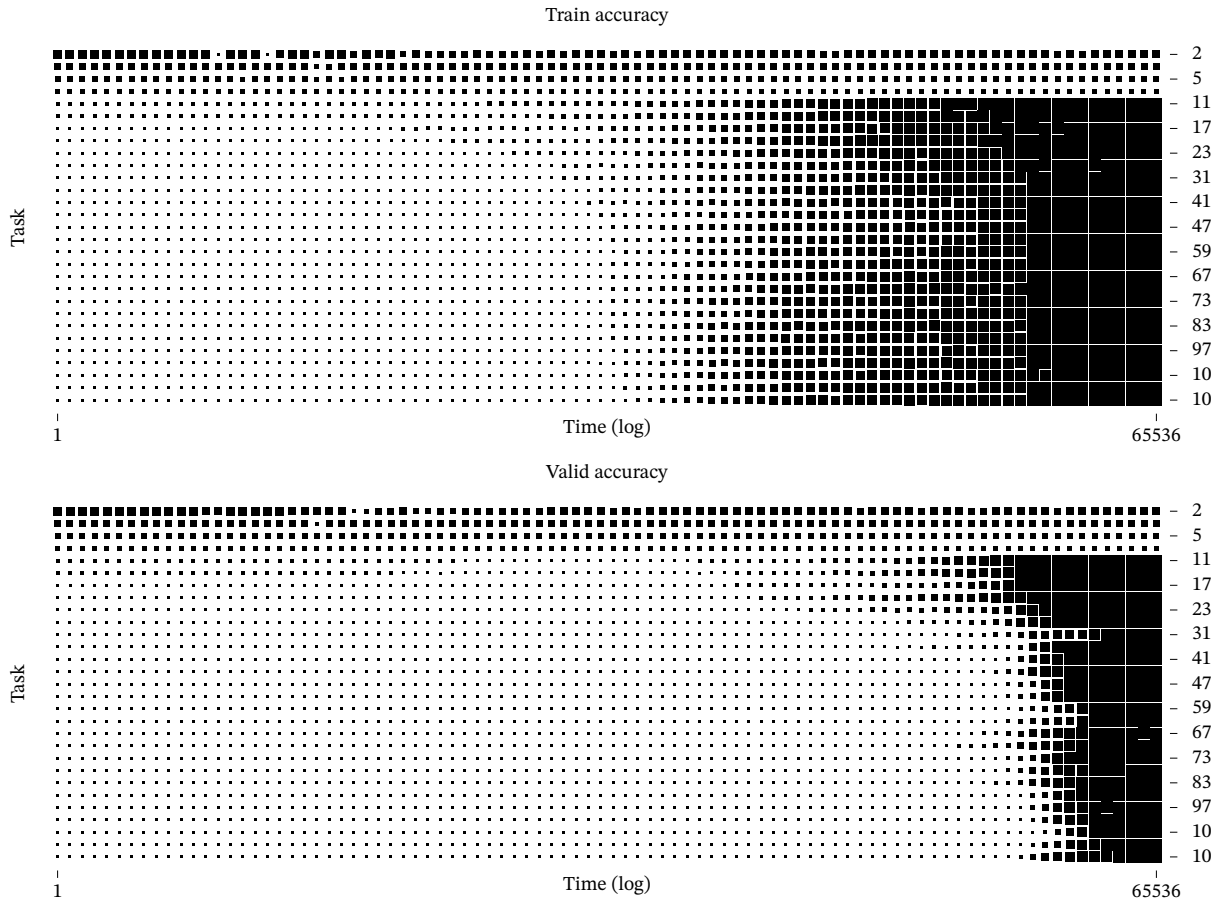


Figure 20: Accuracy when ablating $q \in \{2, 3, 5, 7\}$. Note the model now seems to learn tasks $q \in \{11, 13, 17, 19\}$ first.

