Discovering and Interpreting Shared Components for Sequence Continuation Tasks in a Large Language Model

Anonymous ACL submission

Abstract

While transformer models exhibit strong capabilities on linguistic tasks, their complex architectures make them difficult to interpret. Recent work has aimed to reverse engineer transformer models into human-readable representations called circuits that implement algorithmic functions. We extend this research by analyzing and comparing circuits for similar sequence continuation tasks, which include increasing sequences of Arabic numerals, number words, and months. By applying circuit interpretability analysis, we identify a key sub-circuit in GPT-2 responsible for detecting sequence members and for predicting the next member in a sequence. Our analysis reveals that semantically related sequences rely on shared circuit subgraphs with analogous roles. Overall, documenting shared computational structures enables better model behavior predictions, identification of errors, and safer editing procedures. This mechanistic understanding of transformers is a critical step towards building more robust, aligned, and interpretable language models.

1 Introduction

Transformer-based large language models (LLMs) like GPT-4 have demonstrated impressive natural language capabilities across a variety of tasks (Brown et al., 2020; Bubeck et al., 2023). However, these models largely remain black boxes due to their complex, densely connected architectures. Understanding how these models work is important for ensuring safe and aligned deployment, especially as they are already being used in high-impact real-world settings (Zhang et al., 2022; Caldarini et al., 2022; Miceli-Barone et al., 2023).

Several researchers argue that the ability to interpret AI decisions is essential for the safe implementation of sophisticated machine learning technologies (Hendrycks and Mazeika, 2022; Barez et al., 2023). Previous studies show that AI interpretability is vital for AI safety, for catching deception, and for addressing misalignment (Barredo Arrieta et al., 2020; Amodei et al., 2016). Mechanistic interpretability, a sub-field of interpretability, aims to reverse engineer models into understandable components (such as neurons or attention heads) (Elhage et al., 2021). By uncovering underlying mechanisms, researchers can better predict model behaviors (Mu and Andreas, 2020; Foote et al., 2023) and understand emergent phenomena (Nanda et al., 2023; Quirke and Barez, 2023; Marks et al., 2023).

Recent work in interpretability has uncovered transformer circuits that implement simple linguistic tasks, such as identifying indirect objects in sentences (Wang et al., 2022). However, only a few studies have focused on the existence of shared circuits (Merullo et al., 2023), in which circuits utilize the same sub-circuits for similar tasks. Identifying shared circuits assists in aligning AI via methods such as model editing (Meng et al., 2023), which precisely targets problematic areas for more efficient re-alignment without erroneously altering healthy components. Documenting the existence of shared circuits enables safer, more predictable model editing with fewer risks, as editing a circuit may affect another if they share sub-circuits (Hoelscher-Obermaier et al., 2023). Therefore, interpretability enables safer alignment by understanding adverse effect prevention.

While models use the same components for different tasks, such as when there are far more tasks/features than neurons (Elhage et al., 2022), our focus is on locating components which are shared due to similar, re-usable functionality, and not for vastly different functionalities. Our work tackles the hypothesis that LLMs may re-use circuits across analogous tasks that share common abstractions. For instance, similar sequence continuation tasks, such as number words ("one two three") and months ("Jan Feb Mar"), can be analogously mapped to one another via the natural number abstraction (eg. one and Jan are mapped to 1).

As these tasks share a common abstraction, LLMs may have learned to efficiently re-use components that utilize shared patterns. Understanding how LLMs re-use components based on commonalities can shed light on how they represent and associate semantic concepts with one another (Gurnee and Tegmark, 2023). Not only would this enhance understanding of how LLMs actually perceive information, but it may have potential applications in transfer learning (Zhuang et al., 2020).

Thus, in this paper, we demonstrate the existence of shared circuits for similar sequence continuation tasks, as the similarity across these tasks is clear, allowing us to cleanly pinpoint functionality. Our key finding is that there exist shared sub-circuits between similar tasks in GPT-2 (Radford et al., 2019), where the shared components have the same functionality across tasks. As shown in Figure 1, the circuit for continuing a sequence of numerals shares a sub-circuit with the circuit for continuing a sequence of number words, which generally handles sequence continuation functionality.

The main contributions of this work are: (1) The discovery of shared circuits for sequence continuation tasks, (2) Finding that similar tasks utilize sub-circuits with the same functionality, and (3) A simple iterative approach to find circuits at a coarse granularity level. This advances our understanding of the mechanisms of how transformer models generalize concepts by re-using components.

2 Background and Related Work

Transformer Models. We analyze LLM transformer-based models with a vocabulary size V. The model takes an input sequence $(\mathsf{x}_1, \ldots, \mathsf{x}_p)$ where each $\mathsf{x}_i \in \{1, \ldots, V\}$. Tokens are mapped to d_e-dimensional embeddings by selecting the x_i-th column of $E \in \mathbb{R}^{d_e \times V}$ (Vaswani et al., 2017).

Attention Head. A transformer model consists of blocks of attention heads, which each consists of two matrices: the **QK** matrix that outputs the attention pattern $A_{i,j} \in \mathbb{R}^{N \times N}$, and the **OV** matrix that outputs to the residual stream. The output of an attention layer is the sum of attention heads $h_{i,j}$. We use the notation L.H for attention heads, where L is a layer index and H is a head index in layer L.

Multi-Layer Perceptron. Each attention layer output is passed to a Multi-Layer Perceptron (MLP). The MLPs in transformers are generally made of two linear layers with a ReLU activation function in between.

Residual Stream. Attention head and MLP outputs are added to the residual stream, from which components read from and write to. Components in non-adjacent layers are able to interact via indirect effects from the additivity of the residual stream (Elhage et al., 2021).

Circuit Discovery. To analyze computations within models, a recent approach has been to find *circuits*, which are subgraphs of neural networks that represent algorithmic tasks (Elhage et al., 2021). In transformer circuits, evidence has shown that in general, MLPs associate input information with features (Geva et al., 2020), while attention heads move information (Olsson et al., 2022).

Prior work has employed **causal interventions** to locate circuits for specific tasks (Meng et al., 2023; Vig et al., 2020), such as for the Indirect Object Identification (IOI) task, in which the goal is to complete sentences with the correct subject (Wang et al., 2022). One type of causal intervention is called **knockout**, which, after a model has processed a dataset, replaces (or *ablates*) the activations of certain components with other values, such as activations sampled from another distribution. The sampled activations may come from a *corrupted dataset*, which outputs the wrong answer, but resembles the same dataset without the information of interest (eg. "1 2 3" becomes "8 1 4" to preserve information about numbers, while removing sequence information). After running again, if the ablated nodes do not change model performance much, they are deemed as not part of a circuit of interest.

Another type of causal intervention is **activation patching**, which takes the corrupted dataset as input, and then restores the activations at a certain component with the original activations to observe how much that restored component recovers the original performance. **Path patching** is a different type of patching that allows for a more precise analysis of an intervention's effect on a particular path (Goldowsky-Dill et al., 2023). It can be performed by ablating component interactions, measuring the effect of one component on another. The Automatic Circuit DisCovery (ACDC) technique employs iterative patching automatically find circuit graphs for given tasks (Conmy et al., 2023); however, this technique only seeks to automate finding the connectivity of circuit graphs, and not their functionality interpretation.

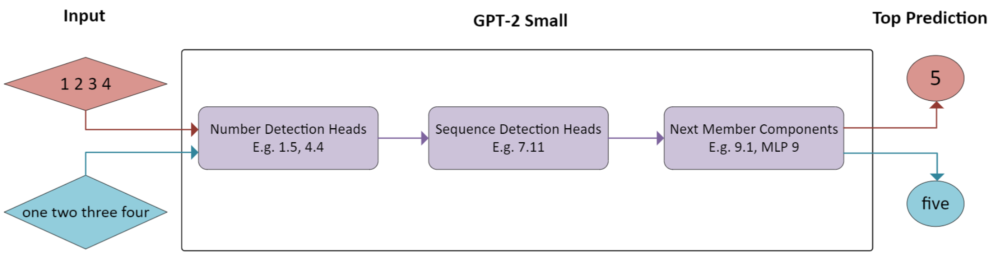


Figure 1: Simplified circuit show important components for the Increasing numerals (red) and Increasing Number Words (blue) tasks merged into one diagram. The purple portions denote a shared, entangled sub-circuit across both tasks. For demonstration simplicity, components exclusive to each tasks' circuit are not shown. In §5.2.1, "Number Detection" Head 4.4 is generalized as an "Adjacent Member Detection" Head.

