Investigating the Indirect Object Identification circuit in Mamba

Danielle Ensign [*1] **Adrià Garriga-Alonso** [*2]

Abstract

How well will current interpretability techniques generalize to future models? A relevant case study is Mamba, a recent recurrent architecture with scaling comparable to Transformers. We adapt pre-Mamba techniques to Mamba and partially reverse-engineer the circuit responsible for the Indirect Object Identification (IOI) task. Our techniques provide evidence that 1) Layer 39 is a key bottleneck, 2) Convolutions in layer 39 shift names one position forward, and 3) The name entities are stored linearly in Layer 39's SSM. Finally, we adapt an automatic circuit discovery tool, positional Edge Attribution Patching, to identify a Mamba IOI circuit. Our contributions provide initial evidence that circuit-based mechanistic interpretability tools work well for the Mamba architecture.

1. Introduction

If we care about using interpretability on new models, we should know: Will interpretability techniques generalize to new architectures?

To investigate this question, we apply existing mechanistic interpretability techniques to a new model developed after most interpretability techniques: Mamba. Mamba is a State Space Model (SSM), a type of recurrent neural network (Gu & Dao, 2023). Mamba is the result of years of work on language modeling with state space models (Gu et al., 2020; 2022; Fu et al., 2023) and has scaling competitive with Transformers (Gu & Dao, 2023; Lieber et al., 2024). Because it is a recurrent network, it only needs to store hidden states from the previous token, resulting in faster inference. The recurrence is also linear (and thus associative) over token position, which permits further optimizations. See Appendix for architecture details.

While we are the first to focus on finding circuits in Mamba, previous work has shown other interpretability techniques apply. For example, Sharma et al. (2024) locate and edit factual information with ROME (Rank One Model Editing). Ali et al. (2024) extract hidden attention matrices, and Torres (2024); Grazzi et al. (2024) use linear probes to identify capabilities. Additionally, Paulo et al. (2024) showed that Contrastive Activation Addition (Rimsky et al., 2024), Tuned Lens (Belrose et al., 2023) and probes to elicit latent knowledge (Mallen et al., 2024) transfer to the Mamba architecture.

This work focuses on applying techniques from circuit-based mechanistic interpretability to Mamba to see how well these techniques transfer to new architectures. In particular, we study state-spaces/mamba-370m, a 370-million-parameter Mamba model pretrained (Gu & Dao, 2023) on The Pile (Gao et al., 2020). We chose this model as it is the smallest Mamba model with good performance ($\sim 96\%$ accuracy) on the Indirect Object Identification (IOI) task (Wang et al., 2023).

In particular, for the IOI task, we:

1. Show multiple lines of evidence suggesting layer 39 is a bottleneck:
   (a) Zero and Resample ablation (Section 3.1.4) experiments point to layer 39 and layer 0.
   (b) We compute greedy minimal subsets of layers allowed to transfer information across tokens ("token cross-talk"). These always include layer 39 (but layer 0 only 18% of the time).
2. Provide evidence that the convolution on layer 39 shifts name data to the next token position.
3. Modify the representations used by layer 39 using averages of activations, resulting in overwriting one output name with another (Rimsky et al., 2024). These results suggest that in the SSM of layer 39, entity names are linearly represented, with different representations for the first and second time (or sentence) the names appear in IOI.

In addition, we show that Edge Attribution Patching (Syed et al., 2023) works well on Mamba and provide the resulting computational graphs.

[*] Equal contribution [1] Independent [2] FAR AI.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

2. The Test Subject: Mamba

Here we provide a brief overview of the Mamba architecture. We refer the reader to Ensign et al. (2024) for a more detailed introduction. We use $\overset{[A,B]}{v}$ to denote that variable v has shape $[A, B]$.

2.1. State Space Model (SSM)

Mamba's SSM block can be written as mapping a 1D space to a N-dimensional state space, then back to a 1D space:

$$\overset{[N]}{h_t} = \overset{[N,N]}{\overline{A}}\ \overset{[N]}{h_{t-1}} + \overset{[N,1]}{\overline{B}}\ \overset{[1]}{x_t} \tag{1}$$

$$\overset{[1]}{y_t} = \overset{[1,N]}{C}\ \overset{[N]}{h_t} + \overset{[1,1]}{D}\ \overset{[1]}{x}_t \tag{2}$$

In Mamba A is diagonal, so we will just write it as $\overset{[N]}{\overline{A}}$ and do an element-wise product "$\odot$".

Each layer does E of these in parallel. A has a separate value for each e, and is encoded as an $[E, N]$ matrix. We can denote $\overline{A_e}$ as the N-sized entry for stream e, giving us,

$$\overset{[N]}{h_{t,e}} = \overset{[N]}{\overline{A_e}} \odot \overset{[N]}{h_{t-1,e}} + \overset{[N,1]}{\overline{B}}\ \overset{[1]}{x_{t,e}} \tag{3}$$

$$\overset{[1]}{y_{t,e}} = \overset{[1,N]}{C}\ \overset{[N]}{h_{t,e}} + \overset{[1,1]}{D}\ \overset{[1]}{x}_{t,e} \tag{4}$$

Finally, $\overline{A}_e$, $\overline{B}$, and C depend on the SSM input, and so gain a subscript t. $\overline{B}$ also gains a subscript e through the variable time-step, $\overset{[1]}{\Delta}_{t,e}$. The final SSM expressions are:

$$\overset{[N]}{h_{t,e}} = \overset{[N]}{\overline{A_{t,e}}} \odot \overset{[N]}{h_{t-1,e}} + \overset{[N,1]}{\overline{B_{t,e}}} \overset{[1]}{x_{t,e}} \tag{5}$$

$$\overset{[1]}{y_{t,e}} = \overset{[1,N]}{C_t}\ \overset{[N]}{h_{t,e}} + \overset{[1,1]}{D}\ \overset{[1]}{x}_{t,e}, \tag{6}$$

where

$$\overset{[N]}{\overline{A_{t,e}}} = \exp(-\overset{[1]}{\Delta}_{t,e} \exp(A_{\text{log}})_e), \tag{7}$$

$$\overset{[N]}{\overline{B_{t,e}}} = \overset{[1]}{\Delta}_{t,e} \overset{[N]}{B_t}, \qquad \text{with } \overset{[N]}{,}\ B_t = \overset{[N,E]}{W_B}\ \overset{[E]}{x_t}, \tag{8}$$

$$\overset{[N]}{C_t} = \overset{[N,E]}{W_C}\ \overset{[E]}{x_t}, \tag{9}$$

$$\overset{[1]}{\Delta}_{t,e} = \text{softplus}(\overset{[E]}{x_t} \cdot \overset{[E]}{W_e^{\Delta}} + \overset{[1]}{B_e^{\Delta}}), \tag{10}$$

with $\overset{[E,E]}{W^{\Delta}}, \overset{[E]}{B^{\Delta}}, \overset{[N,E]}{W_B}, \overset{[N,E]}{W_C}, \overset{[N,E]}{A_{\text{log}}}$ being learned parameters, and $\text{softplus}(x) = \log(1 + e^x)$. This parameterization guarantees that $\overline{A} < 1$, and thus the hidden state does not explode.

2.2. Architecture

Mamba has multiple layers which each add to a residual stream. Each layer does:

- Project input $\overset{[B,L,D]}{\text{resid}}$ to $\overset{[B,L,E]}{x}$
- Project input $\overset{[B,L,D]}{\text{resid}}$ to $\overset{[B,L,E]}{\text{skip}}$
- Conv over the time dimension, with a different filter for each $e \in [E]$ ($x = \text{conv}(x)$)
- Apply non-linearity (silu) ($x = \text{silu}(x)$)
- $y = SSM(x)$
- Gating: $y = y * \text{silu}(\text{skip})$
- Project $\overset{[B,L,E]}{y}$ to $\overset{[B,L,D]}{\text{output}}$

Where B is batch size, L is context length, D is embedding dimension, and $\text{silu}(x) = x * \text{sigmoid}(x)$. See Figure 1.

3. Circuit-based Mechanistic Interpretability

To understand how large language models (LLMs) implement their emergent capabilities (Wei et al., 2022), we focus on finding human-interpretable algorithms (Olah, 2022). This involves representing models as computational graphs and identifying circuits that are subsets of that computational graph. Ideally, each subgraph would also be annotated to describe the role of each component (Geiger et al., 2021).

Finding circuits that capture the behavior on all inputs is intractable for large language models. Therefore, we study behavior on specific tasks.

3.1. Problem Description

Following (Geiger et al., 2021; Conmy et al., 2023): we have a behavior (task) that we would like to study, a metric for evaluating performance, and a coarse-grained computational graph of the neural network on which we express explanations. We would like to find the minimal subgraph that attains a high enough metric score (where target metric score is a hyperparameter), with an explanation of what variations in the data each graph component captures.

3.1.1. IOI Task

We are studying the IOI task, initially examined by (Wang et al., 2023). Consider an example data point:

```
Friends Isaac, Lucas and Lauren went to
the office. Lauren and Isaac gave a
necklace to
```

The model is asked to predict the next token, and the correct answer is `" Lucas"`. `" Lucas"` is the Indirect Object we are trying to identify. See the Appendix for detailed data-generation templates and corruption information.

3.1.2. Metric

There are many choices of metrics: KL-Divergence, Logit Diff, Accuracy, Probability of the correct answer, etc. The best metric to use in general is an open question and may be task specific. For IOI, Zhang & Nanda (2024) suggest the Normalized Logit Diff metric, as that helps propagate information missed by accuracy. See the Appendix for further details.

3.1.3. Computational Graph

We use the MambaLens library (see also Nanda & Bloom (2022)) to intervene at different locations per experiment.

- Section 4.1.1 “Resample Ablation” uses `blocks.{layer}.hook_layer_input`.
- Section 4.1.2 “Layer Removal” uses `blocks.{layer}.hook_out_proj`.
- Section 4.1.3 “Removing Cross Talk” uses `blocks.{layer}.hook_in_proj`.
- Section 4.2 “Layer 39 Uses Conv to Shift Names One Position Forward” uses `blocks.{layer}.hook_in_proj` and `blocks.{layer}.hook_conv`.
- Section 4.3 “Controlling model output by modifying representations on layer 39” uses `blocks.{layer}.hook_ssm_input`. For the cosine sim plots, it uses further hooks inside the ssm, described in Appendix.
- Both Section D “EAP” runs use `blocks.{layer}.hook_layer_input` and `blocks.{layer}.hook_layer_output`. They also use `hook_embed` (described the next section) as the input node, and `blocks.47.hook_resid_post` as the output node.
- The “ACDC” (Section D.2) run following the (non positional) EAP run uses all the hooks from the EAP runs. It also uses `blocks.{layer}.hook_skip`, `blocks.{layer}.hook_conv`, and `blocks.{layer}.hook_ssm_input`.