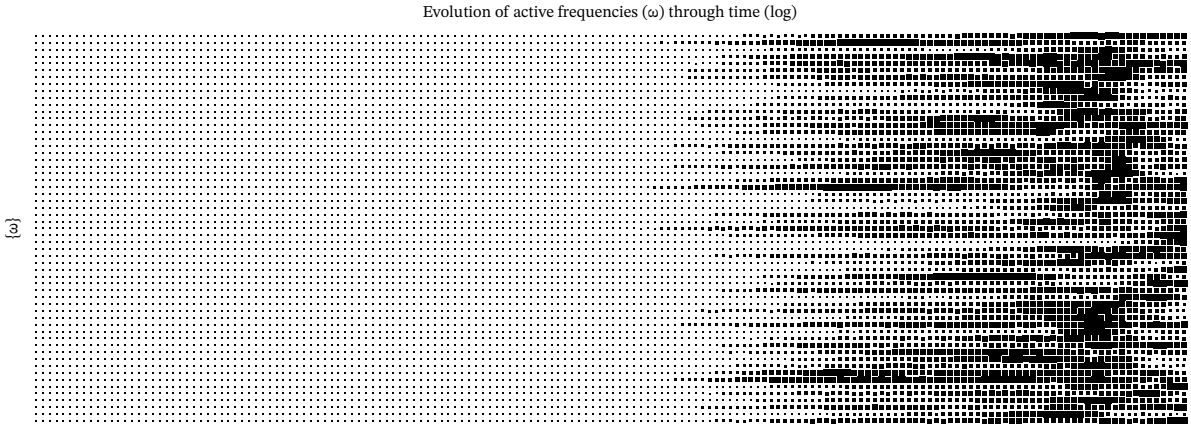


Figure 21: Active frequencies ω during training. We see that the spiking is more subtle and that the model now learns tasks $q \in \{11, 13, 17, 19\}$, before then generalizing to all tasks

B Fast Fourier Transform

The inner product between two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ of length n can be written as per Eq. 14.

$$\sum_{i}^{n} \boldsymbol{v}[i]\boldsymbol{u}[i] \tag{14}$$

We can extend the meaning of inner products to functions f and g over the interval $[a; b]$ with Eq. 15.

$$\int_{a}^{b} f(x)g(x)dx \tag{15}$$

A function $f(x)$, can be written as a sum of cosine and sine terms plus a constant as per:

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty}(A_k \cos(kx) + B_k \sin(kx)) \tag{16}$$

Where A_k and B_k are the normalized inner products $\langle f(x), \cos(kx)\rangle$ and $\langle f(x), \sin(kx)\rangle$ respectively[5]. These are explicitly written out in Eq. 17.

$$A_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dk, \quad B_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dk \tag{17}$$

This can be similarly extended for that grid, which is the basis for the two-dimensional FFT.