Model Interpretability of Sequential Tasks. (Hanna et al., 2023) found circuits for "greater-than" sequence tasks; one such task, for instance, would be completing the sentence, "The war lasted from the year 1732 to the year 17", with any valid two-digit end years (years > 32). Greater-than tasks allow any year greater than a value to be valid, which differs from our sequence completion tasks that only have one valid answer. The authors noted that "similar tasks had similar, but not identical, circuits", but all the tasks they tested were on greater-than number tasks, and not on non-number tasks such as months. In our work, we study similar tasks that are more dissimilar in their content.

Shared Circuits for Similar Tasks. Locating shared circuits is a relatively new research topic. Previous studies have noted that circuits for the Induction task (Olsson et al., 2022) are found in circuits for the IOI task. Recently, (Merullo et al., 2023) discovered shared circuits for the IOI task and Colored Objects task (where the aim is to identify the correct color for an object given a set of colored objects). The authors utilized an intervention experiment to improve the Colored Objects circuit by modifying subject inhibition heads of the IOI circuits to inhibit the wrong color answers. In our paper, we focus on tasks which are much more similar and map to a common abstraction. While the IOI task and Colored Objects task both share similar sub-tasks such as "inhibiting tokens", the focus of our paper is on enhancing our understanding of how LLMs represent analogous concepts by discovering sub-circuits which represent common abstractions, instead of just shared sub-tasks.

3 Methodology

Circuit Discovery Process. Our approach begins by applying iterative pruning to obtain connectivities for circuits of similar tasks. Then, we employ methods to deduce component functionalities shared by similar tasks. We approach circuit discovery in two types of stages: [1]

1. *Connectivity Discovery* consists of applying causal mediation analysis techniques for identifying important connections for varying component granularity levels (eg. residual stream, attention head, MLP, neuron).

2. *Functionality Discovery* aims to describe the tasks handled by circuit components, labeling them with interpretable semantics.

3.1 Connectivity Discovery Methods

Metrics. We utilize the *logit difference* to measure model task capability by taking the difference between the correct token L_C and an incorrect token logit L_I. The incorrect logit may be chosen as a token that is not the correct token. To compare an ablated model with the unablated model, we employ the **performance score**, a percentage calculated as the logit difference of the ablated model over the logit difference of the unablated model.

Iterative Pruning for Nodes. To search for circuit components, we use a knockout method that ablates one candidate component (node) at a time and checks how much performance falls. This method begins with all the components as a *candidate circuit*. At each step, ablation is performed by patching in the mean activations of a corrupted dataset at a candidate node, plus all the nodes not in our candidate circuit. If performance falls below T_n, a user-defined *performance threshold*, the node is kept for the candidate circuit, as it is deemed necessary for the task. Else, it is removed.

[1]The methods we apply to one stage may also yield information about another stage.

We start by removing components from the last layer, continuing until the first layer; we call this procedure the *backward sweep*. At each layer during the backward sweep, we first ablate the layer's MLP, and then consider its attention heads. Next, we then prune again from the first layer to the last layer; we call this the *forward sweep*. At each layer during the forward sweep, we first ablate each attention heads, and then its MLP. We continue iterating by successive backward-forward sweeps, stopping when no new components are pruned during a sweep. The output is the unpruned node set.

This method may be considered as a simplified and coarser variation of ACDC (Conmy et al., 2023), which decomposes heads into key, query, and value (qkv) vector interactions. As head outputs deemed unimportant may also be unimportant when decomposed, our method first filters nodes at a coarse level, then decomposes heads into separate (qkv) nodes during edge pruning. [2]

Iterative Path Patching for Edges. After finding circuit nodes, we utilize path patching to obtain interactions (edges) between them. Edges denote nodes with high effects on other nodes [3]. We apply a form of iterative path patching which works backwards from the last layers by finding earlier components that affect them. First, we ablate the nodes pruned from iterative node pruning. Then, we ablate one candidate edge of the unablated nodes at a time. Using the same order as the backward sweep, we take a node as a *receiver* and find the *sender* nodes that have an important effect on it. If patching the effect of sender A on receiver B causes the model performance to fall below threshold T_e, the edge is kept; else, it is removed.

For example, if node pruning found a circuit that obtains a 85% score above $T_n = 80\%$, we now measure which circuits with the ablated nodes and the ablated candidate edge still have performance above $T_e = 80\%$ [4]. Performing node pruning before edge pruning filters out many nodes, reducing the number of edges to check. After edge pruning, nodes without edges are removed. This method has similarities to the path patching used by (Hanna et al., 2023), but with several differences, such as using our performance metric as a threshold.

3.2 Functionality Discovery Methods

Attention Pattern Analysis. We analyze the $\mathbf{QK}$ matrix of attention heads to track information movement from keys to queries. When we run attention pattern analysis on sequences comprised solely of sequence member tokens such as "1 2 3 4", there are no other 'non-sequence member' words to compare to, so it is hard to tell what 'type' of token each head is attending to. Thus, we measure what types of tokens the heads attend to by using prompts that contained these sequences within other types of tokens, such as "Table lost in March. Lamp lost in April."

Component Output Scores. We analyze head outputs by examining the values written to the residual stream via the heads' output matrices ($\mathbf{OV}$), allowing us to see what information is being passed by each head along in the circuit. These values are measured by component output scores; we utilize a "next sequence" score that measures how well a head, given sequence token I, outputs token $I + 1$. The details of this method are described in Appendix F.

Logit Lens. Logit lens is a method for understanding the internal representations by unembedding each layer output into vocabulary space and analyzing the top tokens (Nostalgebraist, 2020). We use logit lens to uncover the layer at which the predicted token goes from the 'last sequence member' to the 'next sequence member'.

4 Discovering Circuit Connectivity

In this section, we describe the experimental setup for our ablation experiments. We observe that there are multiple circuits, with slight variations between them, that have similar performances for the same task. However, we find that important heads are often found in most circuits, regardless of the method, metric or dataset choices. Thus, we focus more on the "big picture" comparison of scores and on the most important heads, and less on the exact variations between scores or the less important heads. We ran experiments on a NVIDIA A100 GPU.

Model. We test on GPT-2 Small (117M parameters), which has 144 heads and 12 MLPs.

Task Comparison. We compare increasing sequences of: (1) Arabic Numerals (or 'Numerals'),

[2] While the graphs found by ACDC utilize even finer granularity levels than just head decomposition, the authors of the paper note that different granularity levels are valid based on analysis goals (eg. (Hanna et al., 2023) analyze at a level without head decomposition). We find our chosen granularity level to be sufficient for analyzing shared circuits.

[3] As the residual stream allows for indirect effects, edges may be between components at non-adjacent layers,

[4] The edge pruning threshold T_e may be the same or different as the node pruning threshold T_n.

(2) Number Words, and (3) Months.

Datasets. We run a generated prompts dataset of length 4 sequences (eg. 1 2 3 4). We found that the model could continue Numerals sequences even past 1000. However, our focus in this paper is not on finding circuits only for Numeral sequences, but on prompt types that share a common abstraction. Thus, to better compare numbers to months, we use sequences ranging from 1 to 12.

For each task, we generate samples by placing our sequence members among non-sequence tokens. For instance, one sample may be 'Kyle was born in February. Anthony was born in March. Grant was born in April. Madison was born in'. Placing sequence members amongst non-sequence tokens allows us to evaluate the circuit representation of the shared sub-task of how the model selects sequence members from non-sequence members. We generate a total of 1536 samples per task; thus, there are 4608 total samples. More discussion about datasets is found in Appendix A.

Corrupted Datasets. We corrupt sequence information by using randomly chosen tokens of a similar sequence type (eg. '1 2 3' is replaced with '8 1 4'). The non-sequence tokens are kept the same, while the sequence members are replaced.

Metric. We measure using logit difference using the last sequence member as the incorrect token (eg. 1 2 3 has 4 as correct, and 3 as incorrect).

4.1 Shared Sub-Circuits for Similar Sequence Continuation Tasks

We discover shared sub-circuits across the three sequence continuation tasks. Figure 2 combines all three circuits into one graph [5]. These circuits were found using a performance threshold of $T_n = T_e = 80\%$. There is a sub-circuit found across the circuits for all three tasks, which includes heads 4.4, 7.11 and 9.1, which we show to be important in Table 2. As seen in both Figure 2 and in Table 4 in Appendix C, in which only head 0.5 of the Numerals circuit is not part of the Number Words circuit, the Numerals circuit is nearly a subset of the Number Words circuit. This suggests that the Number Words circuit uses the Numerals circuit as a sub-circuit, but requires additional components to make accurate predictions.