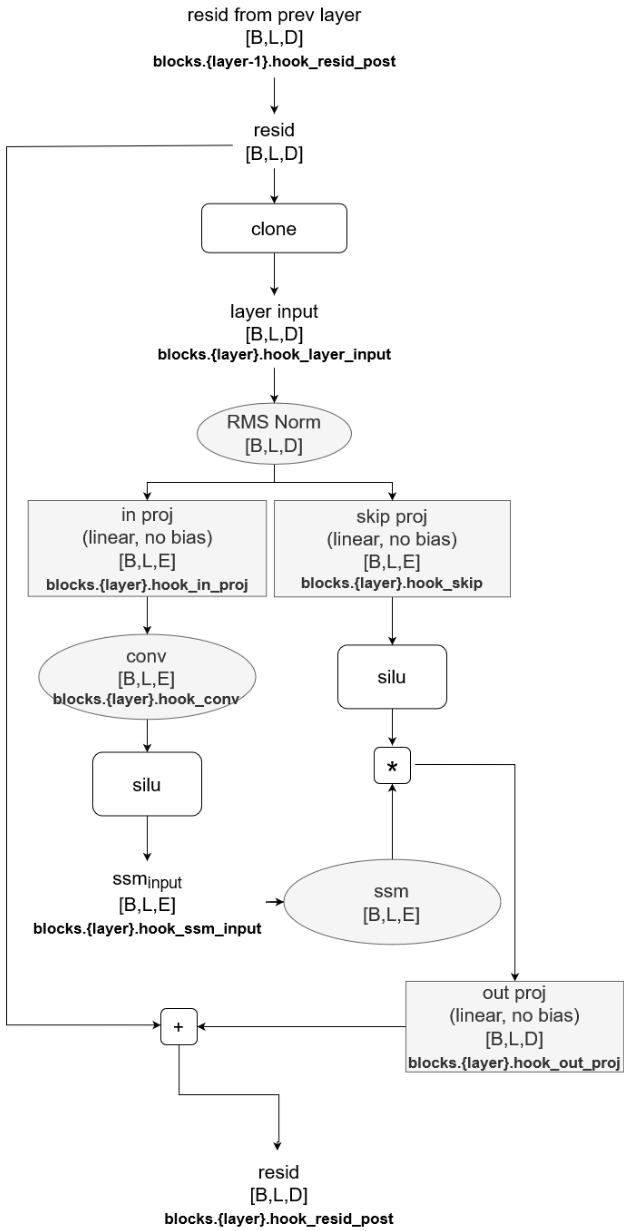


Figure 1. A single layer in the Mamba architecture, with hook points listed in all the locations we intervene. Note that the SSM contains further hook points, described in Section 4.3, “Controlling model output”. The “SSM” and “conv” components are affected by previous time steps.

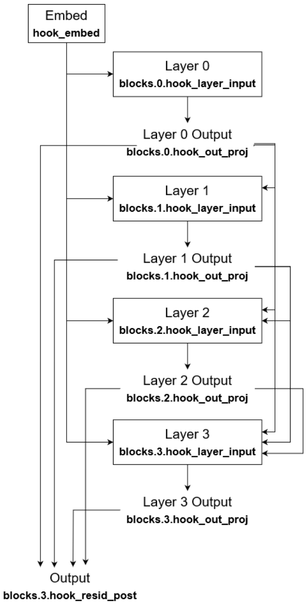


Figure 2. Fully connected causal graph, using the additivity of the residual stream. This is an example network with 4 layers, so the output node is `blocks.3.hook_resid_post`. The full network we study has 48 layers, so the output node is `blocks.47.hook_resid_post`

Note that `blocks.{layer}.hook_layer_input` is the residual stream before normalization. If we patched directly on these, it would modify downstream values as well. Thus, to patch only a single layer's input, we clone this value first.

3.1.4. Ablations

To identify which nodes and edges are important, we take inspiration from causal inference (Pearl, 2009): ablate nodes of our computational graph and observe changes in the output.

Replacing activations with zero (Olsson et al., 2022; Cammarata et al., 2021) or the mean over many data points (Wang et al., 2023) was initially used. However, these can result in activations that are out of distribution (Chan et al., 2022). *Resample ablation* (Geiger et al., 2021), also known as *interchange interventions* and *causal tracing*, is a commonly used alternative(Hanna et al., 2023; Heimersheim & Janiak, 2023; Wang et al., 2023; Conmy et al., 2023). Resample ablation begins by running a forward pass with a *corrupted* prompt, then substitutes those corrupted activations into a forward pass run on the uncorrupted prompt.

In addition to resample ablation, in Mamba (and Transformers), the residual stream is a sum of outputs from every layer. This allows us to create an edge between every layer (Elhage et al., 2021), see Figure 2.

To patch an edge this causal graph (Elhage et al., 2021) going from layer i to layer j, we can do:

$$\overset{[B,L,D]}{\text{patched input}_j} = \overset{[B,L,D]}{\text{input}_j} - \overset{[B,L,D]}{\text{output}_i} + \overset{[B,L,D]}{\text{corrupted output}_i} \tag{11}$$

We give the dimensions of our tensors in square brackets [ ] above the term. Here, B is batch size, L is context length, and D is embedding dimension. The output$_i$ is `blocks.i.hook_out_proj`, computed during the same forward pass as the input$_j$ and patched input$_j$ (which both use `blocks.j.hook_layer_input`). corrupted output$_i$ are stored values from a separate forward pass using the corrupted prompt and `blocks.i.hook_out_proj`.

3.2. Semi-automatic Circuit Discovery

Initially, finding circuits had to be done by hand: patching subsets of nodes and edges (known as path patching (Goldowsky-Dill et al., 2023)) until a circuit emerges. Several methods have since been developed to automate this process. Subnetwork probing learns a mask over the graph using gradient descent (Cao et al., 2021). Automated Circuit DisCovery (ACDC) starts from sink nodes and works backwards to reconstruct the causal graph (Conmy et al., 2023). ACDC requires a separate forward pass for every edge, and this can be very time-consuming. Head Importance Score for Pruning (Michel et al., 2019), and more recently EAP (Edge Attribution Patching) (Syed et al., 2023), use the gradient to approximate the contribution of all edges simultaneously. In particular, EAP approximates the *attribution scores* of an edge between layer i and layer j via:

$$\overset{[B,L,D]}{\text{attr}_{i\mapsto j}} = (-\overset{[B,L,D]}{\text{output}_i} + \overset{[B,L,D]}{\text{corrupted output}_i})\nabla\overset{[B,L,D]}{\text{input}_j} \tag{12}$$

Where ∇input$_j$ is the gradient given from the backward hook made in these steps:

1. For every layer, create a backward hook on `blocks.j.hook_layer_input`
2. Run a `forward` pass that patches every edge
3. Compute the metric on the resulting logits, and call `backward` on the metric's value.

To get an attribution for each edge, we `sum` attr$_{i\mapsto j}$ over the L and D axes, then `mean` over the B axis.

This approximation can be improved by using integrated gradients (Marks et al., 2024; Sundararajan et al., 2017):

compute a separate $\overset{[B,L,D]}{\text{attr}_{i\mapsto j}}(\overset{[1]}{\alpha_k})$ for an $\overset{[1]}{\alpha_k} = k/(\text{ITERS}-1)$ where $k \in [0, \ldots, \text{ITERS}-1]$, then compute the average of all these scores (ITERS is an int hyperparameter that determines how fine grained our approximation is, usually 5-10 is large enough). The attribution is computed in using Equation 12 like before, however, the forward pass for a given α_k only ”partially” applies every patch as follows:

$$\overset{[B,L,D]}{\text{patched input}_j} = \overset{[B,L,D]}{\text{input}_j} - \overset{[1]}{\alpha_k}(\overset{[B,L,D]}{\text{output}_i} + \overset{[B,L,D]}{\text{corrupted output}_i}) \tag{13}$$

Once we have these attribution scores, we can sort all edges by their attribution and perform a binary search to find the minimal set of edges that achieves our desired metric.

4. Findings

4.1. Layer 39 is Important

We have three lines of evidence suggesting layer 39 is important. While two of these lines of evidence also suggest layer 0 is important, we also provide evidence that token cross-talk in layer 0 is not usually needed.

4.1.1. Resample Ablation

To determine which layers are important, we will resample ablate `blocks.{layer}.hook_layer_input`. To determine which tokens matter, we do this patch separately for each (layer, token_position) pair. This forces us to limit to the three templates that share name token positions.

Because each corruption affects different positions, averaging over them does not make sense. Thus, we show results separately for each corruption.

Figure 3 shows that normalized logit diff changes most when patching layer 0 and 39.

4.1.2. Layer Removal

Each layer adds to the residual stream. This allows us to ”remove” a layer by setting this added value to zero, i.e., zero-ablating layer outputs (`blocks.{layer}.hook_proj_out`). We plot probability of the correct answer, as there is no corrupted answer to compare to.

In Figure 4, we again see that, layers 0 and 39 are crucial parts of the circuit that cannot be removed.

We also find that by repeatedly removing the layer that decreases accuracy the least, about half of the layers can be removed with minimal impact on accuracy. We replicated this layer removal robustness on GPT-2-Small (Radford et al., 2019). This might be seen as evidence for the residual stream having a privileged basis that is consistent between layers. However, it is also consistent with there being multiple distinct spaces (for example, embed-0, 0-39, 39-out), or layers being simultaneously compatible with multiple different spaces. See Belrose et al. (2023) for more discussion on this “privileged basis” perspective.

4.1.3. Removing Token Cross-Talk

There are two ways in which a layer at a particular token position can affect future positions (“token cross-talk”): the convolutional (conv) layer, and the SSM block. Putting corrupted data into the conv will also put corrupted data into the SSM, as it is downstream of the conv. Thus, to remove a specific layer’s ability to have token cross-talk, we can apply resample ablation to that layer’s conv inputs (`blocks.{layer}.hook_in_proj`) at all positions. Because we also patched convs in previous positions, the SSM will only have information about the corrupted input.

