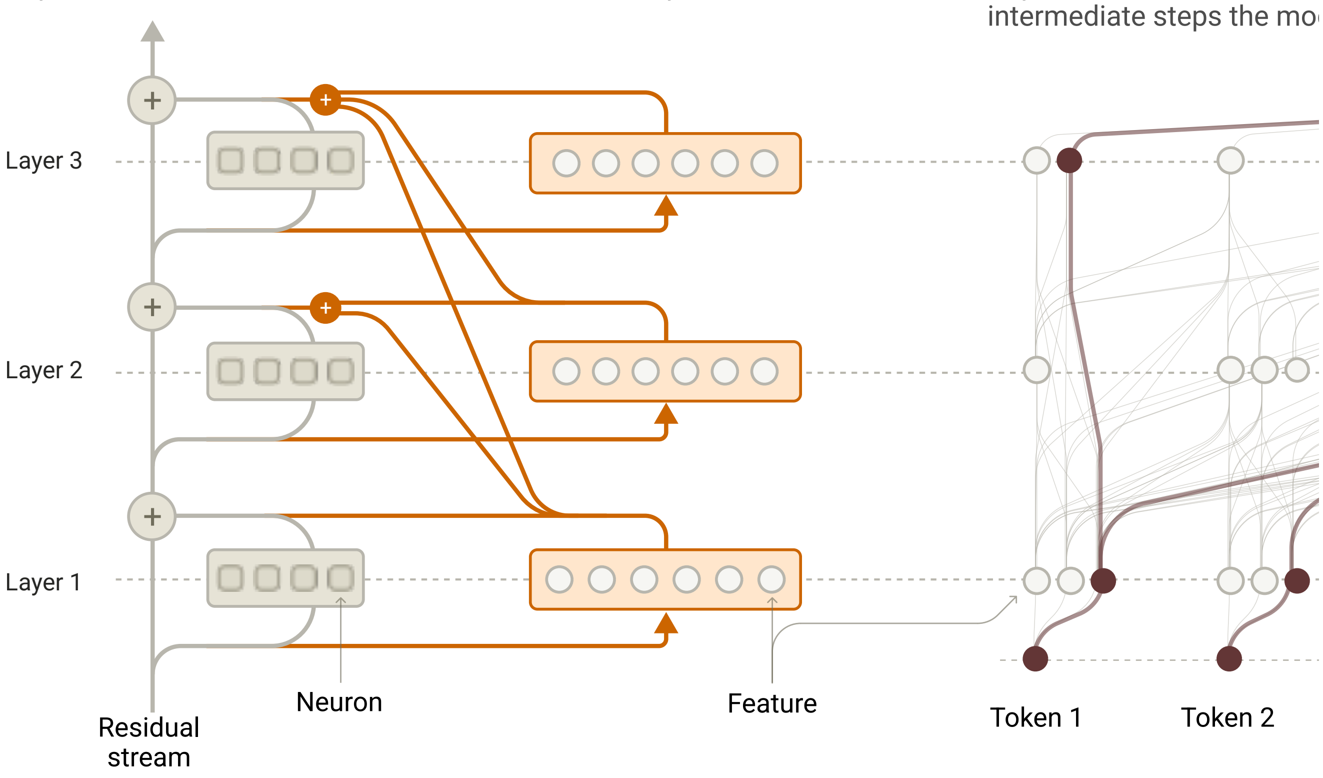
# Circuit Tracing: Revealing Computa Graphs in Language Models

**Replacement Model**

Replaces transformer model neurons with more interpretable features.

**Attribution Graph**

Depicts influence of featur intermediate steps the mo



We introduce a method to uncover mechanisms underlying behaviors of language mo of the model's computation on prompts of interest by tracing individual computationa replacement model substitutes a more interpretable component (here, a "cross-layer t model (here, the multi-layer perceptrons) that it is trained to approximate. We develop tools we use to investigate these "attribution graphs" supporting simple behaviors of a the groundwork for a companion paper applying these methods to a frontier model, C

AUTHORS

Emmanuel Ameisen,* Jack Lindsey,* Adam Pearce,* Wes Gurnee,* Nicholas L. Turner,* Brian Chen,* Craig Citro,*

David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton,

Trenton Bricken, Callum McDougall,◊ Hoagy Cunningham, Thomas Henighan,
Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson,

Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, Joshua Batson*‡

* Core Contributor; ‡ Correspondence to joshb@anthropic.com; ◊ Work performed while at Anthropic; Author contributions statement below.

AFFILIATIONS

Anthropic

PUBLISHED

March 27, 2025

Deep learning models produce their outputs using a series of transformations distributed across many computational units (artificial “neurons”). The field of *mechanistic interpretability* seeks to describe these transformations in human-understandable language. To date, our team’s approach has followed a two-step approach. First, we identify *features*, interpretable building blocks that the model uses in its computations. Second, we describe the processes, or *circuits,* by which these features interact to produce model outputs.

A natural approach is to use the raw neurons of the model as these building blocks.[1] Using this approach, previous work successfully identified interesting circuits in vision models, built out of neurons that appear to represent meaningful visual concepts [4]. However, model neurons are often *polysemantic* — representing a mixture of many unrelated concepts. One reason for polysemanticity is thought to be the phenomenon of *superposition* [5, 6, 7], in which models must represent more concepts than they have neurons, and thus must “smear” their representation of concepts across many neurons. This mismatch between the network's basic computational units (neurons) and meaningful concepts has proved a major impediment to progress to the mechanistic agenda, especially in understanding language models.

In recent years, sparse coding models such as sparse autoencoders (SAEs) [8, 9, 10, 11], transcoders [12, 13, 14], and crosscoders [15] have emerged as promising tools for identifying interpretable features represented in superposition. These methods decompose model activations into sparsely active components (“features”),[2] which turn out in many cases to correspond to human-interpretable concepts. While current sparse coding methods are an imperfect way of identifying features (see § 7 Limitations), they produce interpretable enough results that we are motivated to study *circuits* composed of these features. Several authors have already made promising steps in this direction [16, 12, 17].