In Table 1, we compare every task's circuit with other similar tasks, isolating each circuit by resampling ablation on non-circuit components. First, we observe that in general, the model cannot perform well on these tasks for non-sequence-task circuits. For instance, we show that the model has negative performance for all tasks when run on the IOI circuit. The negative values mean that the $(L_C) - (L_I) < 0$ in the ablated circuit, indicating bad performance.

We observe that for the Numerals task, the model performs better on the Number Words circuit than the Numerals circuit, which may be because the Numerals circuit is nearly a sub-circuit of the Number Words circuit. It is possible to find a Numerals circuit with higher performance by setting the threshold higher. However, this paper's pruning methods attempt to find minimal circuits with only necessary components above a certain threshold; they do not seek to find the circuit with the most optimal performance [6].

For the Number Words task, the model only performs well with the Number Words circuit, as this task may require more components than the other two. On the other hand, for the Months task, the model performs even better than the unblated circuit for all sequence-task circuits, indicating that this task may not require as many components as the other two. Due to components such as inhibition heads (Wang et al., 2022), ablating certain heads may allow the model to perform better for specific tasks, though may hurt its ability on other tasks. Overall, these results show that these tasks do not use the exact same circuit, but may have partially good performance on other sequence task's circuits due to shared sub-circuit(s).

Several important attention heads are identified across various circuits. We define a head as *important* if their ablation from a circuit causes an average drop of at least -20% performance for all tasks [7]. Table 2 compares the importance of these attention heads for our tasks. We note that ablating heads 0.1, 4.4, 7.11, and 9.1 cause drops >20% for all three circuits, while ablating 1.5% causes a drop >20% for Numerals and Number Words circuits.

MLP Connectivity. For all tasks, we find that several MLP ablations cause a >20% performance drop. In particular, MLP 9 causes a substantial drop of more than 90%. These results are found in

[5]Due to the (qkv) circuit's large display size, we show the circuits with (qkv) decomposition in Appendix C.

[6]One can obtain circuits with >100% performance by setting the threshold to be 100.

[7]20% is chosen due to using $T_n = 80\%$, so that for many removal order variations, a component with a 20% importance cannot be removed, unless there are alternative backups.

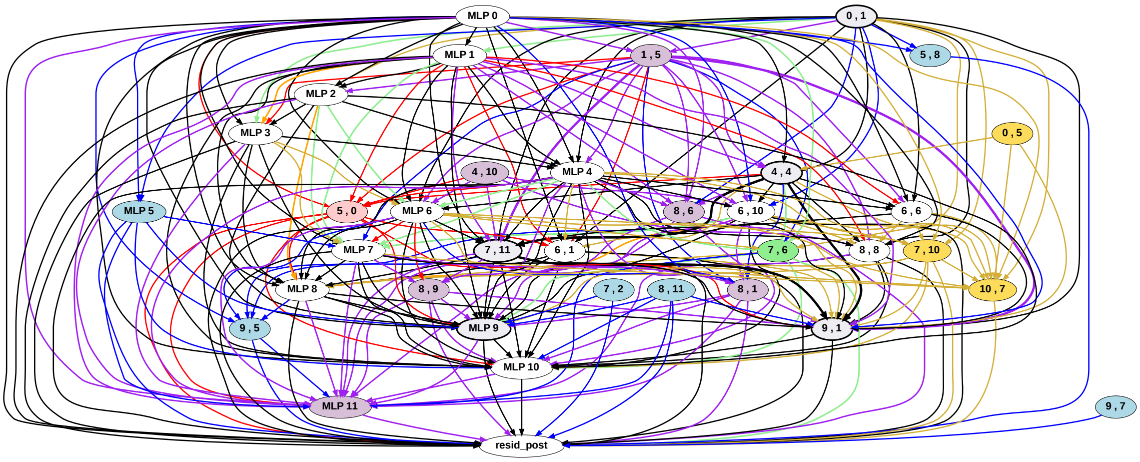


Figure 2: A Numerals Sequence Circuit (red), a Number Words Sequence Circuit (blue), a Months Sequence Circuit (gold). The overlapping sub-circuit parts are coded as follows: Numerals and Number Words only are in purple, Numerals and Months only are in orange, Number Words and Months only are in green, and All Three Tasks are in white with **black edges**. The most important sub-circuit components are in gray with a **bold outline**. Resid_post denotes the residual stream state right before the linear unembedding to logits.

Table 1: Performance Scores for Figure 2 Circuits' Components (cols) run on Similar Tasks (rows)

	Numerals Task	NumWords Task	Months Task
Numerals Circuit	81.01%	48.41%	113.52%
Number Words Circuit	87.35%	81.11%	103.64%
Months Circuit	43.74%	32.36%	80.30%
IOI Circuit	-6.70%	-15.82%	-9.20%

Appendix D.

5 Explaining Shared Component Functionalities

5.1 Sub-Circuit Hypothesis

We hypothesize how the important shared components for the three tasks work together as a *functional* sub-circuit. We define sub-tasks that all three sequence continuation tasks share: (1) Identifying Sequence Members and (2) Predicting the Next Member after the Most Recent Member.

Our hypothesis is that early heads, in particular 1.5 and 4.4, identify similar, adjacent sequence members, such as numbers or Months, without yet attending to the distinction of which numbers should be focused on more than others. Following this, information is passed further along the model to heads, such as 7.11, to discern consecutive number sequences and deem the two most recent elements as more significant. This information is then conveyed to head 9.1 to put more emphasis on predicting the next element in the sequence. Lastly, the next element calculation is done primarily by MLP 9. Thus, this sub-circuit would represent an algorithm that carries out the sub-tasks shared for all three tasks. This section details evidence that supports this circuit hypothesis.

5.2 Attention Head Functionality

Duplicate Head Head 0.1 was noted to be a Duplicate Token Head by (Wang et al., 2022), in which it recognizes repeating patterns. As we did not note that 0.1 had any effects on sequence members in particular, given our non-sequence token patterns, it is likely that 0.1 is recognizing all repeating patterns in general, which is prevalent in our dataset. Though it plays an important role for this sub-task, it does not appear specific to sequence continuation.

Table 2: Drop in Task Performance when a Head is Removed from a Circuit in Figure 2.

Important Head	Numerals	NumWords	Months
0.1	-44.29%	-78.74%	-52.10%
4.4	-33.19%	-34.11%	-73.16%
7.11	-41.64%	-44.78%	-45.37%
9.1	-34.94%	-27.74%	-43.03%
1.5	-27.83%	-18.65%	-

5.2.1 Sequence Member Detection Heads

We discover a "similar member" detection head 1.5, and a "sequence member detection" 4.4. Attention pattern analysis reveals that these heads detect how sequence members (as queries) attend to sequence members (as keys) of the same type, such as numerals. To determine if this detection only occurs if the sequence members are in sequential order, or if this occurs even if they are not, we input prompts with Numerals in random order but with Months in sequential order. In Figure 3, we observe, for head 1.5, similar types attend to similar types. However, for head 4.4, Months attend to Months, but Numerals do not attend to Numerals, as the Numerals are not in sequential order. Therefore, in general, both heads 1.5 and 4.4 appear to detect similar token types that belong to an ordinal sequence such Numerals or Months, but head 4.4 acts even more specifically as an adjacent sequence member detection head. More discussion about these attention patterns are in Appendix G.

5.2.2 Last Sequence Token Detection Head

In Figure 2, there is an edge from heads 1.5 and 4.4 to 7.11, showing 7.11 obtaining sequence token information from earlier heads. Then, we observe in Figure 4 that for head 7.11, query tokens attend to its previous key tokens, indicating 7.11 acts like a "Previous Token" head. Noticeably, at the last query token, the strongest attention appears to be from the non-sequence tokens to the sequence tokens. This head may "ordering" identified sequence tokens to send to the last token, or it may be figuring out the pattern at which token the model should predict the next member of the identified sequence; for instance, it notices that after each non-number token often follows the next member of the number sequence.