If we patch every layer before L in the manner above, this removes any information about previous tokens at layer L. However, if we only patch some previous layers, the previous tokens can have influence: a previous layer could move two tokens into the same position, and then a later layer could process those token interactions in place.

Also, note that this does not completely remove “cross talk”, it only removes cross talk that is specific to the uncorrupted prompt (i.e., cross talk that is needed for outputting the correct answer). Cross talk that occurs in both uncorrupted and corrupted prompts will still occur. This is somewhat acceptable because we only care about task-relevant cross talk.

Given these two disclaimers, we still feel this is a useful proxy for ”removing cross talk”.

Now, start with patching all layers’ cross talk, then “unpatch” the layer that improves accuracy the most. This is repeated until accuracy is about 0.9, resulting in a “minimal cross talk circuit” that can perform the task. We do this separately for each (corruption, template) pair.

In Figure 5, we see that Layers 39 and 15 appear in every minimal circuit found. Layer 15 seems worthy of investigation in future work, as these two also stand out in EAP. Inspecting the logs, Layer 39 is always the first layer added and has a large effect.

In 82% of these minimal circuits, Layer 0 did not appear. This is strong evidence that for the majority of (corruption, template) pairs, computation Layer 0 does is in-place and not cross talk.

In Transformers, it is suspected that layer 0 is responsible

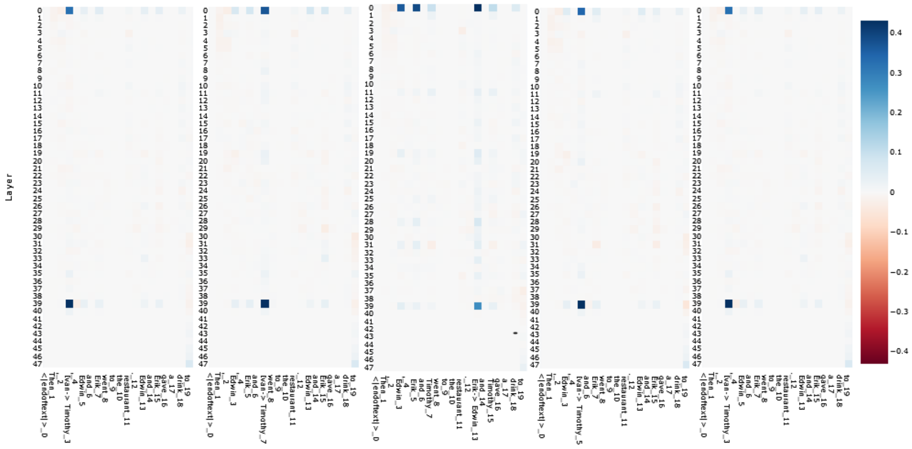


Figure 3. Displayed is 1 - (Normalized logit diff) for each (layer, position) patch, averaged over 80 data points. 0 corresponds to acting like the uncorrupted forward pass, and 1 corresponds to acting like the corrupted forward pass. The y-axis is Layer, and the x-axis is token position. The corruptions can be observed by inspecting the token position labels. Each of the five plots correspond to different IOI templates.

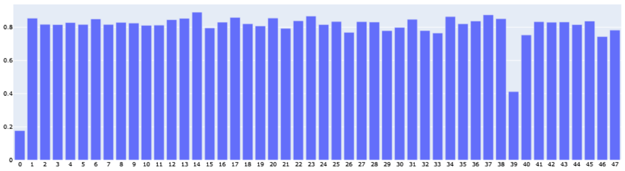


Figure 4. Correct token probability when zero-ablating each layer's outputs. The probabilities shown here are relative probabilities (take the logits of the 3 names in the prompt and the corruption's name, and softmax over only those). For comparison, no interventions give 83%

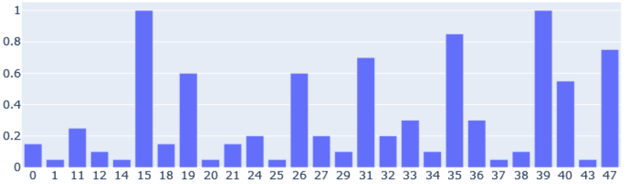


Figure 5. Out of all (corruption, template) pairs, the proportion of times a given layer was in the minimal cross talk circuit.

for multi-token embeddings (Nanda et al., 2023). These results suggest something else happens in Mamba. However, because all of our prompts use single token names, it is possible that these capabilities are simply not needed for this task (but still exist).

4.2. Layer 39 Uses Conv to Shift Names One Position Forward

When examining the hidden state, we can display the cosine similarity of a token's contribution to the current state with future (and previous) hidden states. This allows us to see how much the value was "kept around" (see Appendix for more information on the hooks used here).

As this is not causal, it should not be relied on too heavily. The structure seems to be name-dependent; we show three representative examples in Figure 6.

What stands out is that the horizontal lines are one token after each name. This could either mean that 1) A previous layer shifted the tokens over, or 2) Layer 39 shifted the tokens over using the conv.

To distinguish between these, we can do resample ablation on the individual conv "slices": The conv can be seen as four E-sized "slice" vectors for each (-3,-2,-1,0) relative token position, that are multiplied (element-wise) by the corresponding E-sized token representations.

- If hypothesis 1 were true, we should see the 0 conv slice at token position + 1 have a large value.
- If hypothesis 2 were true, we should see the -1 conv slice at token position + 1 have a large value.

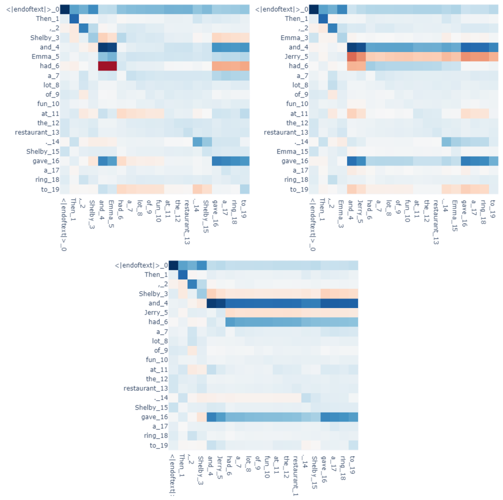


Figure 6. Cosine Similarity between the current token's contribution to h (which is $\bar{B}_i^{[B,E,N]} x_i^{[B,E,1]}$, each i is on the x axis), and the hidden state ($h_j^{[B,E,N]}$ each j is on the y axis)

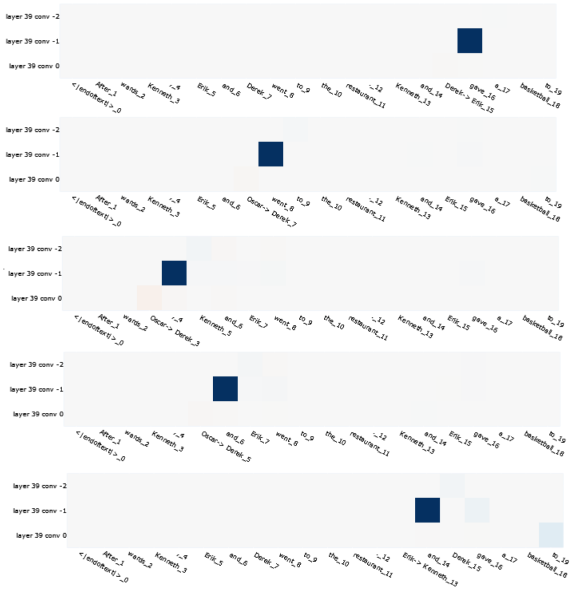


Figure 7. This is 1 - Normalized Logit Diff when patching on the given conv slice. 0 corresponds to acting like uncorrupted, 1 corresponds to acting like corrupted. The x-axis is conv slices (-2, -1, then 0) for layer 39. The y-axis is token position; observe the labels to see which corruption was used.

Figure 7 supports hypothesis 2, that Layer 39 uses conv to shift names one position forward.

We did some tests to investigate multi-token names and found the cosine similarity plots always have lines at the position after the first token of the name (possibly other layers handle multi-token names). It is also worth nothing that we see the horizontal lines for entities, not just names.

We do not yet know why this shifting behavior occurs, and leave that question for future work.

4.3. Controlling Model Output by Modifying Representations on Layer 39

We hypothesize that the representations in the SSM are linear because, on a single layer, the mechanism it has to add or remove information from tokens is linear in h.

We tried to visualize $\Delta_{t,e}^{[1]}$ adding or removing information to various parts but did not find it very insightful. Instead, to investigate whether the internal representation of Layer 39 SSM is linear, we do the following:

1. Create a large IOI dataset. For each data point, store the activations of each name's `blocks.39.hook_ssm_input`. We use the activation at the token position one after the name, because of the shifting behavior we observed earlier.
2. For each name, average the representations. Store a separate average for, say, ”John” in the first position, ”John” in the second position, etc. We use enough data points that each (name, position) pair gets 50-100 values to average over.
3. *Replace Method*: To write a different name, simply substitute the SSM input at that position with the averaged value from a different name.
4. *Subtract and Add Method* Instead of substituting, subtract the current name's average and add the substituted name's average.

We find that the Replace Method works adequately, while the Subtract and Add Method works surprisingly well, changing the logits to the desired output more than 95% of the time.

One thing to note: It was possible that the SSM was using the representations from the name's token position, as well as the name's token position + 1. Patching on conv slices was initial evidence this did not occur, and the efficacy of this replacement procedure provides further evidence that this is not the case.

Having a separate average for each position also lets us test if token position is an important part of the representation.

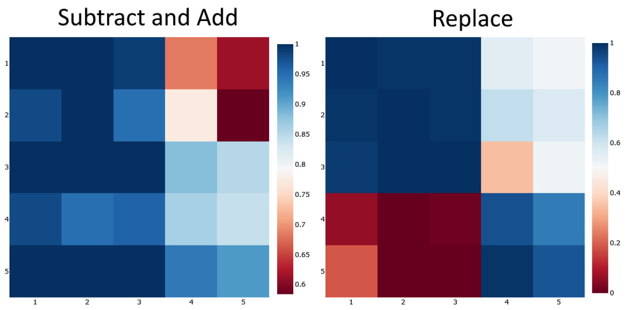


Figure 8. Proportion of data where logit of the corrupted name is higher than the logit of original name, using the two methods described in Section 4.3. The x-axis is the position the average was computed from, the y-axis is the position being substituted. To substitute into the fourth and fifth positions, we substitute the correct answer (instead of a patched name).