Although the basic premise of studying circuits built out of sparse coding features sounds simple, the design space is large. In this paper we describe our current approach, which involves several key methodological decisions:

1. **Transcoders.** We extract features using a variant of *transcoders* [12, 14] rather than SAEs, which allows us to construct an interpretable "replacement model" that can be studied as a proxy for the original model. Importantly, this approach allows us to analyze direct feature-feature interactions.
2. **Cross-Layer.** We base our analysis on cross-layer transcoders (CLT) [15], in which each feature reads from the residual stream at one layer and contributes to the outputs of all subsequent MLP layers of the original model, which greatly simplifies the resulting circuits. Remarkably, we can substitute our learned CLT features for the model's MLPs while matching the underlying model's outputs in ~50% of cases.
3. **Attribution Graphs.** We focus on studying "attribution graphs" which describe the steps a model used to produce an output for a target token on a particular prompt, using an approach similar to Dunefsky et al. [12]. The nodes in the attribution graph represent active features, token embeddings from the prompt, reconstruction errors, and output logits. The edges in the graph represent linear effects between nodes, so the activity of each feature is the sum of its input edges (up to its activation threshold) (see § 3 Attribution Graphs).
4. **Linear Attribution Between Features.** We design our setup so that, for a specific input, the direct interactions between features are linear. This makes attribution a well-defined, principled operation. Crucially, we freeze attention patterns and normalization denominators (following [18]) and use transcoders to achieve this linearity.[3] Features also have indirect interactions, mediated by other features, which correspond to multi-step paths.
5. **Pruning.** Although our features are sparse, there are still too many features active on a given prompt to easily interpret the resulting graph. To manage this complexity, we prune graphs by identifying the nodes and edges which most contribute to the model's output at a specific token position (see § 5.2.4 Appendix: Graph Pruning). Doing so allows us to produce sparse, interpretable graphs of the model's computation for arbitrary prompts.
6. **Interface.** We designed an interactive interface for exploring attribution graphs, and the features they're composed of, that allows a researcher to quickly identify and highlight key mechanisms within them.
7. **Validation.** Our approach to studying circuits is indirect – our replacement model may use different mechanisms from the underlying model. Thus, it is important that we validate the mechanisms we find in attribution graphs. We do so using perturbation experiments. Specifically, we measure the extent to which applying perturbations in a feature's direction produces changes to other feature activations (and to the model's output) that are consistent with the attribution graph. We find that across prompts, perturbation experiments are generally qualitatively consistent with our attribution graphs, though there are some deviations.
8. **Global Weights.** While our paper mostly focuses on studying attribution graphs for individual prompts, our methods also allow us to study the weights of the replacement model ("global weights") directly, which underlie mechanisms across many prompts. In § 4 Global Weights, we demonstrate some challenges with doing so – naive global weights are often less interpretable than attribution graphs due to weight interference. However, we successfully apply them to understand the circuits underlying small number addition.

The goal of this paper is to describe and validate our methodology in detail, using a few case studies for illustration.

- We begin with methods. We describe the setup of our *replacement model* (§ 2 Building an Interpretable Replacement Model) and how we construct *attribution graphs* (§ 3 Attribution Graphs), concluding with two case studies (§ 3.7 Factual Recall Case Study, § 3.8 Addition Case Study). We then go on to explore approaches to constructing *global circuits*, including challenges and some preliminary methods for addressing them (§ 4 Global Weights).

- We then provide a detailed quantitative evaluation of our cross-layer transcoders and the resulting attribution graphs (§ 5 Evaluations), showing metrics by which CLTs provide Pareto-improvements over neurons and per-layer transcoders. Afterwards, we provide an overview of our companion paper, in which we apply our method to various behaviors of Claude 3.5 Haiku (§ 6 Biology). We follow with a discussion of methodological limitations (§ 7 Limitations). These include the role of attention patterns, the impact of reconstruction errors, the identification of suppression motifs, and the difficulty of understanding global circuits. Addressing these limitations, and seeing what additional model mechanisms are then revealed, is a promising direction for future work.

- We close with a broader discussion (§ 8 Discussion) of the design space of methods for producing attribution graphs – parts of our approach can be freely remixed with others while retaining much of the benefit – and a review of related work (§ 9 Related Work).
- Our companion paper, *On the Biology of a Large Language Model*, applies these methods to Claude 3.5 Haiku, investigating a diverse range of behaviors such as multiple hop reasoning, planning, and hallucinations.

We note that training a cross-layer transcoder can incur significant up-front cost and effort, which is amortized over its application to circuit discovery. We have found that this improves circuit interpretability and parsimony enough to justify the investment (see cost estimates for open-weights models and discussion of cost-matched performance relative to per-layer transcoders). Nevertheless, we stress that alternatives like per-layer transcoders or even MLP neurons can be used instead (keeping the same steps 3–8 above), and still produce useful insights. Moreover, it is likely that better methods than CLTs will be developed in the future.

To aid replication, we share guidance on CLT implementation, details on the pruning method, and the front-end code supporting the interactive graph analysis interface.