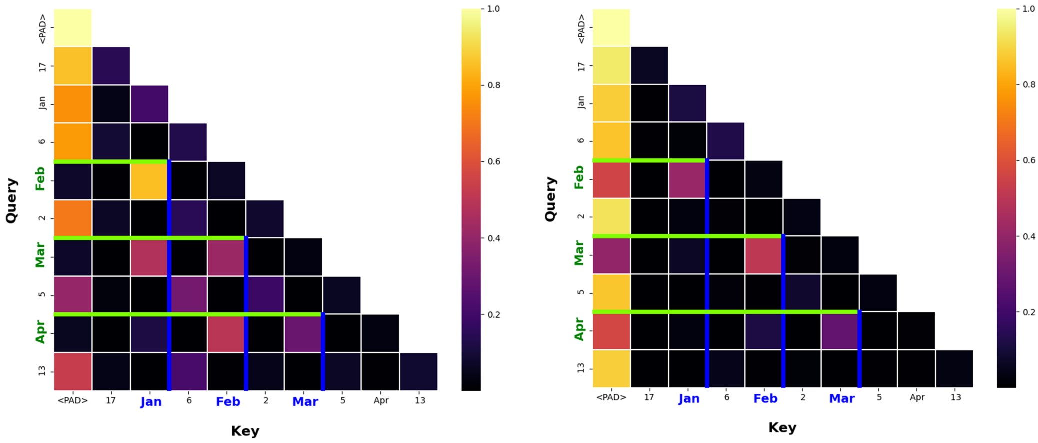
5.2.3 Next Sequence Head

Figure 2 shows that head 9.1 receives information from both head 4.4 and 7.11. Head 9.1, shown in Figure 5, pays strong attention to only the last member of the sequence, and it appears to attend even stronger to the last member than 7.11.

Next Sequence Scores. To check that head 9.1 outputs next sequence tokens, we study its component output scores. Table 3 shows that given a numeral token I as input (eg. 1), head 9.1 often outputs a token I+1 or higher (eg. 2). For numerals between 1 and 100, its next score is 87%, while its copy score is 59%. We also note that the next sequence scores of most heads are low, with an average of 3.29%, and that head 9.1 has the highest next sequence score. Thus, it seems to function as a "next sequence head". This is reinforced by the next sequence score for number words, which is 90.63%, while the average for all attention heads is 2.97%. Although head 9.1 does not appear to output months given any month token, we observe something peculiar: 9.1 is the *only* head that will output the next rank given a month (eg. given "February", output "third", and its "next rank given month" score is 31.25%. This appears to be related to how months can be mapped onto ranks.

Table 3: The top-3 tokens output tokens after OV Unembedding head 9.1 for several input tokens.

Token	**Top-3 Tokens after Unembed**
'78'	' 79', '80', '81'
'six'	' seventh', ' eighth', ' seven'
'August'	'ighth', 'eighth', 'ninth'



(a) (b)

Figure 3: Attention Patterns for (a) Head 1.5 and (b) Head 4.4. Lighter colors mean higher attention values. For each of these detection patterns, the query is shown in green, and the key is shown in blue. The Months are in sequential order, but the Numerals are not. We observe for head 1.5, similar types attend to similar types. However, for head 4.4, Months attend to Months, but that Numerals do not attend to Numerals.

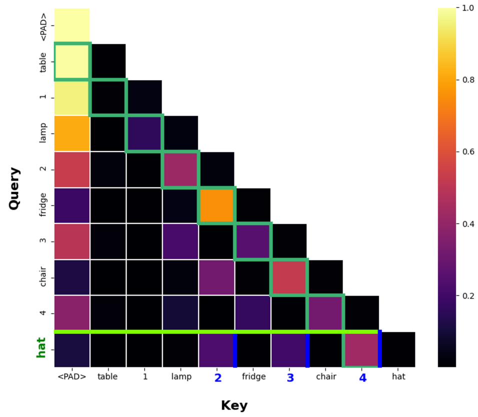


Figure 4: Head 7.11 Attention Pattern for Numerals. At the last token, head 7.11 has attends more to later numbers. The previous token offset pattern is in dark green.

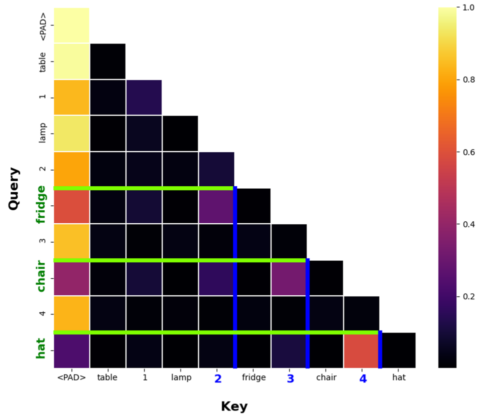


Figure 5: Head 9.1 Attention Patterns shows it pays strong attention to only the most recent number.

5.3 MLP Functionality

For all sequence types, logit lens reveals that the model does not predict the correct answer before MLP 9. However, after the information is processed through MLP 9, the model outputs the next sequence member. These findings suggests that MLP 9 is largely responsible for finding the next sequence member, even more so than Head 9.1, which may just be boosting this information and/or acting as a backup component. Logit lens results can be found in Appendix §D.

6 Conclusion

Understanding the inner workings of neural networks such as transformers is essential for fostering alignment and safety. In this paper, we identify that across similar sequence continuation tasks, there exist shared sub-circuits that exhibit similar functionality. Specifically, we find sequence token detection heads and components associated with next sequence outputs. The aim of this work to advance our understanding of how transformers discover and leverage shared computational structures across similar tasks. By locating and comparing these circuits, we hope to gain insight into both the inductive biases that allow efficient generalization in these models and their semantic representations of abstract concepts, which may provide evidence for hierarchical associations. In future work, we plan to investigate how shared circuits affects model editing.

Limitations

As the research topic of our work is relatively new, the aim of this paper is to first investigate shared circuits for simple tasks. This way, later work may build upon it to look for shared circuits for more complex tasks that are more important for AI safety. We discuss limitations of this paper in this section.

Number of Models. As it is common for many well-received interpretability papers to focus on analyzing one model (Wang et al., 2022; Conmy et al., 2023; Hanna et al., 2023; Nanda et al., 2023) or even just one attention head (McDougall et al., 2023), we only analyze one model. We plan to investigate this phenomenon for multiple models, including toy models and larger models, that can perform other types of sequence continuation in future work. This future work may include Fibonacci sequences, or comparing circuits for adding 2 (e.g. 2 4 6 8 10) vs circuits for multiplying 2 (e.g. 2 4 8 16 24), studying if the model uses diverging circuits to differentiate between the two tasks while still computing parts of them using shared circuits.

Dataset Size. As there are only twelve months, the number of possible continuing sequences was limited. Additionally, even if months were not used, GPT-2 also has limited prediction ability for number word sequences, as detailed in Appendix A. However, our dataset size for the months continuation task fully captures all of the months; in contrast, a small dataset size brings more issues when it does not capture all of the true distribution.

Task Complexity. Though sequence continuation may be seen as possibly simply associating what comes after every sequence member, the aim of our work is to find how this task is internally represented, even if it is done by simple association. Previous works have investigated how association is done by MLPs (Geva et al., 2020), so understanding how language models associate information can shed light on how it represents similar concepts.

Future Work. In the future, we plan to dive deeper into this study, such as by performing neuron-level and feature-level analysis, and by examining components exclusive to the number words task to see if they handle mapping between abstract representations of numbers and number words. Our future work plans also include analyzing the effects of model editing on shared, entangled circuits. These may include quantifying the relationship between circuit entanglement and editing impact (which may be done via embedding space projection (Dar et al., 2022)), modifying the sub-circuit used for sub-task S and observing if the ability to recognize S in multiple tasks is destroyed, and utilizing methods such as model steering to edit task S to perform a similar task S' (Turner et al., 2023; Merullo et al., 2023).

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in ai safety. *arXiv: Learning*, abs/1606.06565.

Fazl Barez, Hosien Hasanbieg, and Alesandro Abbate. 2023. System iii: Learning with domain knowledge for safety constraints.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.

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

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4.

Guendalina Caldarini, Sardar Jaf, and Kenneth McGarry. 2022. A literature survey of recent advances in chatbots. *Information*, 13(1):41.

Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*.

Guy Dar, Mor Geva, Ankit Gupta, and Jonathan Berant. 2022. Analyzing transformers in embedding space.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. 2022. Toy models of superposition. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/toy_model/index.html.

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

Alex Foote, Neel Nanda, Esben Kran, Ioannis Konstas, Shay Cohen, and Fazl Barez. 2023. Neuron to graph: Interpreting language model neurons at scale. In *Proceedings of the Trustworthy and Reliable Large-Scale Machine Learning Models Workshop at ICLR*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. 2023. Localizing model behavior with path patching.

Wes Gurnee and Max Tegmark. 2023. Language models represent space and time.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model.

Dan Hendrycks and Mantas Mazeika. 2022. X-risk analysis for ai research. *arXiv preprint arXiv:2206.05862*.

Jason Hoelscher-Obermaier, Julia Persson, Esben Kran, Ioannis Konstas, and Fazl Barez. 2023. Detecting edit failures in large language models: An improved specificity benchmark.

Luke Marks, Amir Abdullah, Luna Mendez, Rauno Arike, Philip Torr, and Fazl Barez. 2023. Interpreting reward models in rlhf-tuned language models using sparse autoencoders.