If it is, we should expect that ”John” at name position 2 should not be easily substituted for ”Mary” at name position 0.

Instead, in Figure 8 we find that the first three name positions are compatible, while the fourth and fifth positions are much less compatible.

4.3.1. Compatibility of first three name positions

The compatibility of the first three name’s representations could either suggest:

- The IOI circuit does not store positional information in the first three names, or
- There are circuits to handle incorrectly encoded positional information, which got activated and handled our patching well despite having incorrect positional data

Distinguishing between these is left for future work.

4.3.2. Incompatibility of fourth and fifth name positions

Consider one of our data points: “Friends Isaac, Lucas and Lauren went to the office. Isaac and Lucas gave a necklace to” (answer is “ Lauren”)

We see that when names occur in the fourth or fifth position, it is the second time they occur in the prompt.

One hypothesis is that the conv sees a period and encodes that in the name representation. However, while the model we study (mamba-370m) has four conv slices, we find that the conv slice attending to the -3 position is always zero, likely due to a bug in the Mamba training code. Thus, it can only attend to the previous 2 positions in practice (the third conv slice is for attending to the current position). This means that the name in the fifth position’s representation is not distinguishable from the name in the third position by the conv.

Thus, some token cross-talk must be happening in a layer before 39. As mentioned above, for 82% of (corruption, template) pairs, cross-talk in layer 0 is not needed. So while these experiments provide strong evidence that layer 39 is a bottleneck, more circuit analysis is needed.

5. Positional Edge Attribution Patching (Positional EAP)

Here we describe a simple modification to EAP that allows us to have token-level edge attributions.

Typically, in EAP, after we compute $\text{attr}_{i\mapsto j}^{[B,L,D]}$ we `sum` over the L and D dimensions, then take the `mean` over the B dimension to get an attribution for each edge.

Instead, we will just `sum` over the D dimension and `mean` over the B dimension, giving us an attribution for every (edge, position). See Appendix D.

The results of EAP further emphasize the importance of Layer 39. However, there is also significant activity elsewhere that merit further analysis.

6. Future Work

There are still many open questions we have about the IOI circuit in mamba-370m. Future work can focus on:

- Examining PCA of the average name representations
- Analysis of what other layers are doing (before and after 39)
- Training Sparse Autoencoders (SAEs) and using EAP to make a feature circuit capable of doing the task, to get a more fine grained analysis (similar to work in Marks et al. (2024))
- Conducting similar analysis on other tasks (such as docstring (Heimersheim & Janiak, 2023) or greater than (Hanna et al., 2023))

7. Reproducibility

We will open source our code upon publication. All experiments were conducted on a RTX A6000.

8. Credits

The authors would like to thank the ML Alignment & Theory Scholars (MATS) program for providing a workspace to conduct this research, FAR AI Labs for compute, and LTFF for funding. We would also like to thank Niels uit de Bos, Iván Arcuschin Moreno, Rohan Gupta, Thomas Kwa, Scott Neville, Gonçalo Paulo and Joseph Bloom for the helpful conversations.

References

Ali, A., Zimerman, I., and Wolf, L. The hidden attention of mamba models, 2024.

Belrose, N., Furman, Z., Smith, L., Halawi, D., Ostrovsky, I., McKinney, L., Biderman, S., and Steinhardt, J. Eliciting latent predictions from transformers with the tuned lens, 2023.

Cammarata, N., Goh, G., Carter, S., Voss, C., Schubert, L., and Olah, C. Curve circuits. *Distill*, 2021. doi: 10.23915/distill.00024.006. https://distill.pub/2020/circuits/curve-circuits.

Cao, S., Sanh, V., and Rush, A. Low-complexity probing via finding subnetworks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 960–966, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.74. URL https://aclanthology.org/2021.naacl-main.74.

Chan, L., Garriga-Alonso, A., Goldowsky-Dill, N., Greenblatt, R., Nitishinskaya, J., Radhakrishnan, A., Shlegeris, B., and Thomas, N. Causal scrubbing: A method for rigorously testing interpretability hypotheses. Alignment Forum, 2022. URL https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. Towards automated circuit discovery for mechanistic interpretability, 2023.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Ensign, D., Paulo, G., and Garriga-alonso, A. Ophiology (or, how the mamba architecture works), 2024. URL https://www.lesswrong.com/posts/TYLQ8gAMAmpeFcwXN/ophiology-or-how-the-mamba-architecture-works. Accessed: 2024-04-09.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models, 2023.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Geiger, A., Lu, H., Icard, T., and Potts, C. Causal abstractions of neural networks, 2021. URL https://arxiv.org/abs/2106.02997.

Goldowsky-Dill, N., MacLeod, C., Sato, L., and Arora, A. Localizing model behavior with path patching, 2023.

Grazzi, R., Siems, J., Schrodi, S., Brox, T., and Hutter, F. Is mamba capable of in-context learning?, 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. Hippo: Recurrent memory with optimal polynomial projections, 2020.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022.

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

Heimersheim, S. and Janiak, J. A circuit for Python docstrings in a 4-layer attention-only transformer, 2023. URL https://www.alignmentforum.org/posts/u6KXXmKFbXfWzoAXn/a-circuit-for-python-docstrings-in-a-4-layer-attention-only.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., Abend, O., Alon, R., Asida, T., Bergman, A., Glozman, R., Gokhman, M., Manevich, A., Ratner, N., Rozen, N., Shwartz, E., Zusman, M., and Shoham, Y. Jamba: A hybrid transformer-mamba language model, 2024.

Mallen, A., Brumley, M., Kharchenko, J., and Belrose, N. Eliciting latent knowledge from quirky language models, 2024.

Marks, S., Rager, C., Michaud, E. J., Belinkov, Y., Bau, D., and Mueller, A. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models, 2024.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 14014–14024, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html.

Nanda, N. and Bloom, J. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

Nanda, N., Rajamanoharan, S., Kramár, J., and Shah, R. Fact finding: Do early layers specialise in local processing? (post 5), 2023. URL https://www.lesswrong.com/posts/xE3Y9hhriMmL4cpsR/fact-finding-do-early-layers-specialise-in-local-processing. Accessed: 2023-12-22.

Olah, C. Mechanistic interpretability, variables, and the importance of interpretable bases. https://www.transformer-circuits.pub/2022/mech-interp-essay, 2022.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. In-context learning and induction heads, 2022. URL https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Paulo, G., Marshall, T., and Belrose, N. Does transformer interpretability transfer to rnns?, 2024.

Pearl, J. *Causality*. Cambridge University Press, 2 edition, 2009. ISBN 978-0-521-89560-6. doi: 10.1017/CBO9780511803161.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rimsky, N., Gabrieli, N., Schulz, J., Tong, M., Hubinger, E., and Turner, A. M. Steering llama 2 via contrastive activation addition, 2024.

Sharma, A. S., Atkinson, D., and Bau, D. Locating and editing factual associations in mamba, 2024.

Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks, 2017.

Syed, A., Rager, C., and Conmy, A. Attribution patching outperforms automated circuit discovery, 2023.

Torres, A. Othello Mamba: Evaluating the mamba architecture on the othellogpt experiment, 2024. URL https://github.com/alxndrTL/othello_mamba.

Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models, 2022.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods, 2024.

Appendices

A. IOI Task Details

We use the 4 prompt templates from (Conmy et al., 2023):

```
Then, [NAME], [NAME] and [NAME] went to
the [PLACE]. [NAME] and [NAME] gave a
[OBJECT] to

Afterwards [NAME], [NAME] and [NAME]
went to the [PLACE]. [NAME] and [NAME]
gave a [OBJECT] to

When [NAME], [NAME] and [NAME] arrived
at the [PLACE], [NAME] and [NAME] gave
a [OBJECT] to

Friends [NAME], [NAME] and [NAME] went
to the [PLACE]. [NAME] and [NAME] gave
a [OBJECT] to
```

In resample ablation, there are many ways to corrupt a prompt. We create a dataset choosing randomly from all possible corruptions and locations of names that:

1. Replace all instances of a single name with another name, and

2. Change the output

While we could patch two names at the same time, 1 simplifies the number of things being changed at the same time. 2 is necessary to determine if the patch had any effect.

This results in the following 5 corruptions:

```
CAB AB C
DAB AB D

ACB AB C
ADB AB D

ABC AB C
ABD AB D

ABC AB C
ABC AC B

ABC AC B
ABC BC A
```

In each of these, the top line represents the uncorrupted prompt, and the bottom line represents the corrupted prompt. Letters correspond to names: the first three are the first three names, the second two are the fourth and fifth names, and the last is the output. If two letters are the same, that means that those places share the same name. Otherwise, the names are different.

B. Normalized Logit Diff

Normalized Logit Diff is defined as:

```
min_diff =
    A_logit_corrupted - B_logit_corrupted
max_diff =
    A_logit_unpatched - B_logit_unpatched
possible_range = abs(max_diff - min_diff)
# prevent divide by zero
possible_range[possible_range == 0] = 1
logit_diff =
    A_logits_patched - B_logits_patched
normalized_logit_diff =
    (logit_diff-min_diff)/possible_range
```

where

- unpatched is a forward pass without our intervention (the baseline forward pass)

- corrupted is a forward pass without our intervention, where the prompt is modified to make B the correct answer

- patched is a forward pass that patches edges (as described above)

This results in a 1 when the model acts like the unpatched forward pass, and 0 when the model acts like the corrupted forward pass. Note that it is possible to obtain scores outside the [0,1] range.

The abs is a novel addition by us. For data points where the model is incorrect, maximizing normalized logit diff would result in the model becoming more incorrect. This abs modification fixes that issue.

C. SSM Hooks used

For the cosine similarity plots, we used `blocks.39.hook_B_bar`, `blocks.39.hook_ssm_input`, and `blocks.39.hook_h.{pos}`

See figure C for a detailed overview of the SSM nternals.

D. Automated Circuit Discovery Results

We use the following edges:

- embed ↦ layer input (hook_embed ↦ blocks.i.hook_layer_input)

- layer output ↦ later layer input (blocks.i.hook_proj_out ↦ blocks.j.hook_layer_input)

- layer output ↦ output (blocks.i.hook_proj_out ↦ blocks.47.hook_resid_post)

- embed ↦ output hook_embed ↦ blocks.47.hook_resid_post)

Where output is the residual stream after the final layer has added its layer output.

We do a few separate experiments.