## Building an Interpretable Replacement Model

## Architecture

**Cross-Layer Transcoder**

Features read from one layer and write to all following ones

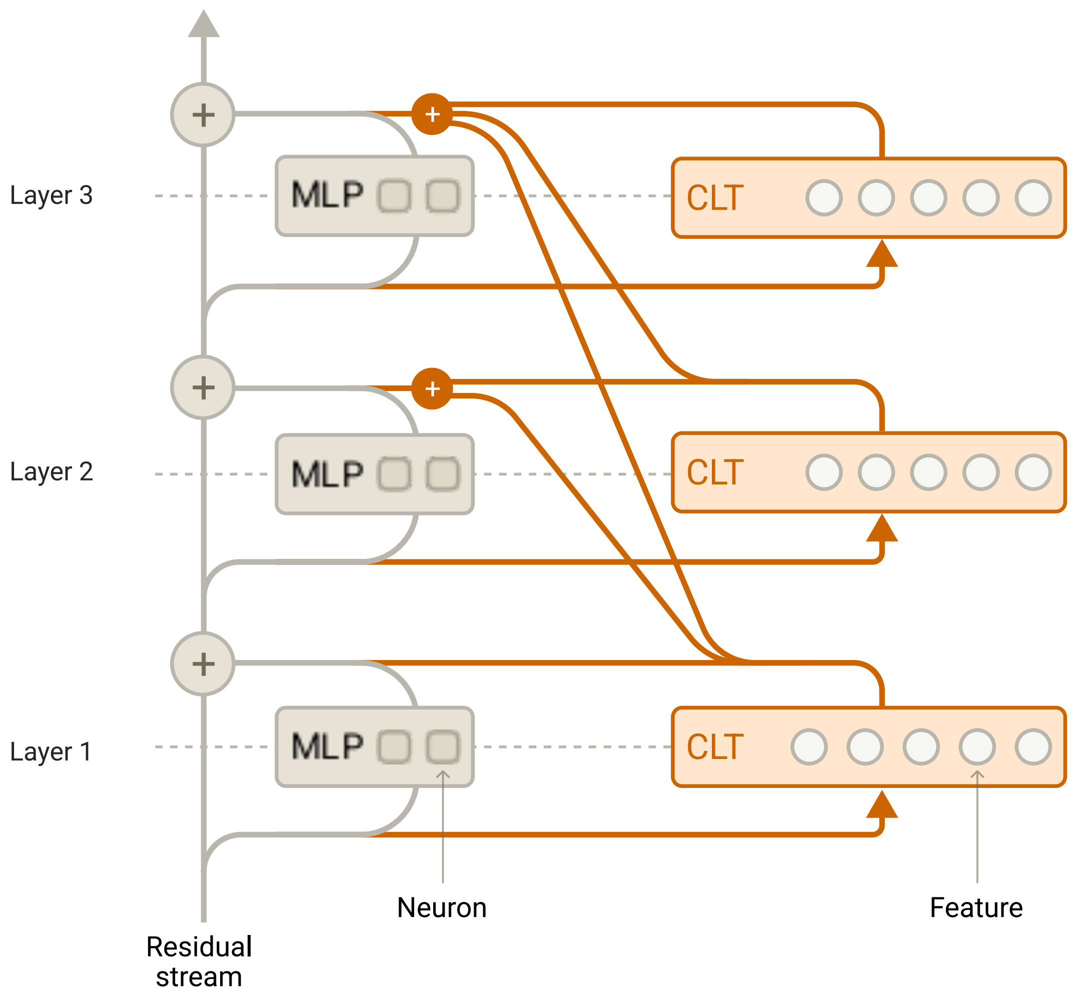


**Figure 1**: The cross-layer transcoder (CLT) forms the core architecture of our replacement model.

A cross-layer transcoder (CLT) consists of neurons ("features") divided into $L$ layers, the same number of layers as the underlying model. The goal of the model is to reconstruct the outputs of the MLPs of the underlying model, using sparsely active features. The features receive input from the model's residual stream at their associated layer, but are "cross-layer" in the sense that they can provide output to all subsequent layers.[4] Concretely:

- Each feature in the $\ell^{\text{th}}$ layer "reads in" from the residual stream at that layer using a linear encoder followed by a nonlinearity.
- An $\ell^{\text{th}}$ layer feature contributes to the reconstruction of the MLP outputs in layers $\ell, \ell+1, \ldots, L$, using a separate set of linear decoder weights for each output layer.
- All features in all layers are trained jointly. As a result, the output of the MLP in a layer $\ell'$ is jointly reconstructed by the features from all previous layers.

More formally, to **run** a cross-layer transcoder, let $\mathbf{x}^{\boldsymbol{\ell}}$ denote the original model's residual stream activations at layer $\ell$. The CLT feature activations $\mathbf{a}^{\boldsymbol{\ell}}$ at layer $\ell$ are computed as

$$\mathbf{a}^{\boldsymbol{\ell}} = \text{JumpReLU}\left(W_{enc}^{\ell} \mathbf{x}^{\boldsymbol{\ell}}\right)$$

where $W_{enc}^{\ell}$ is the CLT encoder matrix at layer $\ell$.

We let $\mathbf{y}^{\boldsymbol{\ell}}$ refer to the output of the original model's MLP at layer $\ell$. The CLT's attempted reconstruction $\hat{\mathbf{y}}^{\boldsymbol{\ell}}$ of $\mathbf{y}^{\boldsymbol{\ell}}$ is computed using the JumpReLU activation function [11] as:

$$\hat{\mathbf{y}}^{\boldsymbol{\ell}} = \sum_{\ell'=1}^{\ell} W_{dec}^{\ell' \to \ell} \mathbf{a}^{\boldsymbol{\ell'}}$$

where $W_{dec}^{\ell' \to \ell}$ is the CLT decoder matrix for features at layer $\ell'$ outputting to layer $\ell$.

To **train** a cross-layer transcoder, we minimize a sum of two loss functions. The first is a reconstruction error loss, summed across layers:

$$L_{\text{MSE}} = \sum_{\ell=1}^{L} \|\hat{\mathbf{y}}^{\boldsymbol{\ell}} - \mathbf{y}^{\boldsymbol{\ell}}\|^2$$

The second is a sparsity penalty (with an overall coefficient $\lambda$, a hyperparameter, and another hyperparameter $c$) summed across layers:

$$L_{\text{sparsity}} = \lambda \sum_{\ell=1}^{L} \sum_{i=1}^{N} \tanh(c \cdot \|\mathbf{W}_{\mathbf{dec,i}}^{\boldsymbol{\ell}}\| \cdot a_i^{\ell})$$

Where $N$ is the number of features per layer and $\mathbf{W}_{\mathbf{dec,i}}^{\boldsymbol{\ell}}$ is the concatenation of all decoder vectors of feature $i$.