Callum McDougall, Arthur Conmy, Cody Rushing, Thomas McGrath, and Neel Nanda. 2023. Copy suppression: Comprehensively understanding an attention head.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2023. Locating and editing factual associations in gpt.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2023. Circuit component reuse across tasks in transformer language models.

Antonio Valerio Miceli-Barone, Fazl Barez, Ioannis Konstas, and Shay B. Cohen. 2023. The larger they are, the harder they fail: Language models do not recognize identifier swaps in python.

Jesse Mu and Jacob Andreas. 2020. Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, 33:17153–17163.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability.

Nostalgebraist. 2020. Interpreting gpt: The logit lens. https://www.alignmentforum.org/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. Accessed: [Insert Date Here].

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

Philip Quirke and Fazl Barez. 2023. Understanding addition in transformers.

Alec Radford, Jeff Wu, Rewon Child, D. Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. 2023. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks.

Alexander Matt Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. 2023. Activation addition: Steering language models without optimization.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small.

Audrey Zhang, Liang Xing, James Zou, et al. 2022. Shifting machine learning for healthcare from development to deployment and from models to data. *Nature Biomedical Engineering*, 6:1330–1345.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning.

A Dataset Details

Code and data will be released after review.

Dataset Generation Procedure. To generate each sample, we place each sequence within a specific template that is generated from an abstract template. For instance, the abstract template of '<name 1> born in <seq mem A>. <name 2> born in <seq mem B>. <name 3> born in' can fill in names with (Kyle, Anthony, Madison) to make a specific template. Then, the specific template can be filled in with sequence members (February, March) to create the sample 'Kyle was born in February. Anthony was born in March. Madison was born in'. We generate 1024 samples from each of three abstract templates, where each sequence (eg. 1 2 3, or 8 9 10) is represented the same number of times, for a total of 1536 samples per task. We use single tokens for all tokens in each sample. We choose samples such that the model outputs the correct answer with at least twice as high probability as the incorrect answer's probability. Each specific template must also meet these conditions for all sequences (eg. must work for 1 2 3, 8 9 10, and two three four); else, it is not used. Each template is represented in equal proportion.

We use the same templates for all tasks. For example: given the sample for the months task "Ham was bought in February. Egg was bought in March. Bread was bought in April. Steak was bought in", we use the same non-sequence tokens to make a sample for the digits task: "Ham was bought in 2. Egg was bought in 3. Bread was bought in 4. Steak was bought in".

The three templates we used are: <name> born in, <item> lost in, <item> done in. We choose from a set of 136 names and 100 items.

Originally, we use the token "was" in our samples (eg. "Steak was sold in March.") However, we find that the prediction outcomes were largely the same whether we included "was" or not. Thus, although "was" would make the sentences sound more natural to a human, we choose to omit it. Additionally, this allows to reduce the memory usage while running in Colab.

Random Words vs Meaningful Sentences. We find that using random words as non-sequence tokens could also allow the model to sometimes predict the next sequence member correctly. However, this did not always occur; thus, we choose to use semantically meaningful templates instead.

Sequence Member Input Positions. We did not construct samples such that there are different intervals between the sequence members, placing them at different positions in the input (eg. "1 2 house fork 3" or "1 house fork 2 3"), because we want the model to be able to predict the next sequence member with high probability. Thus, we give it an in-context pattern where after every random word, it should predict a sequence member.

Sequence Length. We find that using sequences with four members allows the model to consistently obtain high probability predictions for the correct answer for all three tasks. For continuing sequences without non-sequence members, four members is usually enough to obtain a correct token probability of around 90% or more for the three tasks, within a certain range (eg. not above twenty for number words for GPT-2).

Model Sequence Continuation Abilities. For number words, as GPT-2 Small does not seem to be able to continue number word sequences higher than twenty, even when giving it the starting prefix with and without hyphens (eg. twenty or twenty- for twenty-one). We add a space in front of each number word as without the space in front, the model tokenizer would break some words greater than ten into more than one token (eg. eleven into two tokens, and seventeen into three tokens), while we aim for all our samples in a dataset to have the same number of tokens. Similarly, for digit sequences there were cases where it would break the answer into multiple tokens (eg. in the 500-600 digit range, sometimes the next token predicted would be "5", and sometimes it would be "524").

Corrupted Dataset Details. We ensure that our randomly chosen sequence does not contain any elements in sequence for the last two elements of the input, as if the last two elements are not sequential, sequence continuation cannot successfully occur. We also test variations of several corruptions other than randomly chosen tokens of a similar sequence type, such as repeats and permutations. Overall, the most important components remain the same regardless of the ablation dataset and metric choices.

Other Task Datasets. We also look for similarities between other types of tasks, such as decreasing sequences, greater-than sequences, and alphabet sequences. However, while there were a few shared circuit overlaps between these tasks and three main tasks of this paper, there were more dissimilarities. Thus, we mainly focus on the similarities of the three tasks of this paper.

A.1 IOI Circuit.

The IOI circuit we use for comparison in Table 1 uses all MLPs and the following heads:

(0, 1), (0, 10), (2, 2), (3, 0), (4, 11), (5, 5), (5, 8), (5, 9), (6, 9),

(7, 3), (7, 9), (8, 6), (8, 10), (9, 0), (9, 6), (9, 7), (9, 9),

(10, 0), (10, 1), (10, 2), (10, 6), (10, 7), (10, 10), (11, 2), (11, 9), (11, 10)

B Computational Resources and Packages

For each task, the node and edge iterative methods took a total of 1 to 2 hours to run on an A100 GPU. The code for the experiments was written in Python, utilizing the TransformerLens package, and were run on Colab Pro+.

C Individual Circuit Results

Figure 6 shows a Numerals circuit, Figure 7 shows a Number Words circuit, and Figure 8 shows a Months circuit, each with Attention Head Decomposition. In Table 4, we show the result of dropping each head from the circuit shown in each of the Figures.

In Table 5, we show the result of dropping each head from the *fully unablated* circuit shown in each of the Figures. While Head 0.1 is of little importance when using the full circuit for the Numerals task with a -4.60% performance drop when ablated, it is of very significant importance for the Number Words task, with a -91.90% performance drop when ablated. Similar results are found for heads 4.4 and 9.1. This may occur because the model has learned multiple "backup circuits or paths" for the Numerals task, which activate when main components are ablated; it may also suggest that these heads are not important when the full circuit is present and are only important when certain components are ablated, acting as backup. The results also demonstrates that, for the Months task, the model places different importance on the heads than for the other two tasks. Overall, this shows that for each task, though the model re-uses many of the same important circuit parts, the importance of each part for each task varies greatly.

D MLP Analysis Details

In Table 6, we show the performance drop for the three tasks when ablating each MLP from the full, unablated circuit. We note that MLP 0 and MLP 9 are highly important. MLP 0 may be important due to acting as a "further embedding" after the embedding layer, which embeds the tokens into latent space. For all numeral sequence-member only samples ("1 2 3 4" to "8 9 10 11"), we find with logit lens that the "last sequence member" (eg. for 1 2 3, this is "3") is always output at some layer between MLP 6 to MLP 8. However, after MLP 9 to the last MLP, the output is always the next sequence member. The logit lens results for the top-3 tokens at each layer for a sample with non-sequence-members is shown in Table 7, and the logit lens results for a sample with only sequence-members is shown in Table 8. This pattern occurs in 1531 out of 1536, or 99.67%, of the samples with non-sequence-members. The anomalies have MLP 8 predicting the correct answer of '5' or '7'.

However, for number words with only sequence-members, MLP 9's role is not so clear. In some cases, MLP 8 will output the last sequence member and MLP will output the next one. In other cases, MLP 8 will output the last sequence member as a numeral, and MLP 9 will output the next sequence member as a number word. Yet in other cases, MLP 8 will output a number word related token, such as "thousand" or "teen", and MLP 9 will output the correct answer. For one sample, "six seven eight nine", the token '10' is outputed by MLP 9, and only until MLP 11 does the output become 'ten'. Table 9 displays a number words prompt's results.

The pattern of MLP 8 outputting a sequence member before MLP 9 outputs the next sequence member occurs in 1396 out of 1536, or 90.89%, of samples with non-sequence-members. The main culprits where this does not occur are for sequences that have correct answers of "seven" (in which MLP 8 outputs "seven") or "ten" (in which MLP 9 outputs '10' and MLP 10 outputs 'ten'). These results suggest that the role of MLP 9 is more nuanced than simply acting as a key:value store for next sequence members. Instead, this task may be distributed across various components, with MLP 9 being one of the most important parts for this task.

For months, all the samples with only sequence-members have the last sequence member at MLP 8, and the next sequence member at MLP 9. For samples with non-sequence-members, this occurs

Table 4: All Head Drops from Circuits of Figure 2.