[5]Note the pointy brackets denote inner product

BA Discrete Fourier transform (DFT) matrix

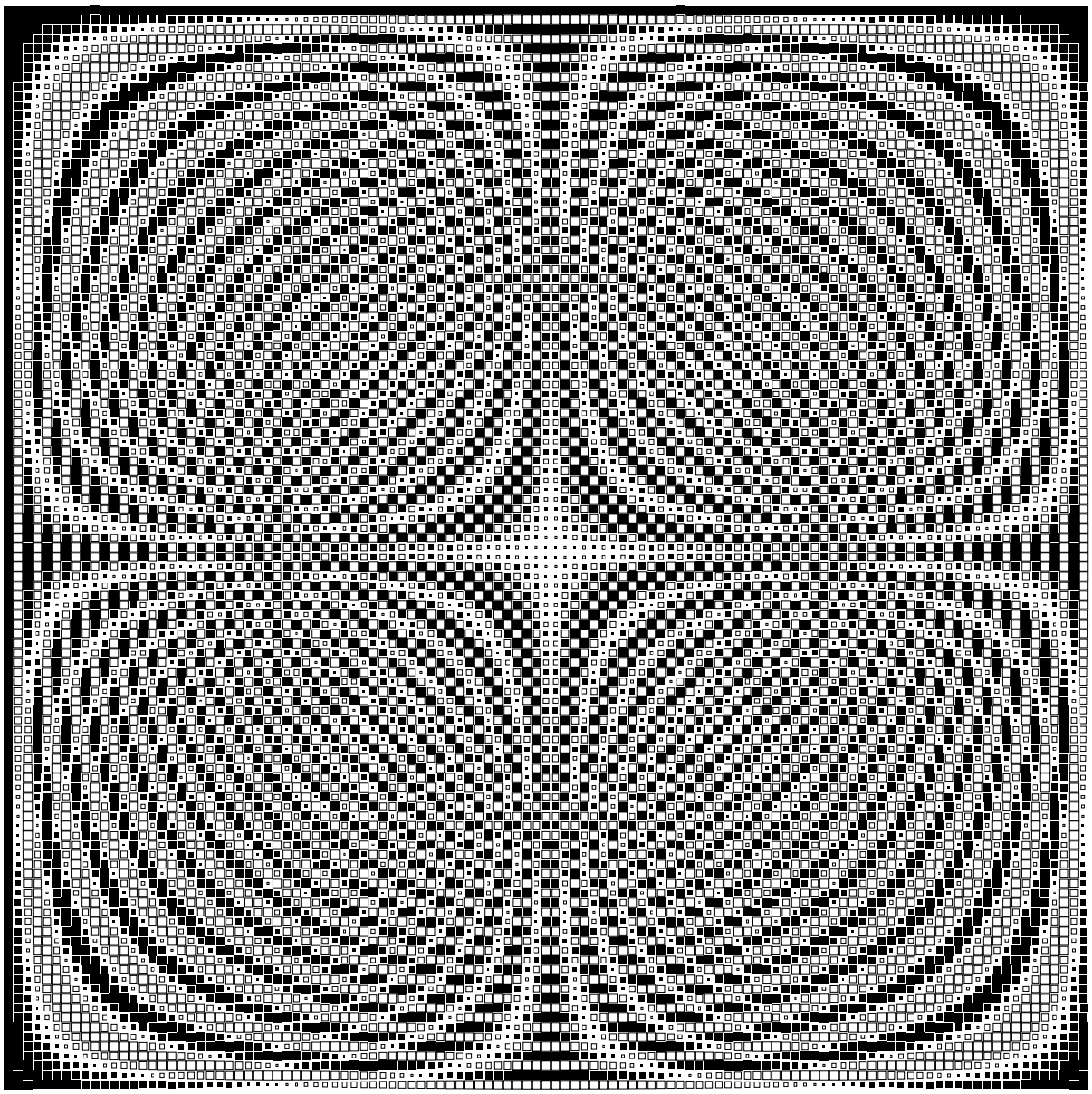


Figure 22: DFT matrix

C Dropout disabled

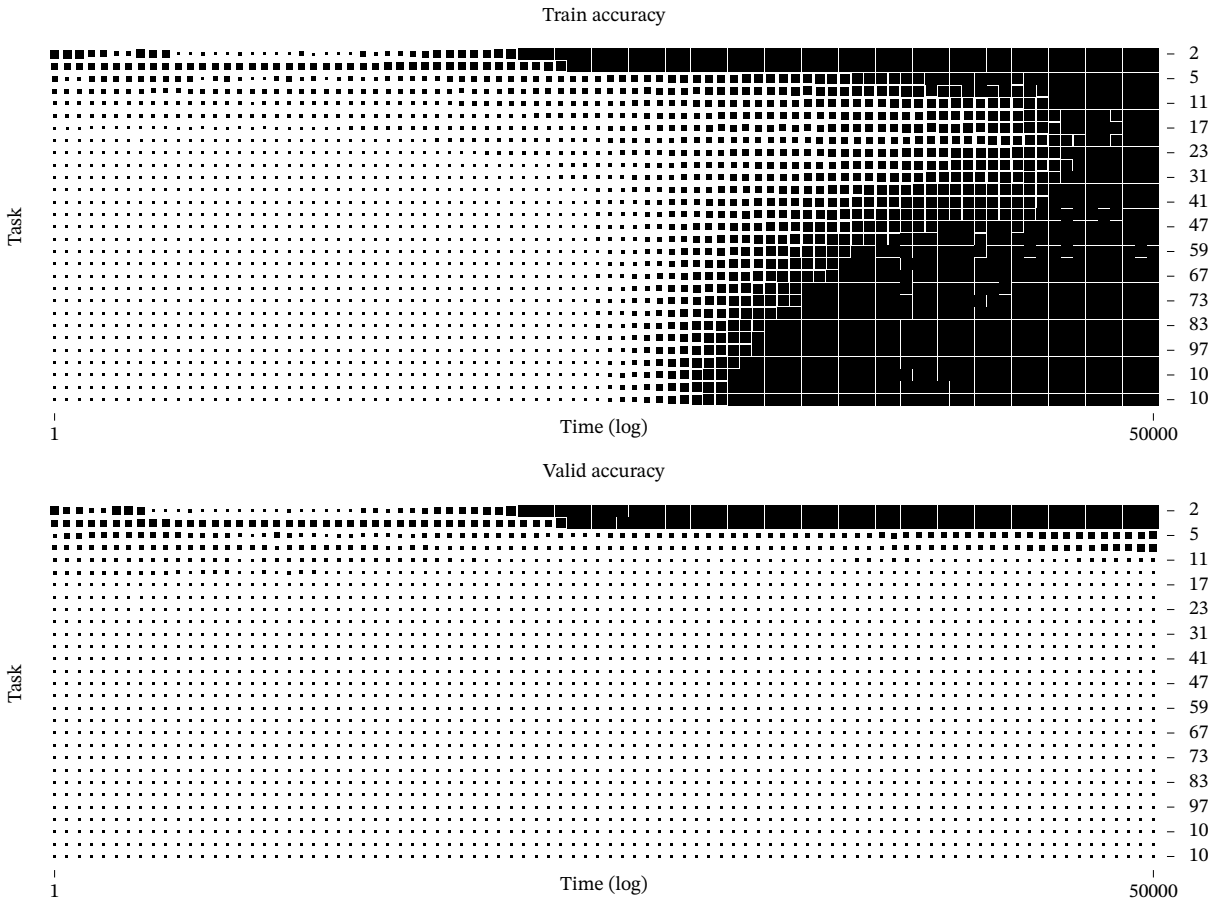


Figure 23: Disabling dropout: Without dropout, the model over-fits to all but the very simplest ($q \in \{2, 3\}$) tasks.

D Sub-symbolic implementation of $f(x, y)$

Compute $f(x)$ for $\{(a, b) \in \mathbb{N}^2 : 0 \leq a, b < 113)$, by adding the two rows of $W_{E_{\text{pos}}}$ in Figure 24 to a one-hot encoded a and b, and then multiplying by $W_{E_{\text{tok}}}$. Then multiply by W_k, W_q and W_v as per the operation described in Eq. 8, and then add to the output of the embedding operations. Send that through the feed-forward network with the weights in Figure 26. The reader is asked to confirm visually that the weight in the figures indeed computes $f(x, y) = \cos(ax) + \sin(bx)$ when applied in the order described above.