We trained CLTs of varying sizes on a small 18-layer transformer model ("18L")[5] , and on Claude 3.5 Haiku. The total number of features across all layers ranged from 300K to 10M features (for 18L) and from 300K to 30M features (for Haiku). For more training details see § D Appendix: CLT Implementation Details.

## From Cross-Layer Transcoder to Replacement Model

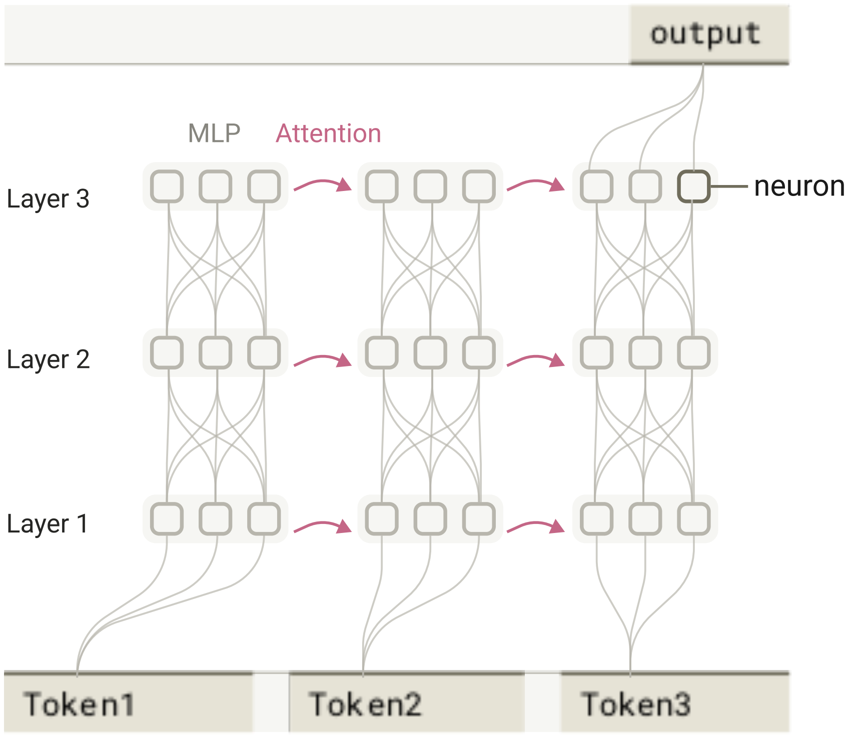
Given a trained cross-layer transcoder, we can define a “replacement model” that substitutes the cross-layer transcoder features for the model’s MLP neurons – that is, where each layer's MLP output is replaced by its reconstruction by all CLTs that write to the that layer. Running a forward pass of this replacement model is identical to running the original model, with two modifications:

- Upon reaching the *input* to the MLP in layer $\ell$, we compute the activations of the cross-layer transcoder features whose encoders live in layer $\ell$.
- Upon reaching the *output* of the MLP in layer $\ell$, we overwrite it with the summed outputs of the cross-layer transcoder features in this and previous layers, using their decoders for layer $\ell$.

Attention layers are applied as usual, without any freezing or modification. Although our CLTs were only trained using input activations from the underlying model, “running” the replacement model involves running CLTs on "off-distribution" input activations from intermediate activations from the replacement model itself.

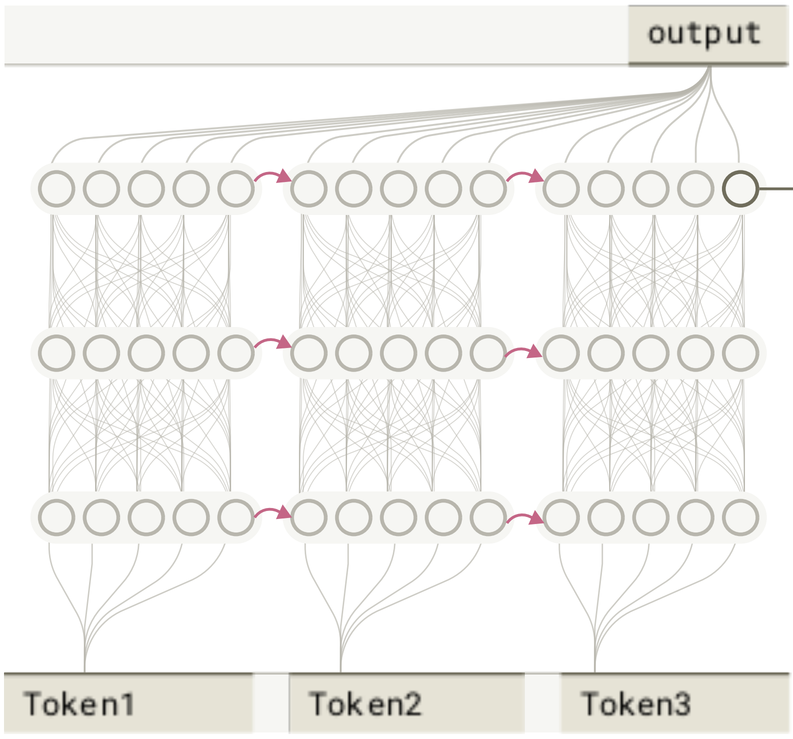
## Original Transformer Model

The underlying model that we study is a transformer-based large language model.

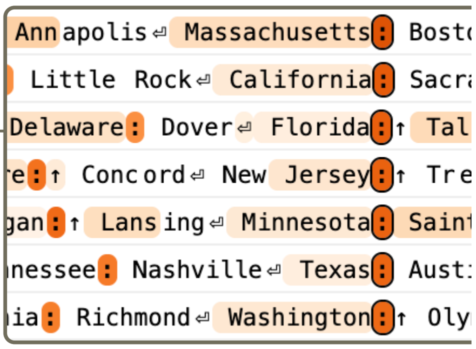


## Replacement Model

We replace the neurons of the original model with *features*. There are typically more featur
than neurons. Features are sparsely active and often represent interpretable concepts.