Important Head	Numerals	NumWords	Months	Average
0.1	-44.29%	-78.74%	-52.10%	-58.38%
4.4	-33.19%	-34.11%	-73.16%	-46.82%
7.11	-41.64%	-44.78%	-45.37%	-43.93%
9.1	-34.94%	-27.74%	-43.03%	-35.24%
1.5	-27.83%	-18.65%	-	-23.24%
6.10	-14.00%	-24.28%	-16.90%	-18.39%
10.7	-	-	-13.1%	-13.10%
8.8	-15.23%	-13.21%	-10.15%	-12.86%
8.1	-12.93%	-12.61%	-	-12.77%
8.11	-	-10.86%	-	-10.86%
6.6	-7.56%	-9.70%	-8.93%	-8.73%
8.6	-11.02%	-6.22%	-	-8.62%
7.10	-	-	-6.25%	-6.25%
6.1	-10.28%	-4.49%	-3.77%	-6.18%
4.10	-4.87%	-5.73%	-	-5.30%
5.8	-	-5.15%	-	-5.15%
5.0	-5.02%	-	-	-5.02%
7.6	-	-4.96%	-5.23%	-5.10%
9.5	-	-5.84%	-3.77%	-4.81%
0.5	-	-	-3.79%	-3.79%
8.9	-4.09%	-3.36%	-	-3.72%
9.7	-	-3.08%	-	-3.08%
7.2	-	-2.84%	-	-2.84%

Table 5: Drop in Task Performance when a Head is Removed from the Full, Unablated (Original) Circuit.

Important Head	Numerals	NumWords	Months
0.1	-4.60%	-91.90%	-29.89%
4.4	-13.10%	-52.08%	-54.40%
7.11	-47.21%	-61.51%	-46.63%
9.1	-8.78%	-29.93%	-44.01%
1.5	-14.30%	-38.03%	-13.15

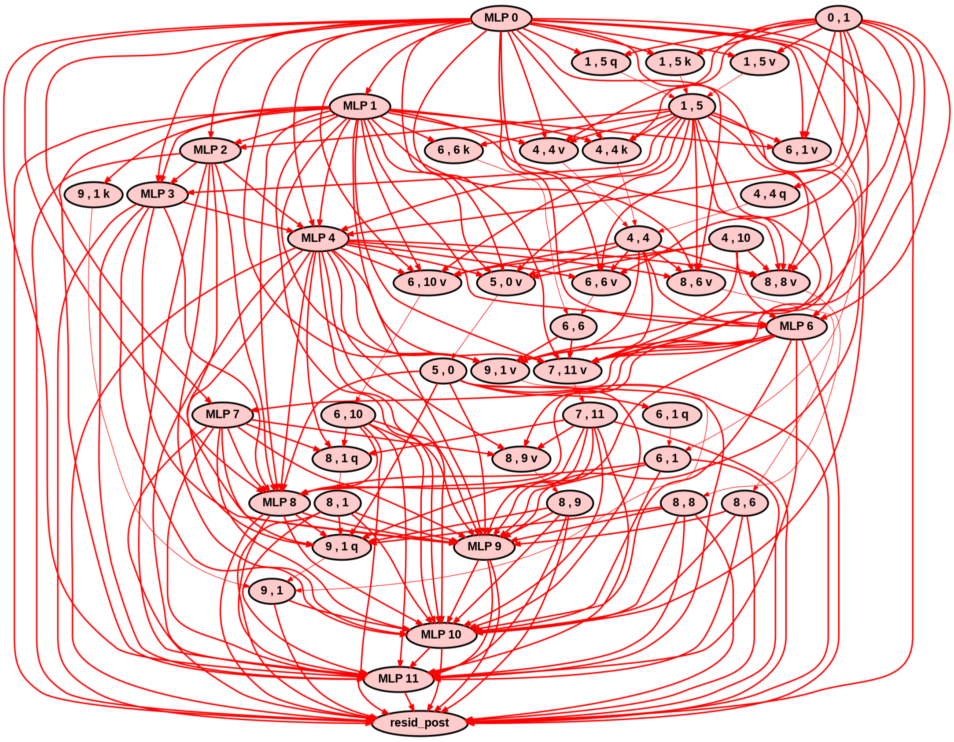


Figure 6: Numerals Circuit with Attention Head (QKV) Decomposition.

in 1495 out of 1536, or 97.33%, cases. The anomalies are samples that have the correct answer of "September", in which MLP 8 will output September. Table 10 shows that for the sample with only sequence-members that has the correct answer of "September", this does not occur, but strangely, MLP 0 will output "Aug" while MLPs 1 to MLP 5 will output years. It is possible that the sequence of months is more predictable than the other sequences. This is because for numerals and number words, a sequence of numerals doesn't always result in the next one, as there can be cases in natural language where "1 2 3 4" results in "55" because it is recording counts in general, or there may be some non-linear growth. Unlike numbers, months are more constrained in a smaller range.

E Iterative Method Details

Instead of absolute impact on performance score, we can also use relative impact. This means that the removal won't cause it to go down by more than 0.01 of the existing score, rather than an absolute threshold of 80%. However, this still doesn't take combos into account. one edge removal may make it 0.01, and another 0.01, but doesn't mean their combined effect is also 0.02; it may be more. Thus, order of removal appears to an impact on the circuit that is found.

Table 6: Drop in Task Performance when a MLP is Removed from the Full, Unablated (Original) Circuit.

MLP	Numerals	NumWords	Months
0	-62.58%	-95.98%	-84.80%
1	-9.28%	-34.71%	-8.30%
2	-2.68%	-20.18%	-16.40%
3	-2.67%	-18.19%	-9.33%
4	-14.19%	-49.24%	-23.88%
5	-12.64%	-25.16%	6.42%
6	-15.83%	-33.46%	-10.22%
7	-11.90%	-29.71%	-19.42%
8	-25.19%	-43.17%	-41.33%
9	-71.33%	-84.10%	-83.97%
10	-32.71%	-42.09%	-32.53%
11	-21.16%	-24.97%	-19.50%

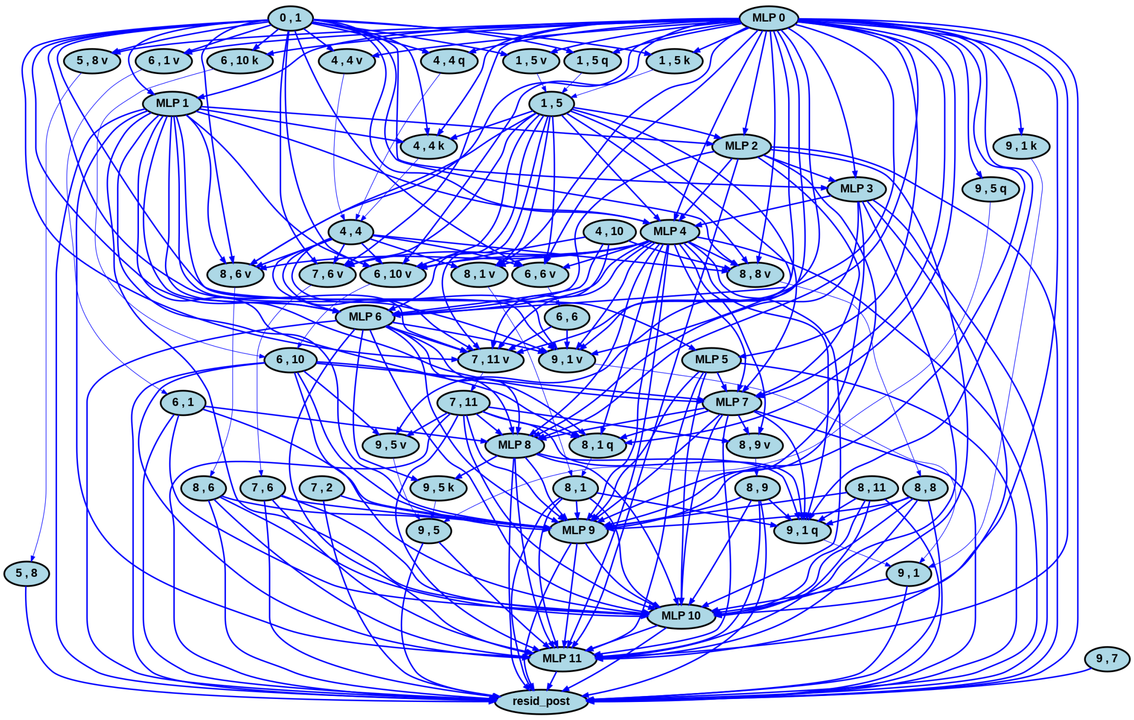


Figure 7: Number Words Circuit with Attention Head (QKV) Decomposition.