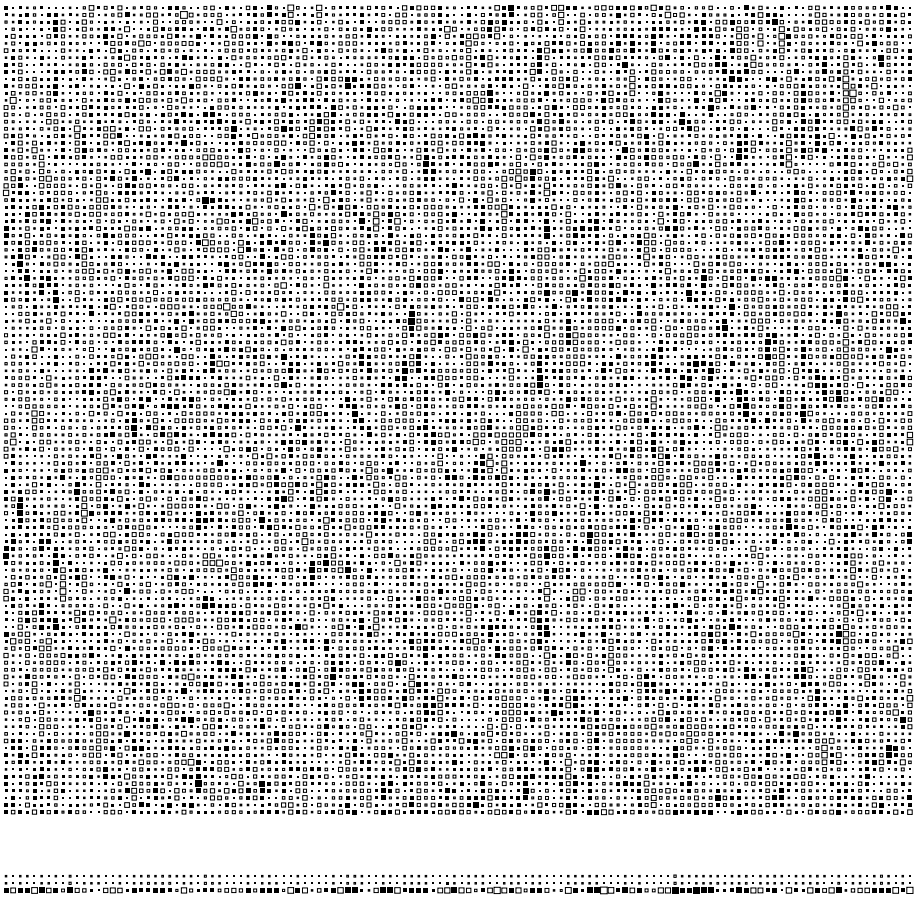


Figure 24: $W_{E_{\text{tok}}}$ and $W_{E_{\text{pos}}}$