D.1. EAP

At the most high level, we can run (integrated gradient) EAP without positions. Using binary search to find the minimum number of edges to give us at least 85% accuracy results in the following adjacency matrix:

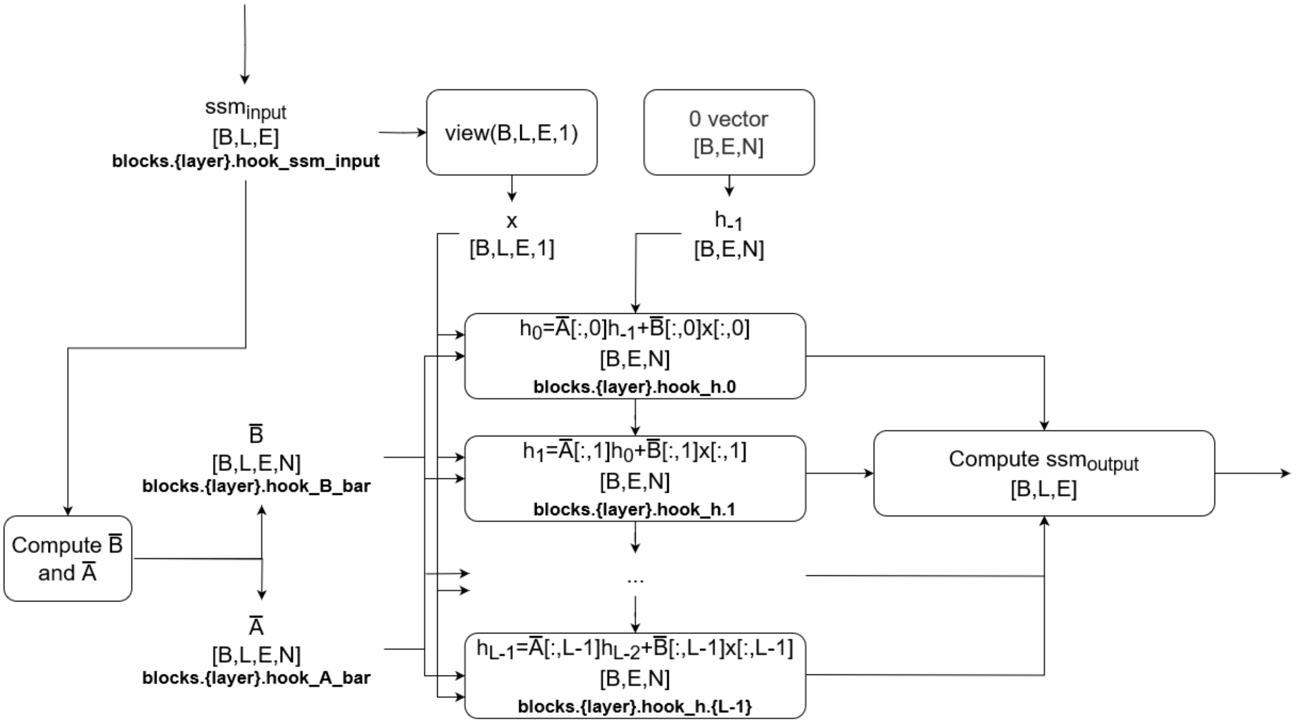

ssm_{input}
[B,L,E]
blocks.{layer}.hook_ssm_input
view(B,L,E,1)
0 vector
[B,E,N]
x
[B,L,E,1]
h_{-1}
[B,E,N]
$h_0=\bar{A}[:,0]h_{-1}+\bar{B}[:,0]x[:,0]$
[B,E,N]
blocks.{layer}.hook_h.0
$\bar{B}$
[B,L,E,N]
blocks.{layer}.hook_B_bar
$h_1=\bar{A}[:,1]h_0+\bar{B}[:,1]x[:,1]$
[B,E,N]
blocks.{layer}.hook_h.1
Compute ssm_{output}
[B,L,E]
Compute $\bar{B}$ and $\bar{A}$
...
$\bar{A}$
[B,L,E,N]
blocks.{layer}.hook_A_bar
$h_{L-1}=\bar{A}[:,L-1]h_{L-2}+\bar{B}[:,L-1]x[:,L-1]$
[B,E,N]
blocks.{layer}.hook_h.{L-1}

Figure 9. Internals of the SSM block, we restrict this diagram to only the parts we are interested in

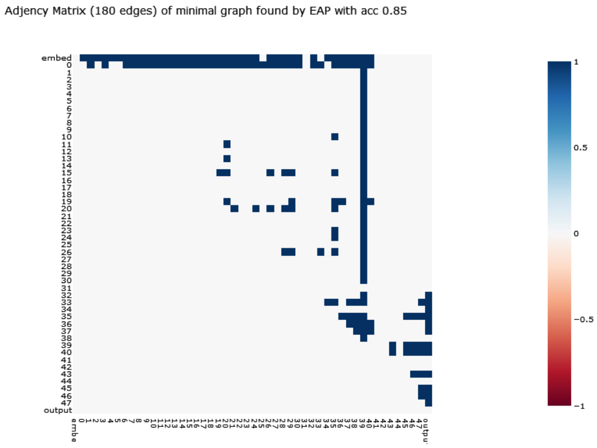


Figure 10. Integrated gradients EAP, minimum set of edges. A blue dot means the edge is present. The y-axis is the input node, the x-axis is the output node

We also present the edge attributions here, and the corresponding actual effects on normalized logit diff (determined during the ACDC run below)

This is not very insightful, so we clamp values to let us see more (the thresholds chosen by hand)

D.2. ACDC for layer information

We run EAP without positions using the edges listed above, and use binary search to find the least edges needed to get 85% accuracy. We take the resulting graph and run ACDC on it with a thresh of 0.0001, with (non-positional) edges for individual conv slices, the ssm, the skip connection, and all the edges above.

This allows us to get a hint at what parts each layer is using.

This resulting circuit has an average normalized logit diff of 0.84 and achieves 88% accuracy on a held-out test set, so there is little loss in performance from doing this further prune (because the thresh is so low, most edges pruned are those that decrease ability to do the task, which is why accuracy has gone up).

Of note, this reproduces the ”Convs of layer 39 shift names to the next position” result from above.

Ideally we could do this with EAP and no longer need ACDC, but but leave that for future work.

From the

D.3. EAP With Positions

In the following:

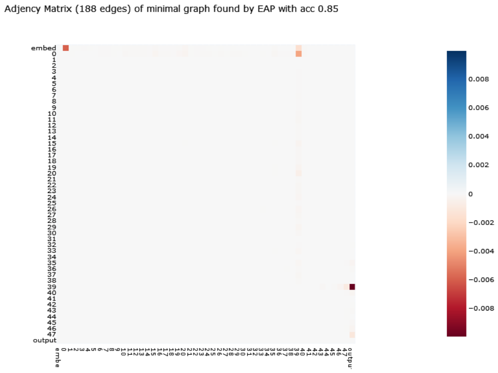

Adjency Matrix (188 edges) of minimal graph found by EAP with acc 0.85

Figure 11. Edge Attributions

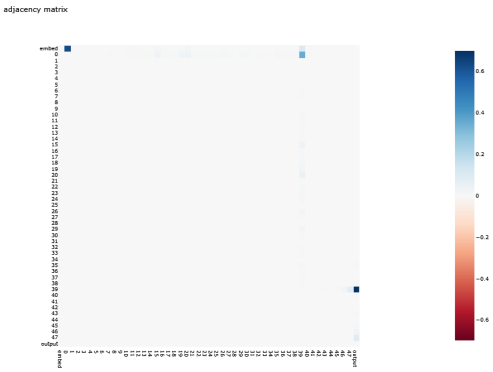
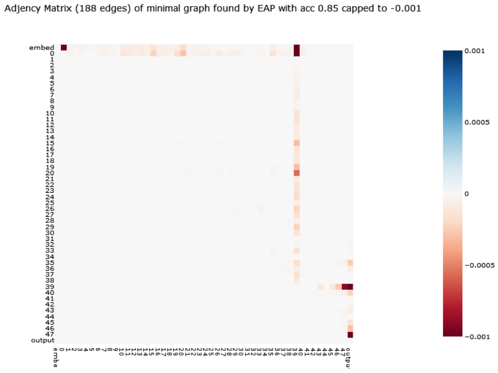
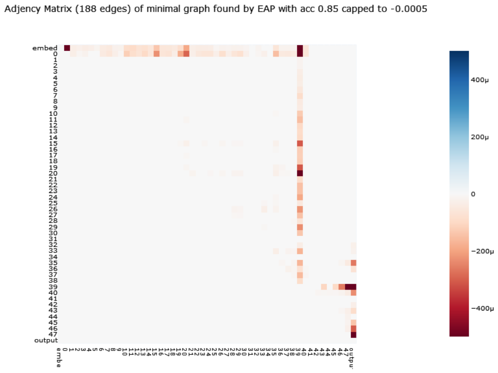
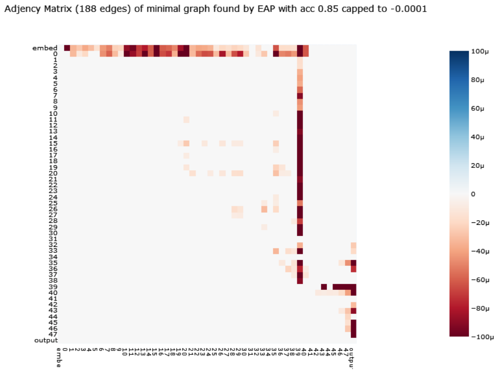
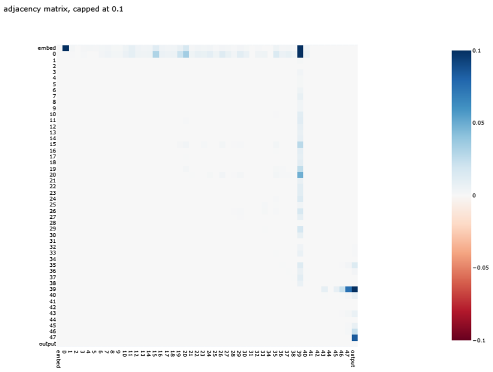

adjacency matrix

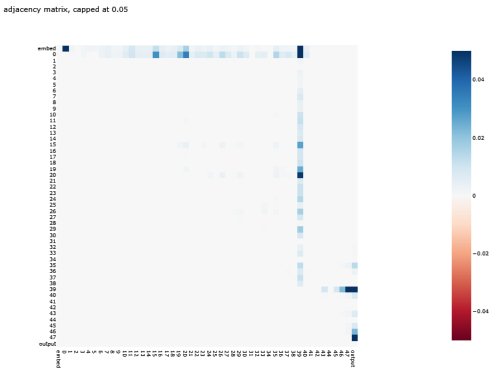
Figure 12. Effects on normalized logit diff