Table 7: Logit Lens- "Anne born in 2. Chelsea born in 3. Jeremy born in 4. Craig born in 5. Elizabeth born in"

MLP	Top-3 Tokens
0	order, the, particular
1	the, order, a
2	the, order, a
3	the, order, accordance
4	order, the, front
5	18, 3, 2
6	5, 3, 2
7	3, 5, 2
8	5, 6, 4
9	6, 5, 7
10	6, 7, 8
11	6, 7, 1

Table 8: Logit Lens- "8 9 10 11"

MLP	Top-3 Tokens
0	th, 11, 11
1	th, 11, 45
2	th, 30, 45
3	30, 45, 34
4	45, 34, th
5	votes, ., 9
6	9, ., 11
7	11, 9, 1
8	11, 12, 111
9	12, 11, 12
10	12, 13, 12
11	12, 13, \n

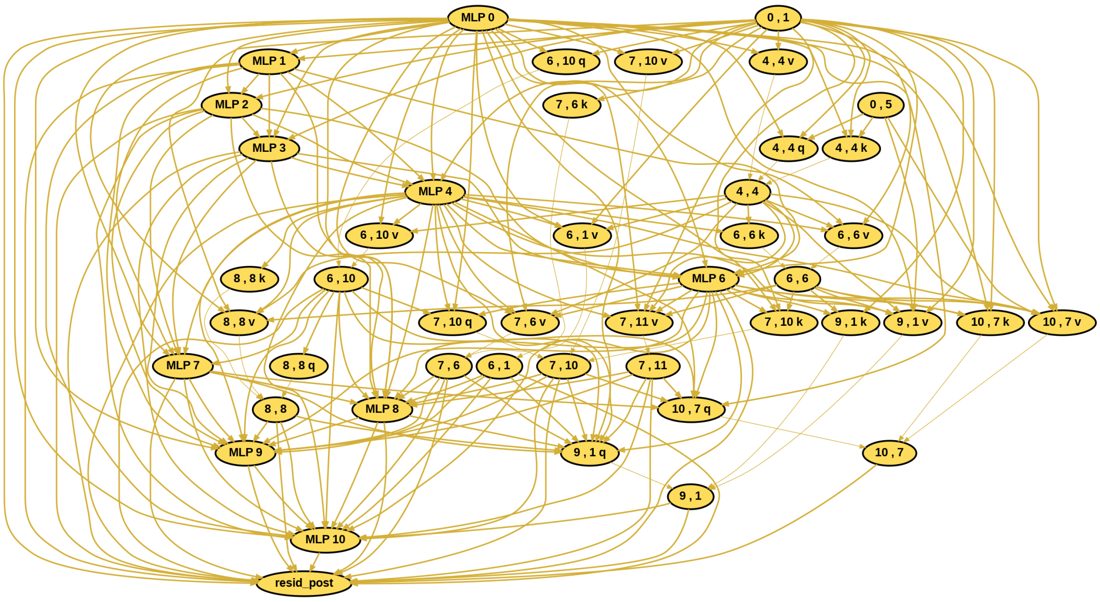


Figure 8: Months Circuit with Attention Head (QKV) Decomposition.

Table 9: Logit Lens- "seven eight nine ten"

MLP	Top-3 Tokens
0	thousand, ten, years
1	thousand, fold, hundred
2	thousand, percent, minutes
3	thousand, percent, years
4	thousand, percent, million
5	thousand, ths, million
6	thousand, million, years
7	thousand, ths, 9
8	nine, 11, 9
9	eleven, 11, twelve
10	eleven, 11, twelve
11	eleven, twelve, 11

Table 10: Logit Lens- "May June July August"

MLP	Top-3 Tokens
0	Aug, August, 2017
1	2017, 2014, Aug
2	2014, 2017, 2015
3	2017, 2014, 2015
4	2014, 2017, 2018
5	2014, 2013, 2018
6	September, December, August
7	December, September, August
8	August, September, October
9	September, August, October
10	September, August, October
11	September, August, October

F Functionality Method Details

Component Output Scores Details. To continue from Section §3, we employ the heads' output projection (**OV**) matrices to examine the attention head outputs written to the residual stream by the OV circuit. For example, we can check if a head is copying tokens, a behavior introduced as *copy scores* by (Wang et al., 2022). Copy scores measure how well a head reproduces a token from the input. We modify this method to obtain the *component output score*, which follows a similar principle but measures how many prompts have a *keyword* token are in the output. For instance, a keyword may be the integer $I+1$, given integer I as the last token in a sequence. To calculate these scores, we multiply the state of the residual stream after the first MLP layer at the last token with the OV matrix of the attention head of interest. This result is unembedded and layer normalized to get logits. If the keyword is in the top-5 of these logits, +1 is added to the score. Finally, we divide the total score by the total number of keywords across all prompts to obtain a percentage. In this paper, we use all sequence members of the prompt as keywords.

G Attention Pattern Extended Results

Sequence Member Detection Heads Details. We discovered a "similar member" detection head, 1.5, and a "sequence member detection", 4.4, both shown in Figure 9, where numerals attend to previous numerals, and in Figure 10, where number words attend to previous number words. Furthermore, in Figure 11, we use prompts consisting of names, same tokens ("is") and periods in the format of "<name> is <number>" (such as "Adam is 1.") to discern whether these heads are "similarity detection" heads in general, or are more specific to detecting sequence members such as numbers. This analysis shows that not all token types attend to their similar types; for instance, names do not attend to names. We also do not observe every token attending to a previous position k tokens back (where k is an integer), so we do not conclude that these heads also act as previous token heads. Additionally, Figure 12 shows that when both Numerals and Months are in sequence order, the heads attend to both Numerals and Months.

H Months Circuit Keeping MLP 11

Although the iterative node pruning algorithm removes MLP 11 for the Months task, we note

Table 11: Performance Drop when Head is Removed from Months Circuit with MLP 11.

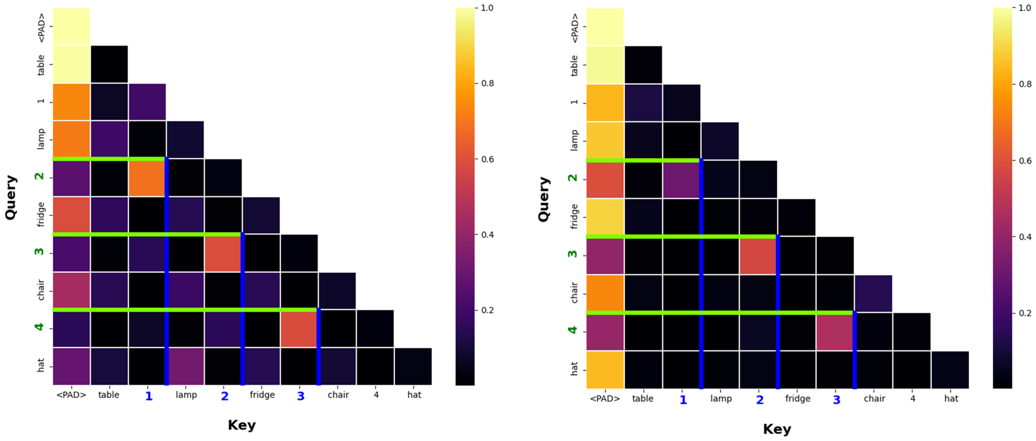
Important Head	Months
(4, 4)	-78.27
(7, 11)	-68.61
(0, 1)	-61.09
(9, 1)	-51.60
(6, 10)	-33.00
(8, 8)	-12.47
(6, 6)	-11.87
(4, 10)	-6.95
(7, 10)	-4.74
(6, 1)	-4.52
(7, 2)	-2.06

this is only because the performance drop of -19.50%, shown in Table 6, is barely within threshold $T = 20\%$. Thus, we also run experiments to iteratively find a Months circuit that keeps MLP 11, which is shown with the other two tasks' circuits in Figure 13. Table 11 shows the importance of each head in the circuit. Overall, the results are largely similar to the Months circuit shown in Figure 2.