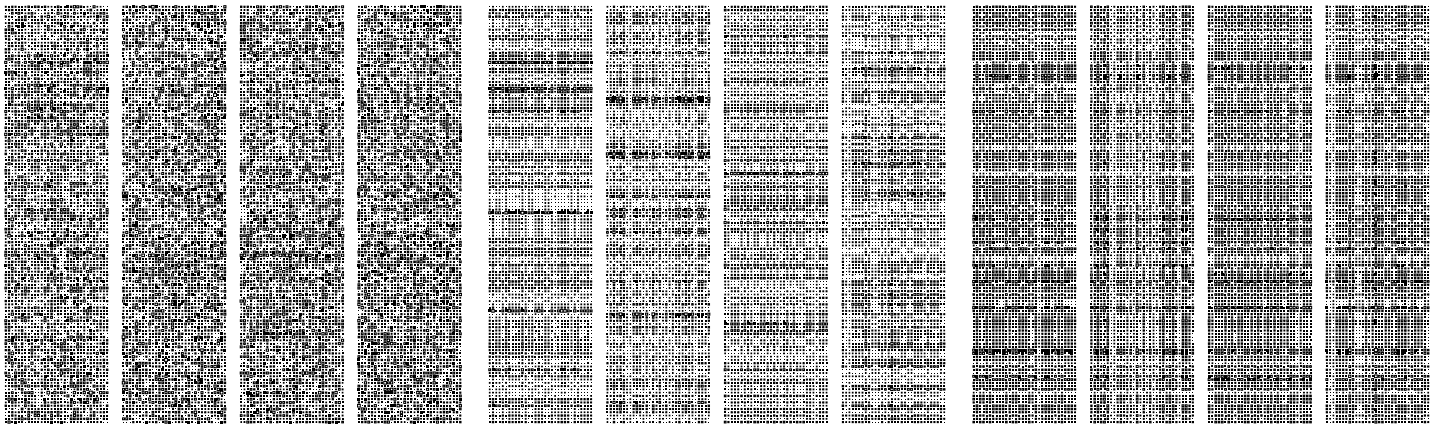


Figure 25: W_k, W_q and W_v

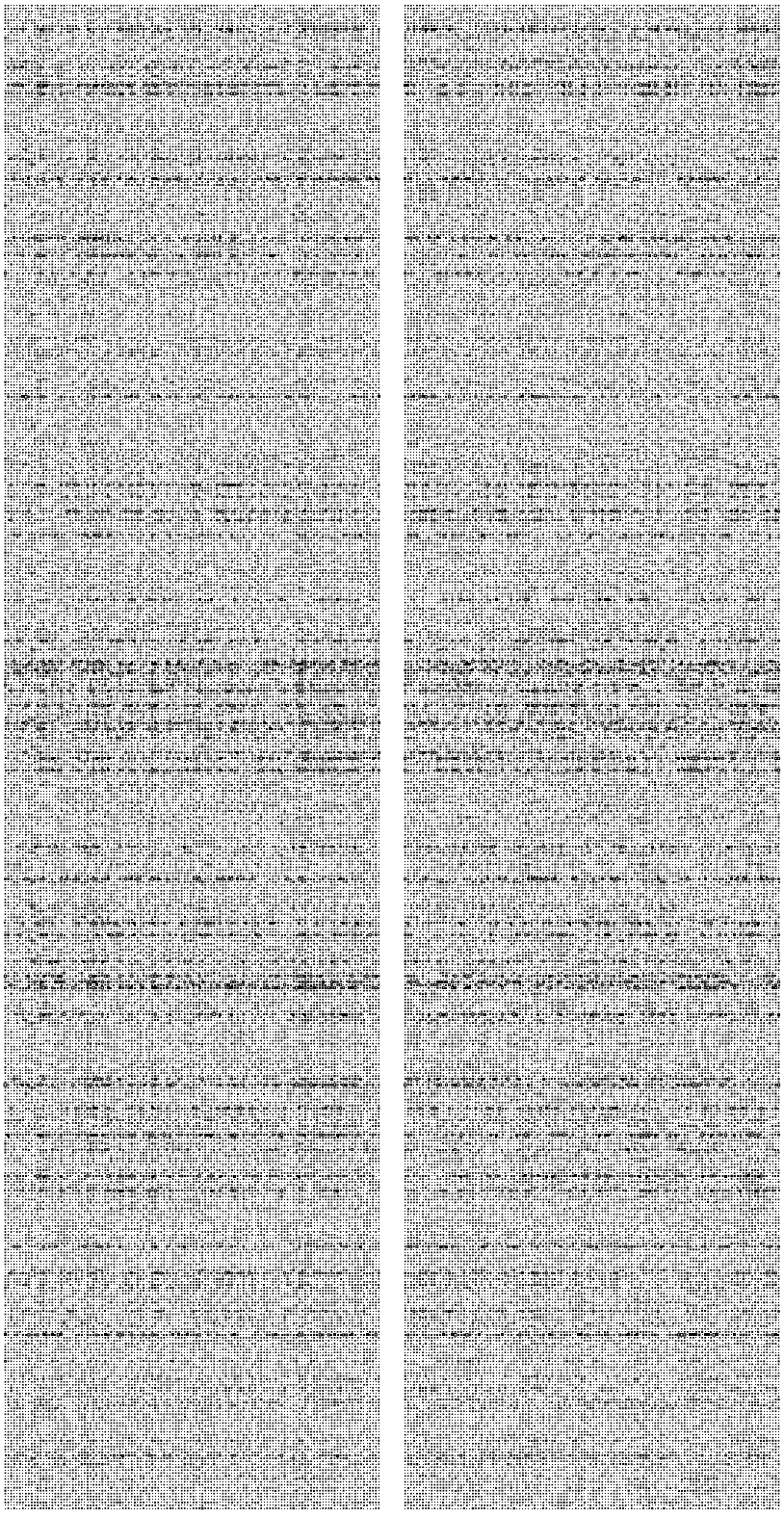