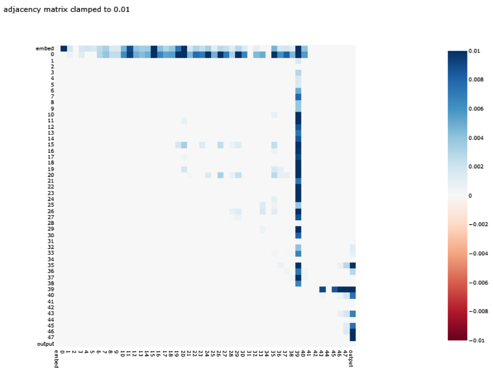

Adjency Matrix (188 edges) of minimal graph found by EAP with acc 0.85 capped to -0.001

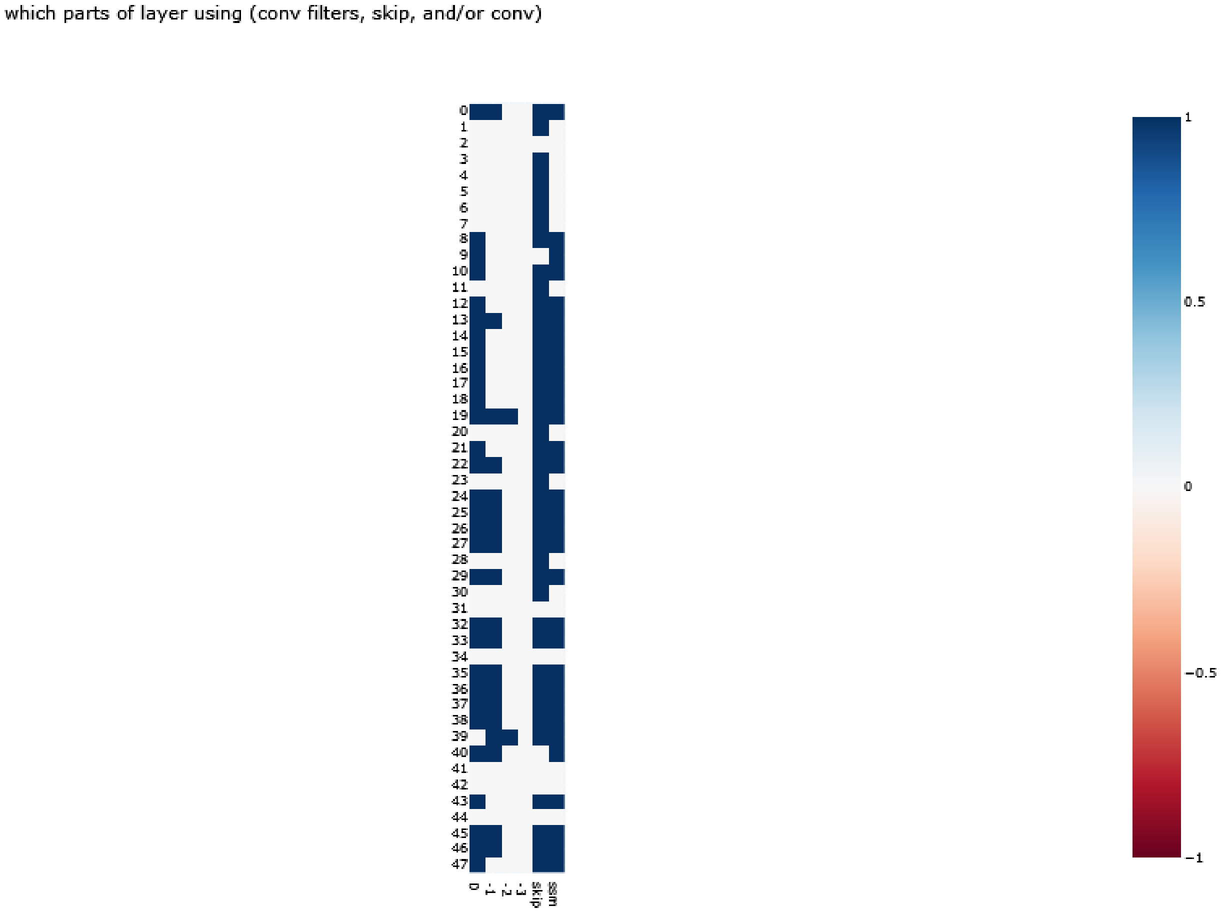

Adjency Matrix (188 edges) of minimal graph found by EAP with acc 0.85 capped to -0.0005


Adjency Matrix (188 edges) of minimal graph found by EAP with acc 0.85 capped to -0.0001


adjacency matrix, capped at 0.1


adjacency matrix, capped at 0.05


adjacency matrix clamped to 0.01



which parts of layer using (conv filters, skip, and/or conv)

Figure 13. ACDC results from inside layers. Each node is 1 if the edge is present, 0 if it is not. 0,-1, and -2 are the corresponding conv slices

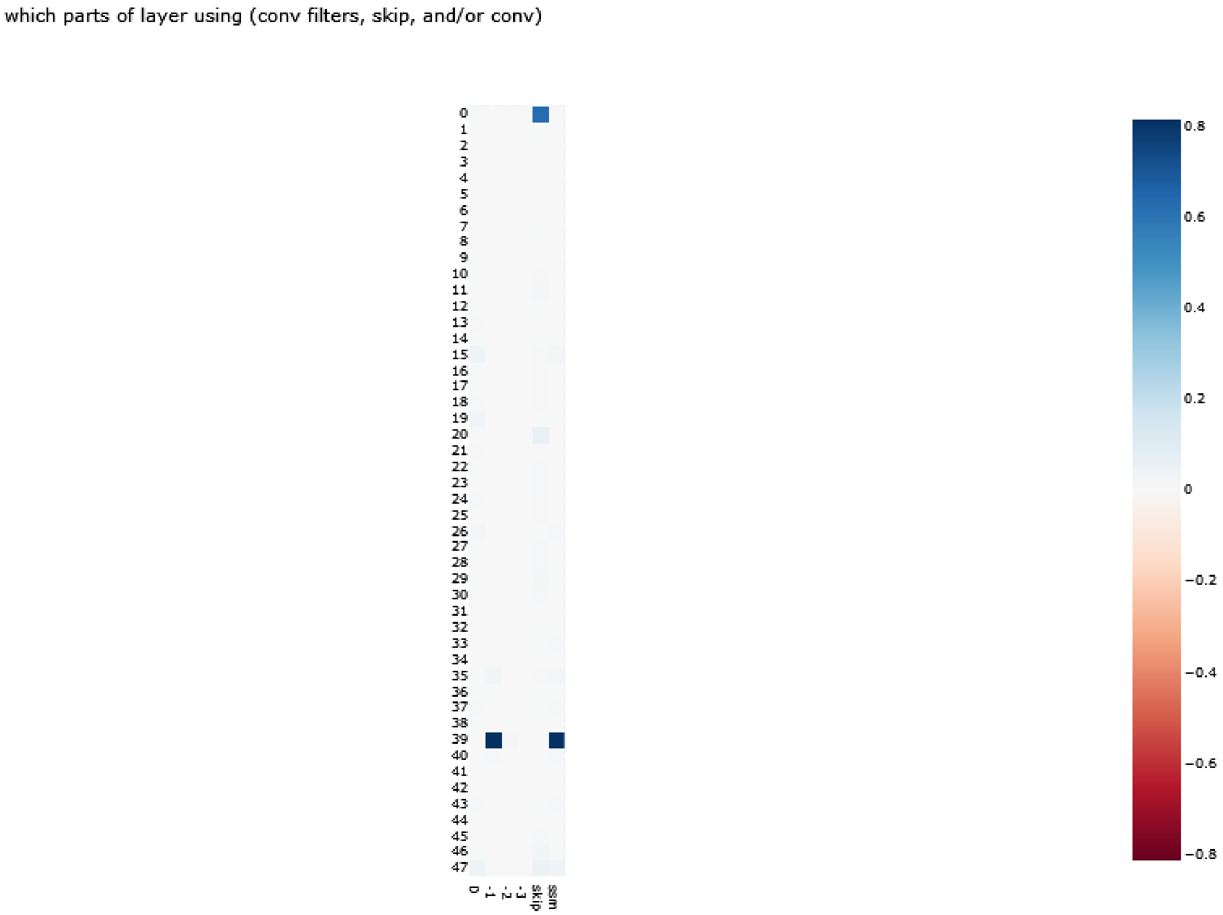


Figure 14. Same as the above figure, however, each cell shows the decrease in normalized logit diff if that edge is patched