I Circuit Entanglement and Editing Definitions

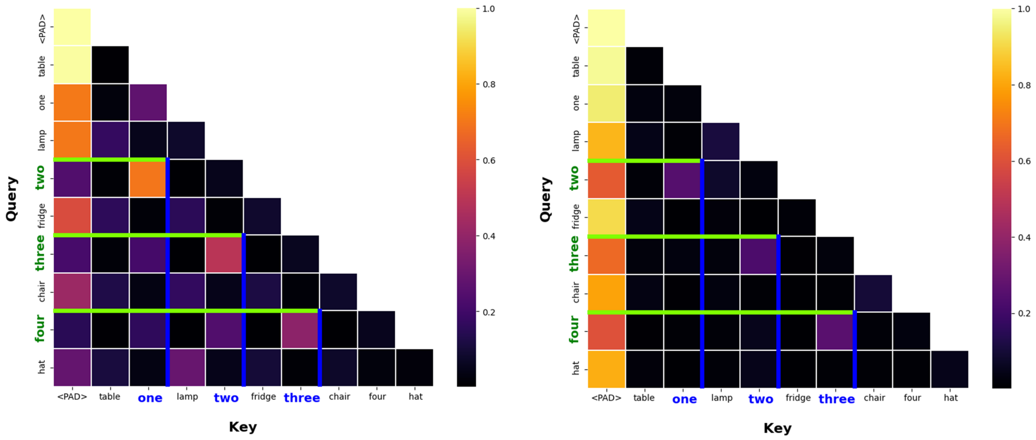
Definitions. A *circuit* can be defined as "a human-comprehensible subgraph, which is dedicated to performing certain task(s), of a neural network model" (Räuker et al., 2023). To describe the circuits representations in this paper, we define a *circuit graph* as a connected graph C with (1) a node set N of components, and (2) an edge set E, in which an edge (n_1, n_2) represents how component n_1 affects of component n_2. [8] We define that a circuit graph C is *used* for a task T based on how ablating all model components aside from those in the circuit still allows the model to have a certain level of performance; we determine this level as described in iterative pruning in §3.1. We note that a task T can be broken into a set of sub-tasks $S_T = \{S_1, ..., S_n\}$; for instance, one sub-task of

[8] Different studies may define "circuit" in different ways.



(a) (b)

Figure 9: Attention Patterns for Increasing Digits of (a) Head 1.5 and (b) Head 4.4. Lighter colors mean higher attention values. For each of these detection patterns, the query is shown in green, and the key is shown in blue. We observe that digits attend to digits.

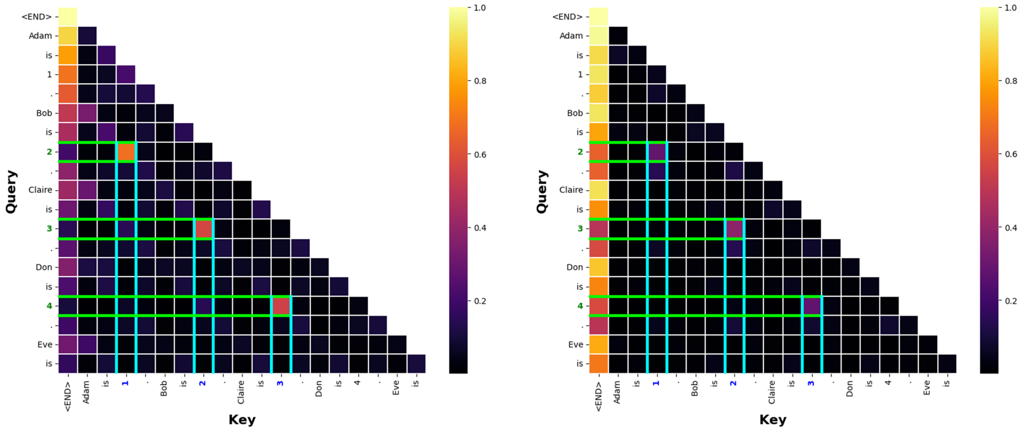


(a) (b)

Figure 10: Attention Patterns for Increasing Number Words of (a) Head 1.5 and (b) Head 4.4. We observe that number words attend to number words. We also observe that the attention scores here are less than they are for digits, suggesting that head 4.4 is more important for digit detection, which is consistent with its importance for the digits task over the number words task as shown in Table 2.

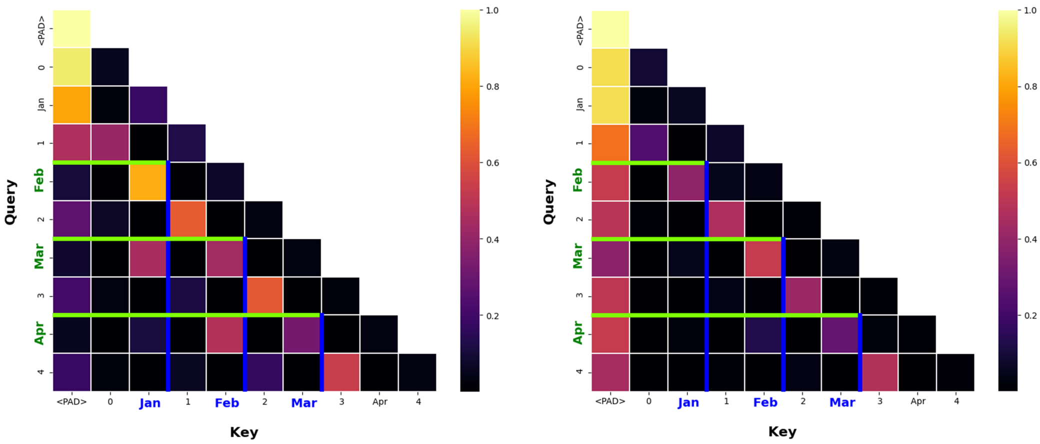
IOI is to inhibit repeated subjects. Next, we define a circuit graph C_1 to be a *circuit subset* of circuit graph C_2 if all the nodes in the node set of C_1 are contained in the set of nodes for C_2. We also define C_1 to be a *sub-circuit* of C_2 all the edges in the edge set of C_1 are also contained in the edge set of C_2. We further define circuit graph C_1 used for task T_1 to be a *functional sub-circuit (or subset)* of circuit graph C_2 used for task T_2 if these conditions are met: (1) C_1 is a sub-circuit of C_2, and (2) T_1 is a subtask in S_{T_2}, the set of subtasks of T_2.

Circuit Entanglement. Given that components play multiple roles (Merullo et al., 2023), editing components in a circuit used for task A can have an effect on task B. Instead of just vaguely assuming this would have "some effect", such as ruining task B in "some way", our aim is to precisely describe, and thus approximately predict, what this effect is. We define two circuits C_1 and C_2 as being *analogously entangled* if editing the functionality of C_1 affects the functionality of C_2 in an analogous way. For instance, let C_1 be for "digits continuation" and let C_2 be for "months continuation". If component H in C_1 finds the "next digit of a sequence" and we edit it to now find the "previous digit", then if task B now finds the "previous month", we say C_1 and C_2 are *analogously entangled*.



(a) (b)

Figure 11: Attention Patterns for (a) Head 1.5 and (b) Head 4.4. We observe that digits attend to digits, but they are not considered general "similarity detection heads" as non-number token types do not attend to their similar or same token types.



(a) (b)

Figure 12: Attention Patterns for (a) Head 1.5 and (b) Head 4.4. We observe that digits attend to digits, and that months attend to months. In general, they appear to be adjacent sequence member detection heads.

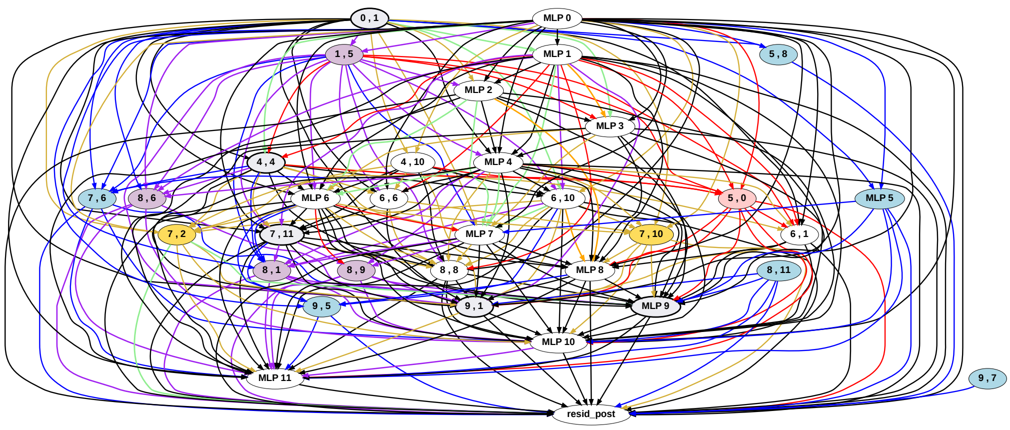


Figure 13: Showing all three circuits and their overlap, but using a Months circuit that keeps MLP 11.