Figure 26: W_{in} and W_{out}^{T}

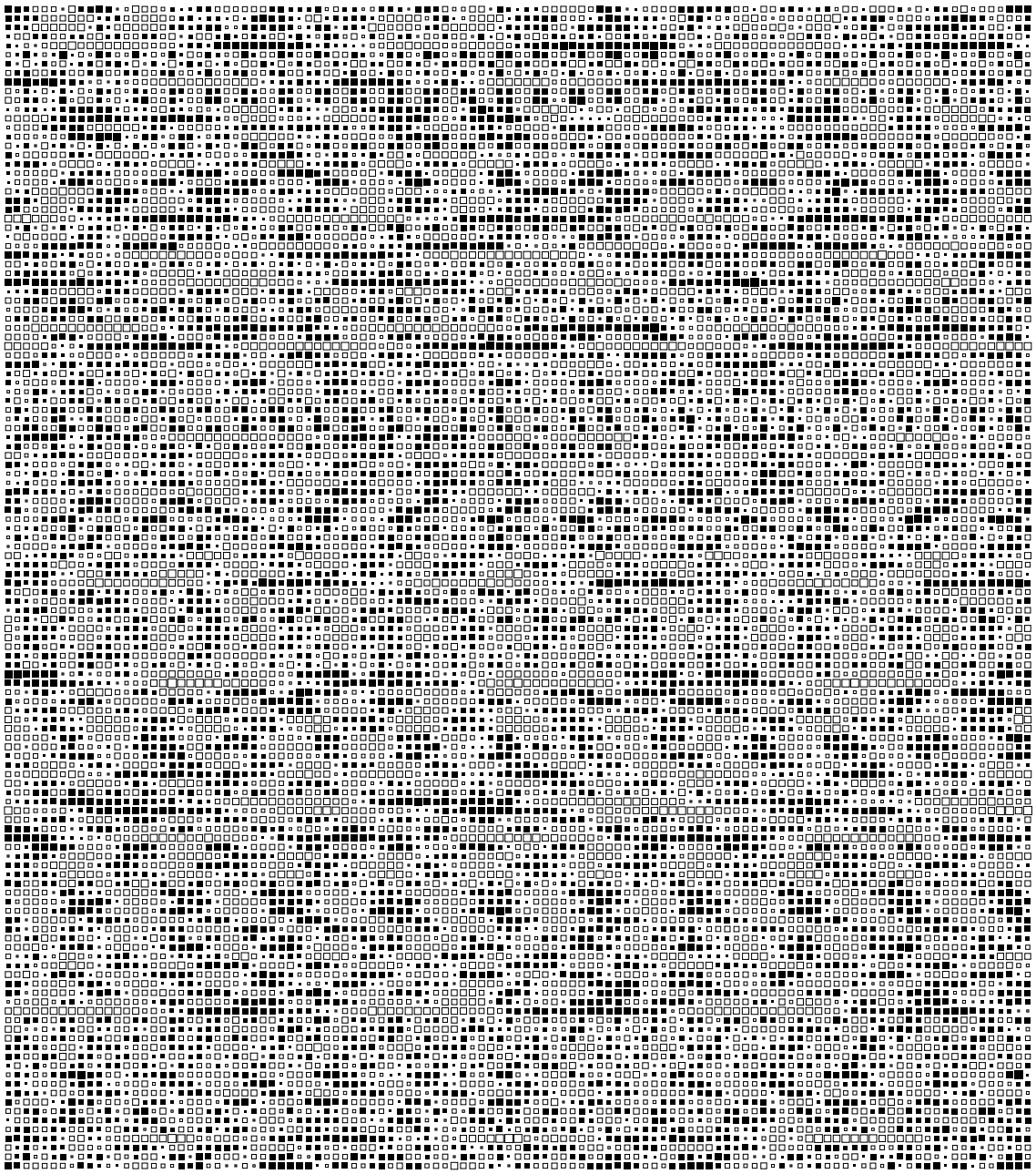


Figure 27: W_U