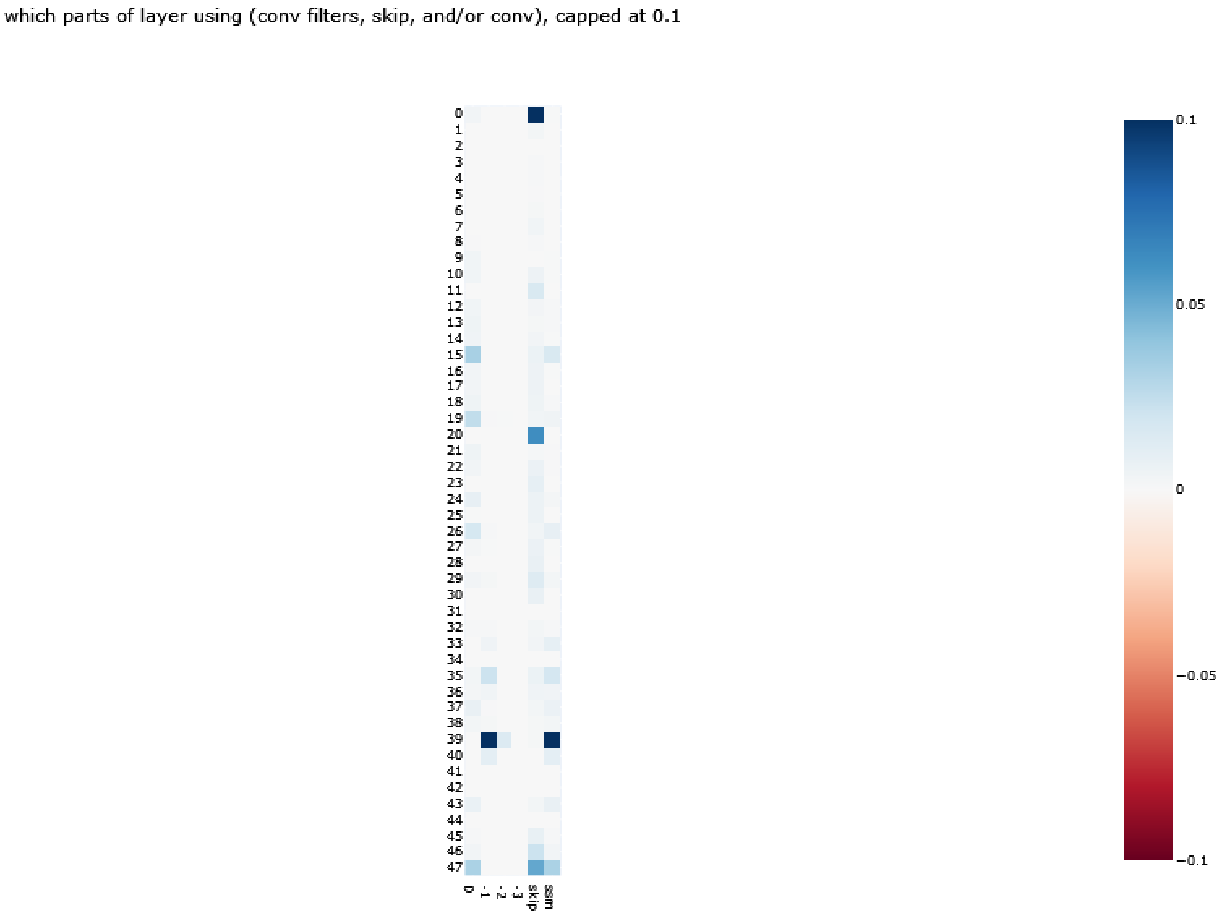

which parts of layer using (conv filters, skip, and/or conv), capped at 0.1

Figure 15. Same as above, however, values are clamped to 0.1

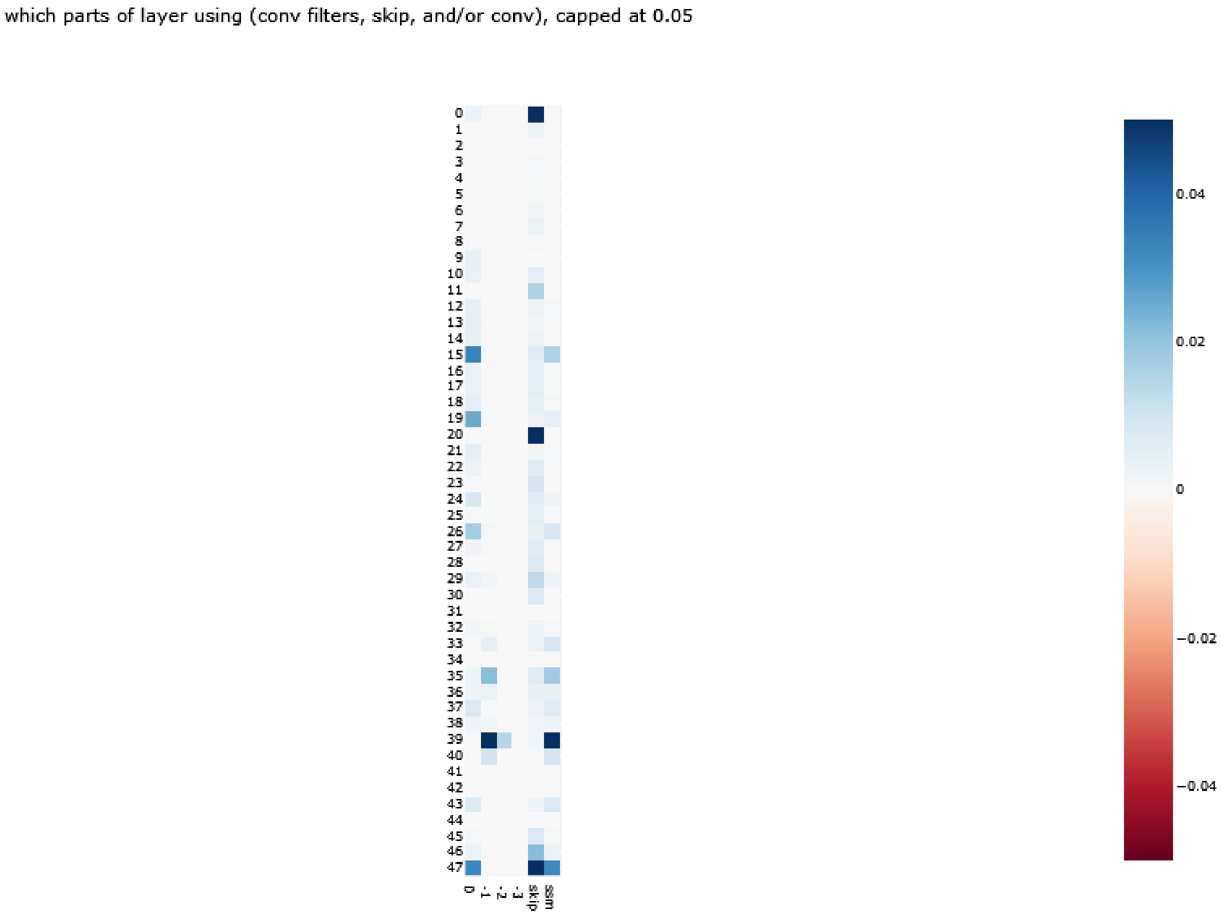


Figure 16. Same above, however, values are clamped to 0.05

- n1 means the first name in the prompt, n2 means the second name in the prompt, etc.
- pos0 means the first token, pos1 means the second token, etc. (these are used for non-name tokens)
- out means the final token, where the answer is generated

For reference, here is a prompt:

```
pos0 <|endoftext|>
pos1 Then
pos2 ,
n1 Sally
pos4 ,
n2 Martha
pos6 and
n3 Edwin
pos8 went
pos9 to
pos10 the
pos11 restaurant
pos12 .
n4 Edwin
pos14 and
n5 Sally
pos16 gave
pos17 a
pos18 drink
out to
```

D.3.1. Connections from embed

Every layer receives n1-n5, except:

```
Missing n1: layers 1, 5, 8, 25, 28, 29,
     30, 32, 34, 36
Missing n2: 30, 31, 36
Missing n3: layers 25, 27, 29, 30, 31,
     34, 36, 38
Missing n4: layers 3, 9, 25, 32, 34, 40
Missing n5: layer 9, 25, 27, 31, 34, 37,
     38, 40
Missing n1-n5: 33, 41-47
TODO: Output?
```

D.3.2. Connections of Layer 0

Layer 0 takes as input n1-n5, and sends n1-n5 to every layer, except:

```
Missing n1: 1, 2, 3, 28, 31, 32, 33, 34
Missing n2: 6, 7, 31, 33
Missing n3: 2, 31, 33
Missing n4: 1, 2, 3, 9, 18, 21, 32, 34, 40
Missing n5: 3, 9, 18, 31, 32, 34, 40
Missing n1-n5: 4, 5, 41-47
```

D.3.3. Layers that are missing names

If we consinder a layer as ”having” a name if it received it from embed or layer 0, the following layers have n1-n5:

And these are ones that are missing names:

```
Missing n1: 1, 5, 28, 32, 33, 34
Missing n2: 31, 33
Missing n3: 31, 33
Missing n4: 32, 34, 3, 40, 9
Missing n5: 9, 31, 34, 40
Missing n1-n5: 41-47
```

Of those, `1, 3, 5, 9` are only connected to 0/embed and 39. In particular, we have:

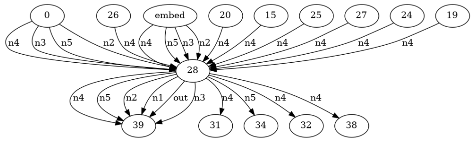
```
1: missing n1
   n1,n2,n3,n5 -> 39
3: missing n4
   n1-n5 -> 39
5: missing n1
   n1-n5 -> 39
9: missing n4,n5
   n1-n5 -> 39
```

Otherwise, we have

```
Missing n1: 28, 32, 33, 34
Missing n2: 31, 33
Missing n3: 31, 33
Missing n4: 32, 34, 40
Missing n5: 31, 34, 40
Missing n1-n5: 41-47
```

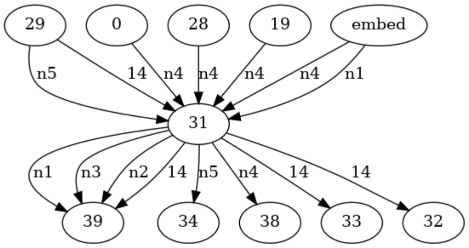
`28, 31, 32, 33, 34, 40` all seem to be involved in a complex circuit, and have inputs from other layers.