### Feature


Annapolis⏎ Massachusetts: Bosto
Little Rock⏎ California: Sacra
Delaware: Dover⏎ Florida:↑ Tal
e:↑ Concord⏎ New Jersey:↑ Tre
gan:↑ Lansing⏎ Minnesota: Saint
nessee: Nashville⏎ Texas: Aust
ia: Richmond⏎ Washington:↑ Oly

To understand what a feature re
we use a *feature visualization*, v
dataset examples for which the
most strongly active. In this exa
feature fires strongly when the r
about to say a state capital.

**Figure 2**: The replacement model is obtained by replacing the original model's neurons with the cross-layer transcoder's spars

**Figure 2**: The replacement model is obtained by replacing the original model's neurons with the cross-layer transcoder's spars

As a simple evaluation, we measure the fraction of completions for which the most likely token output of the replacement model matches that of the underlying model. The fraction improves with scale, and is better for CLTs compared to a per-layer transcoder baseline (i.e., each layer has a standard single layer transcoder trained on it; the number of features shown refers to the total number across all layers). We also compare to a baseline of thresholded neurons, varying the threshold below which neurons are zeroed out (empirically, we find that higher neuron activations are increasingly interpretable, and we indicate below where their interpretability roughly matches that of features according to our auto-evaluations in § 5.1.2 Quantitative CLT Evaluations). Our largest 18L CLT matches the underlying model's next-token completion on 50% of a diverse set of pretraining-style prompts from an open source dataset (see § R Additional Evaluation Details).[6]

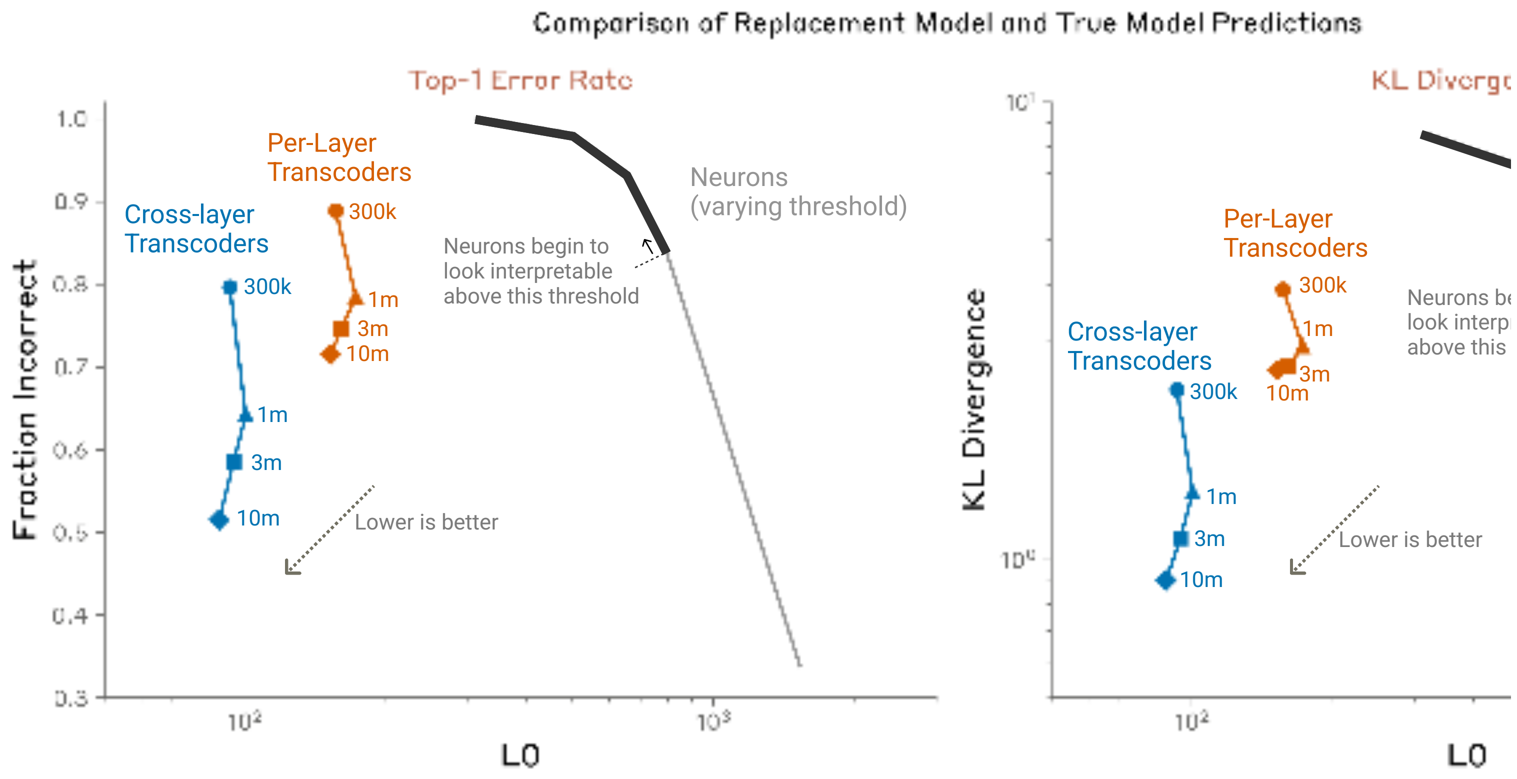


**Figure 3**: Comparison of cross-layer transcoders, per-layer transcoders, and thresholded neurons in terms of their top-1 accuracy and KL divergence when used as the basis for a rep thresholds are determined using sort and contrastive eval (see Evaluations).