Just examining the missing terms:



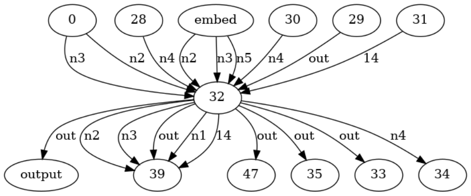
```
28 is missing n1

28 does not receive n1 from anyone
  outputs n1 to 39
```



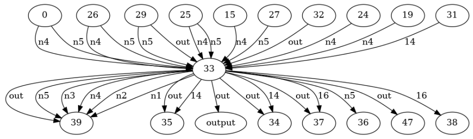
```
31 is missing n2, n3, n5

31 does not receive n2 or n3 from anyone
    output n2-n3 to 39
31 receives n5 from 29
    outputs n5 to 34
```



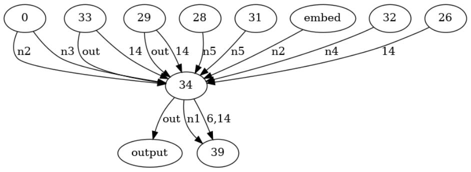
```
32 is missing n1, n4

32 does not receive n1
   outputs n1 to 39
32 receives n4 from 28, 30
   outputs n4 to 34
```



```
33 is missing n1-n3

33 does not receive n1-n3
   outputs n2-n3 to 39
   outputs n1 to 35
```

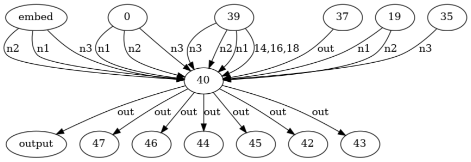


```
34 is missing n1, n4, n5

34 does not receive n1 from anyone
    outputs n1 to 39
```

```
34 receives n4 from 32
    does not output it

34 receives n5 from 28, 31
    does not output it
```



```
40 is missing n4, n5

40 does not receive n4-n5
    does not output n4-n5
```

D.3.4. Connections to 39

As expected, layer 39 stands out as noteworthy. Every layer before 39 has a connection to 39 for every name, with these exceptions:

```
Missing n1: layer 2
Missing n2: layer 34
Missing n3: layer 34
Missing n4: layers 1, 31, 32, 34
Missing n5: layers 2, 31, 32, 34
```

In addition, there are these extra connections to layer 39

```
pos6: layer 34
pos14: layers 31, 32, 34
out: layers  28, 29, 30, 33, 35, 37, 38
```

Where

- pos6 is the ” and” between n2 and n3
- pos14 is the “ and” between n4 and n5

D.3.5. Connections from 39

```
n1,n2,n3: layer 40
pos12: layer 43
pos14: layer 40
pos16: layers 40,41,43,44,45,46,47
pos18: layer 40
out: layers 43,45,46,47, and output
```

- pos12 is the ”.”
- pos14 is the ” and” between n4 and n5
- pos16 is the ” gave” after n5
- pos18 is the object (for example, ” drink”)

D.3.6. Graph after hiding 39, embed, and 0

Keeping the above in mind, once we hide those three nodes the graph is quite readable. If we hide 35, it is even more readable. We also plot layer 35, for reference.

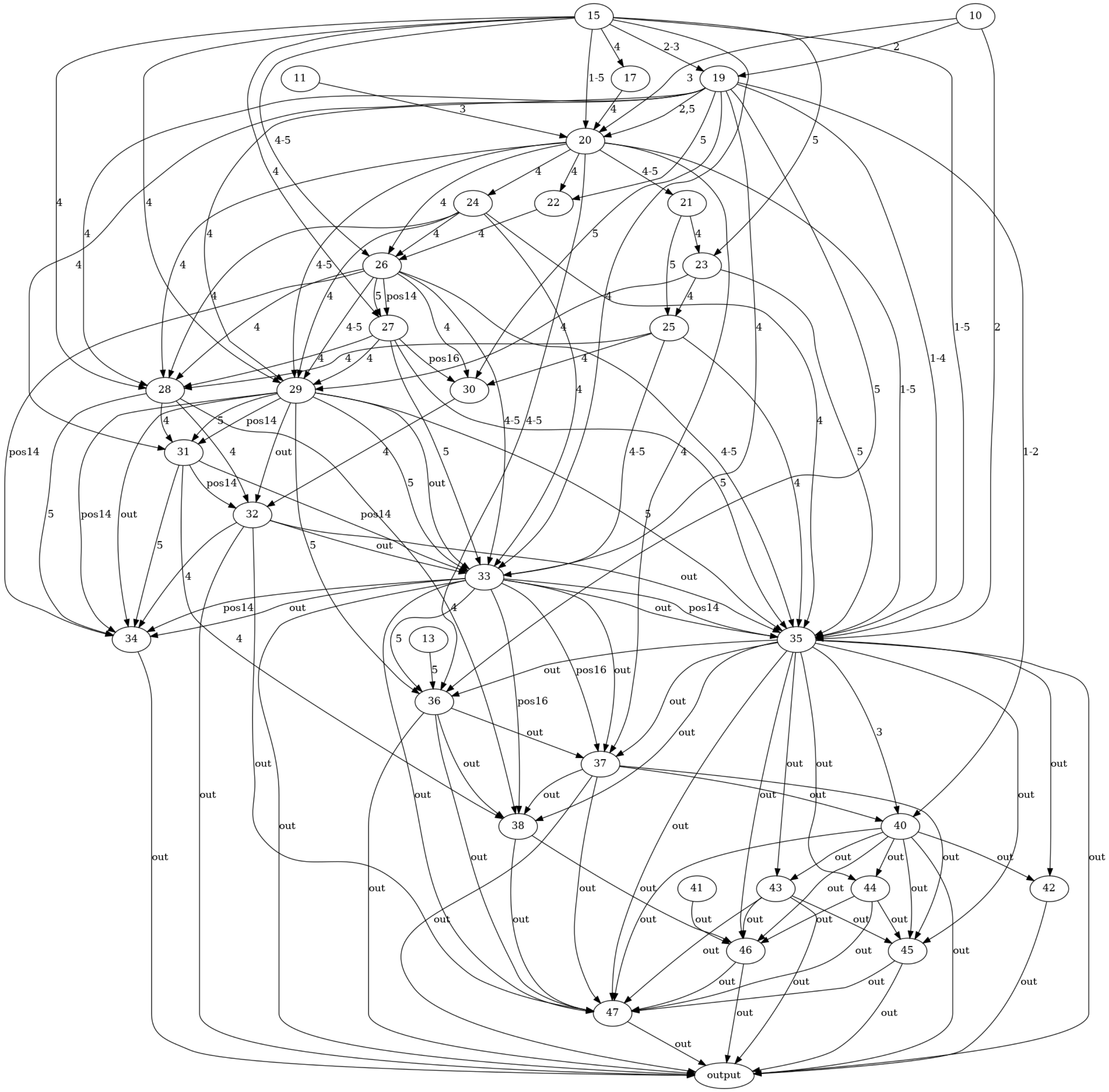


Figure 17. Positional EAP After hiding Embed, Layer 0, and Layer 39. Numbers correspond to names, pos14 means token in position 14, out means the final token

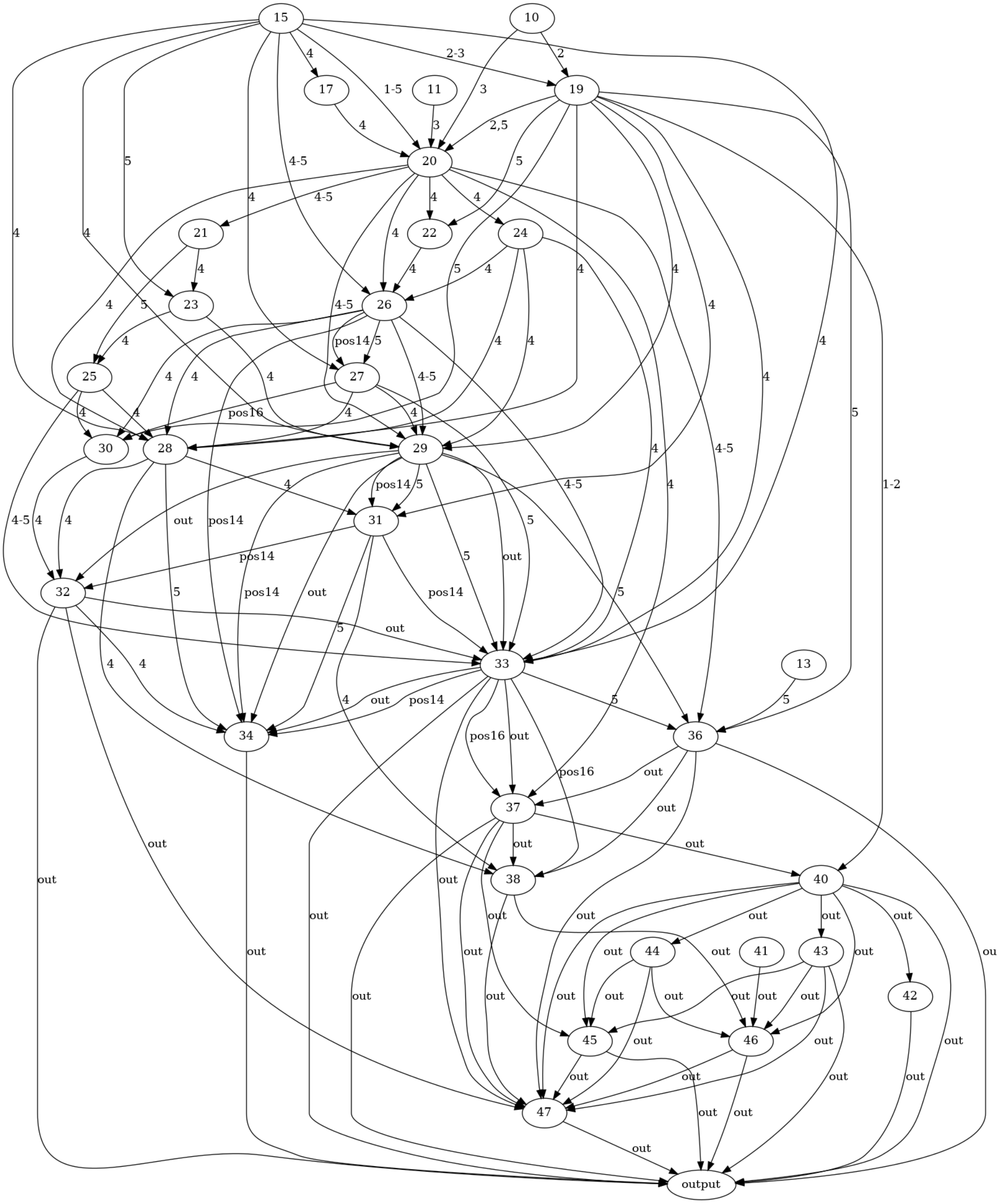


Figure 18. Same as above, but also hide 35

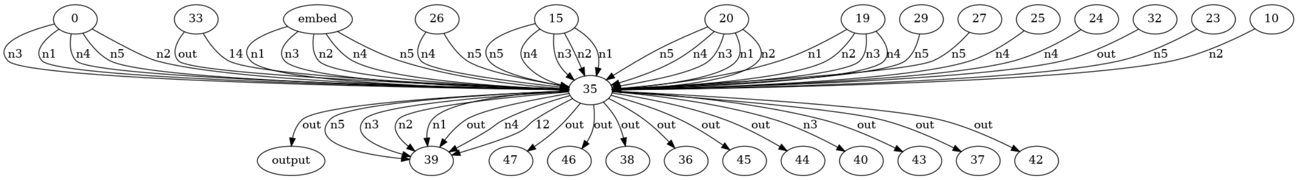


Figure 19. Layer 35 Positional EAP results