## The Local Replacement Model

While running the replacement model can sometimes reproduce the same outputs as the underlying model, there is still a significant gap, and reconstruction errors can compound across layers. Since we are ultimately interested in understanding the underlying model, we would like to approximate it as closely as possible. To that end, when studying a fixed prompt $p$, we construct a *local replacement model*, which

- Substitutes the CLT for the MLP layers (as in the replacement model);
- Uses the attention patterns and normalization denominators from the underlying model's forward pass on $p$ (as in [18, 12]);
- Adds an error adjustment to the CLT output at each (token position, layer) pair equal to the difference between the true MLP output on $p$ and the CLT output on $p$ (as in [16]).

After this error adjustment and freezing of attention and normalization nonlinearities, we've effectively re-written the underlying model's computation on the prompt $p$ in terms of different basic units; all of the error-corrected replacement model's activations and logit outputs exactly match those of the underlying model. However, this does not guarantee that the local replacement model and underlying model use the same *mechanisms*. We can measure differences in mechanism by measuring how differently these models respond to perturbations; we refer to the extent to which perturbation behavior matches as "mechanistic faithfulness", discussed in § 5.3 Evaluating Mechanistic Faithfulness.[7]

The local replacement model can be viewed as a very large fully connected neural network, spanning across tokens, on which we can do classic circuit analysis:

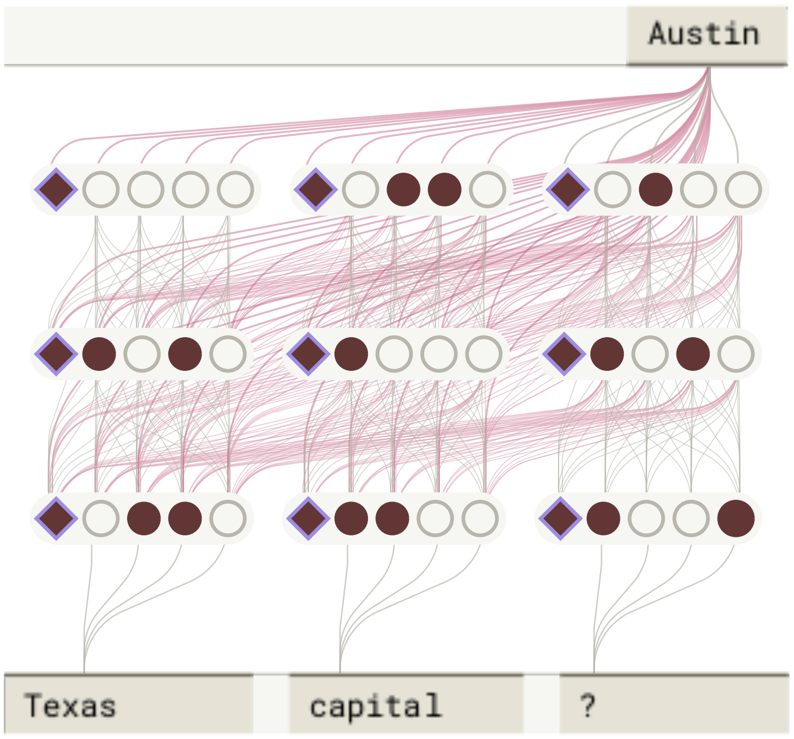
- Its input is the concatenated set of one-hot vectors for each token in the prompt.
- Its neurons are the union of the CLT features active at every token position.
- Its weights are the summed interactions over all the linear paths from one feature to another, including via the residual stream and through attention, but not passing through MLP or CLT layers. Because attention patterns and normalization denominators are frozen, the impact of a source feature's activation on a target feature's pre-activation via each path is linear in the activation of the source feature. We sometimes refer to these as "virtual weights" because they are not instantiated in the underlying model.
- Additionally, it has bias-like nodes corresponding to error terms, with a connection from each bias to each downstream neuron in the model.

The only nonlinearities in the local replacement model are those applied to feature preactivations.

The local replacement model serves as the basis of our *attribution graphs*, where we study the feature-feature interactions of the local replacement model on the prompt for which it was made. These graphs are the primary object of study of this paper.

## Local Replacement Model

The local replacement model is specific to a prompt of interest. We add an error adjustment freeze attention patterns to be what they were in the original model on the given prompt. It p exact same output as the original model, but replaces as much computation as possible witl



**Reconstruction Error**

Error nodes represent the diff between the original MLP out replacement model's reconstr

**Attention-mediated weights**

Attention patterns are frozen value in the original model, all to define weights between fea different token positions

## Attribution Graph

We trace from input to output through active features, pruning paths that don't influence the output.

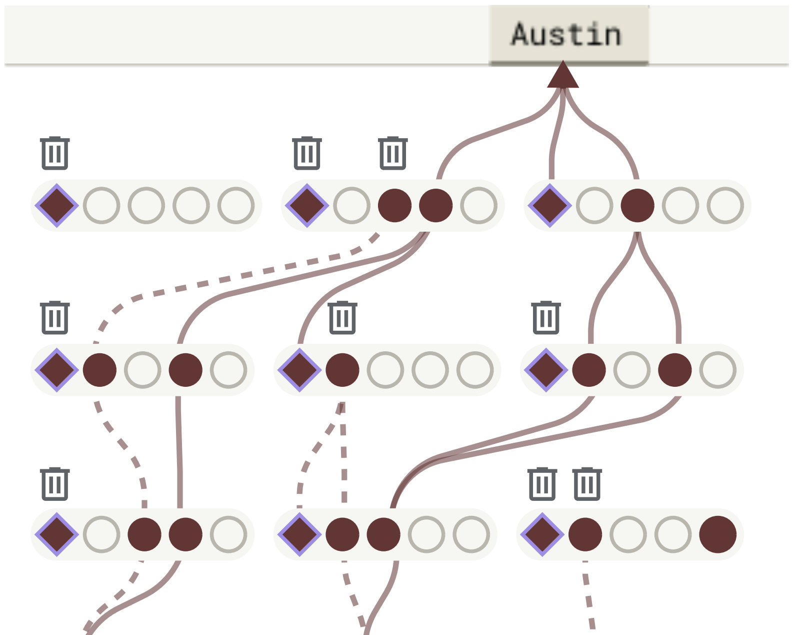
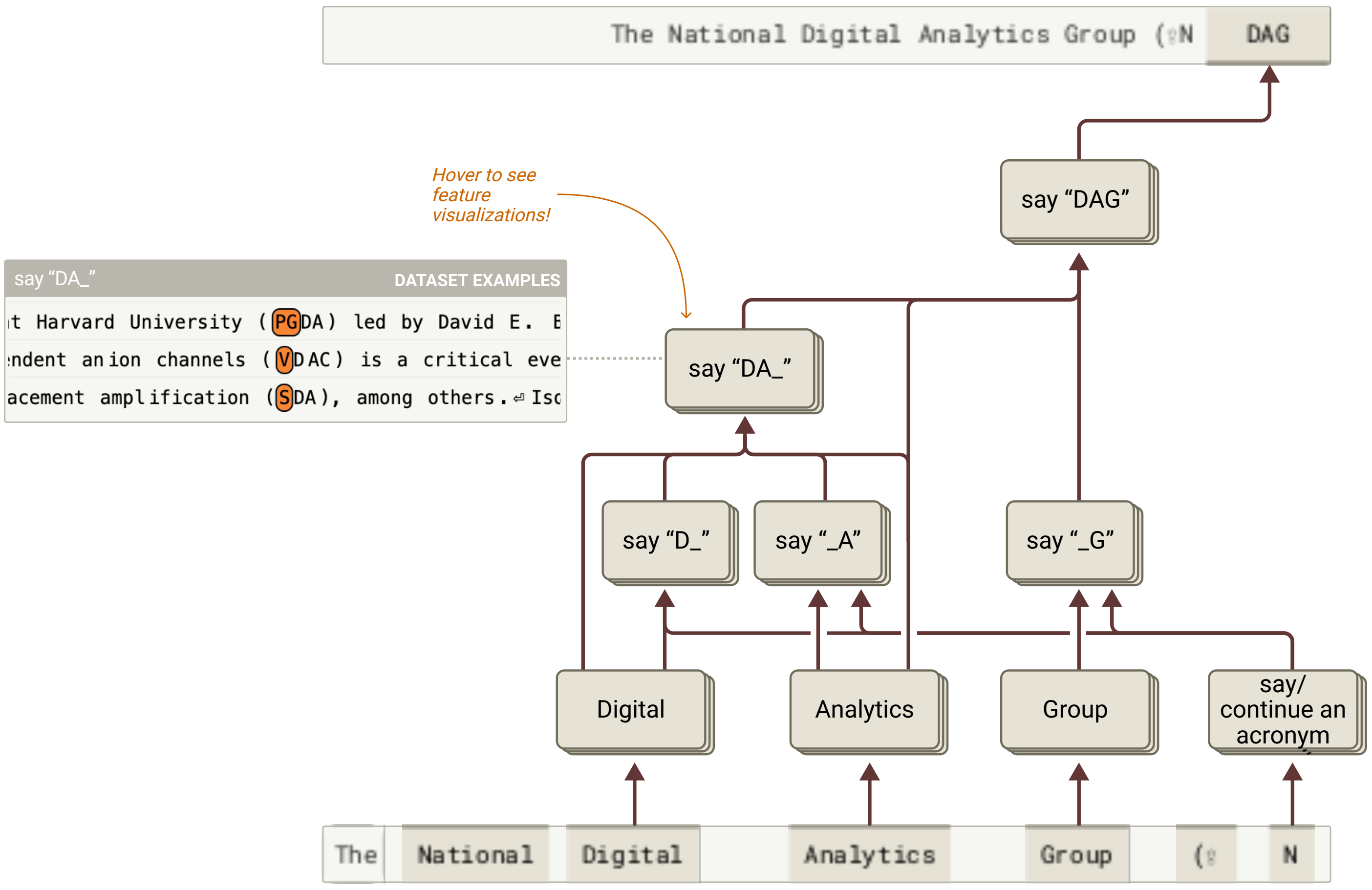


**Figure 4**: The local replacement model is obtained by adding error terms and fixed attention patterns to the replacement model t
reproduce the original model’s behavior on a specific prompt.

## Attribution Graphs

We will introduce our methodology for constructing attribution graphs while working through a case study regarding the model's ability to write acronyms for arbitrary titles. In the example we study, the model successfully completes a fictional acronym. Specifically, we give the model the prompt `The National Digital Analytics Group (N` and sample its completion: `DAG)`. The tokenizer the model was trained with uses a special "Caps Lock" token, which means the prompt and completion are tokenized as follows: `The` `National` `Digital` `Analytics` `Group` ` (` `⇪` `n` `dag`.

We explain the computation the model performs to output the "DAG" token by constructing an attribution graph showing the flow of information from the prompt through intermediary features and to that output.[8] Below, we show a simplified diagram of the full attribution graph. The diagram shows the prompt at the bottom and the model's completion on top. Boxes represent groups of similar features, and can be hovered over to display each feature's visualization. We discuss our interpretation of features in § 3.3 Understanding and Labeling Features. Arrows represent the direct effect of a group of features or a token on other features and the output logit.



The National Digital Analytics Group (⇪N
DAG
Hover to see feature visualizations!
say "DAG"
say "DA_"
DATASET EXAMPLES
t Harvard University (PGDA) led by David E. E
ndent anion channels (VDAC) is a critical eve
acement amplification (SDA), among others.⏎ Isc
say "D_"
say "_A"
say "_G"
Digital
Analytics
Group
say/ continue an acronym
The
National
Digital
Analytics
Group
(⇪
N

**Figure 5**: A simplified diagram of the attribution graph for 18L completing a fictional acronym.

The graph for the acronym prompt shows three main paths, originating from each of the tokens that compose the desired acronym. Paths originate from features for a given word, promoting features about "saying the first letter of that word in the correct position", which themselves have positive edges to a "say DAG" feature and the logit. "say X" labels describe "output features", which promote a specific token X, and arbitrary single letters are denoted with underscores. The "Word → say _W" edges represent attention heads' OV circuits writing to a subspace that is then amplified by MLPs at the target position. Each group of features also has a direct edge to the logit in addition to the sequential paths, representing effects mediated only via attention head OVs (i.e., paths to the output in the local replacement model that don't "touch" another MLP layer).

In order to output “DAG”, the model also needs to decide to output an acronym, and to account for the fact that the prompt already contains N, and indeed we see features for “in an acronym” and “in an N at the start of an acronym” with positive edges to the logit. The word National has minimal influence on the logit. We hypothesize that this is due to its main contribution being through influencing attention patterns, which our method does not explain (see § 7.1 Limitations: Missing Attention Circuits).

In the rest of this section, we explain how we compute and visualize attribution graphs.

## Constructing an Attribution Graph for a Prompt

To interpret the computations performed by the local replacement model, we compute a causal graph that depicts the sequences of computational steps it performs on a particular prompt. The core logic by which we construct the graph is essentially the same as that of Dunefsky *et al.* [12], extended to handle cross-layer transcoders. Our graphs contain four types of nodes:

- The *output* nodes correspond to candidate output tokens. We only construct output nodes for the tokens required to reach 95% of the probability mass, up to a total of 10.[9]
- The *intermediate* nodes correspond to active cross-layer transcoder features at each prompt token position.
- The primary *input* nodes of the graph correspond to the embeddings of the prompt tokens.
- Additional input nodes (“*error* nodes”) correspond to the portion of each MLP output in the underlying model left unexplained by the CLT.

Edges in the graph represent direct, linear attributions in the local replacement model. Edges originate from feature, embedding, and error nodes, and terminate at feature and output nodes. Given a source feature node $s$ and a target feature node $t$, the edge weight between them is defined to be $A_{s\to t} := a_s w_{s\to t}$, where $w_{s\to t}$ is the (virtual) weight in the local replacement model viewed as a fully connected neural network and $a_s$ is the activation of the source feature.[10]

In terms of the underlying model, $w_{s\to t}$ is a sum over all linear paths (i.e., through attention head OVs and residual connections) connecting the source feature's decoder vectors to the target feature's encoder vector.

We now give details on how to efficiently compute these in practice, using backwards Jacobians. Let $s$ be a source feature node at layer $\ell_s$ and context position $c_s$ and let $t$ be a target feature node at layer $\ell_t$ and context position $c_t$. We write $J^{\blacktriangledown}_{c_s,\ell_s\to c_t,\ell_t}$ for the Jacobian of the underlying model with a stop-gradient operation applied to all model components with nonlinearities – the MLP outputs, the attention patterns, and normalization denominators – on a backwards pass on the prompt of interest, from the residual stream at context position $c_t$ and layer $\ell_t$ to the residual stream at context position $c_s$ and layer $\ell_s$. The edge weight from $s$ to $t$ is then

$$A_{s\to t} = a_s w_{s\to t} = a_s \sum_{\ell_s \le \ell < \ell_t} (W^{\ell_s\to\ell}_{\text{dec, }s})^T J^{\blacktriangledown}_{c_s,\ell\to c_t,\ell_t} W^{\ell_t}_{\text{enc, }t},$$

where

- $W^{\ell_s\to\ell}_{\text{dec, }s}$ is the decoder vector of the feature for $s$ writing to layer $\ell$,
- $W^{\ell_t}_{\text{enc, }t}$ is the encoder vector of the feature for $s$.

The formulas for the other edge types are similar, e.g., an embedding-feature edge weight is given by $w_{s\to t} = \mathrm{Emb}_s^T J^{\blacktriangledown}_{c_s,\ell_s \to c_t,\ell_t} W^{\ell_t}_{\mathrm{enc},\ t}$. Note that error nodes have no input edges. For all such formulas, and an expansion of the Jacobian in terms of paths in the underlying model, see § E Appendix: Attribution Graph Computation.)

Because we have added stop-gradients to all model nonlinearities in the computation above, the preactivation $h_t$ of any feature node $t$ is simply the sum of its incoming edges in the graph: $h_t = \sum_{\mathcal{S}_t} w_{s\to t}$ where $S_t$ is the set of nodes at earlier layers and equal or earlier context positions as $t$. Thus the attribution graph edges provide a linear decomposition of each feature's activity.

Note that these graphs do not contain information about the influence of nodes on other nodes via their influence on attention *patterns*, but do contain information about node-to-node influence through the *outputs* of frozen attention. In other words, we account for the information which flows from one token position to another, but not why the model moved that information.[11] Note also that the outgoing edges from a cross-layer feature aggregate the effect of its decodings at *all* of the layers that it writes to on downstream features.

While our replacement model features are sparsely active (on the order of a hundred active features per token position), attribution graphs are too large to be viewed in full, particularly as prompt length grows – the number of edges can grow to the millions even for short prompts. Fortunately, a small subgraph typically accounts for most of the significant paths from the input to the output.

To identify such subgraphs, we apply a pruning algorithm designed to preserve nodes and edges that directly or indirectly exert significant influence on the logit nodes. With our default parameters, we typically reduce the number of nodes by a factor of 10, while only reducing the behavior explained by 20%. See § F Appendix: Graph Pruning for methodological details of our algorithms and metrics.

## Learning from Attribution Graphs

Even following pruning, attribution graphs are quite information-dense. A pruned graph often contains hundreds of nodes and tens of thousands of edges – too much information to interpret all at once. To allow us to navigate this complexity, we developed an interactive attribution graphs visualization interface. The interface is designed to enable “tracing” key paths through the graph, retain the ability to revisit previously explored nodes and paths, and materialize the information needed to interpret features on an as-needed basis.

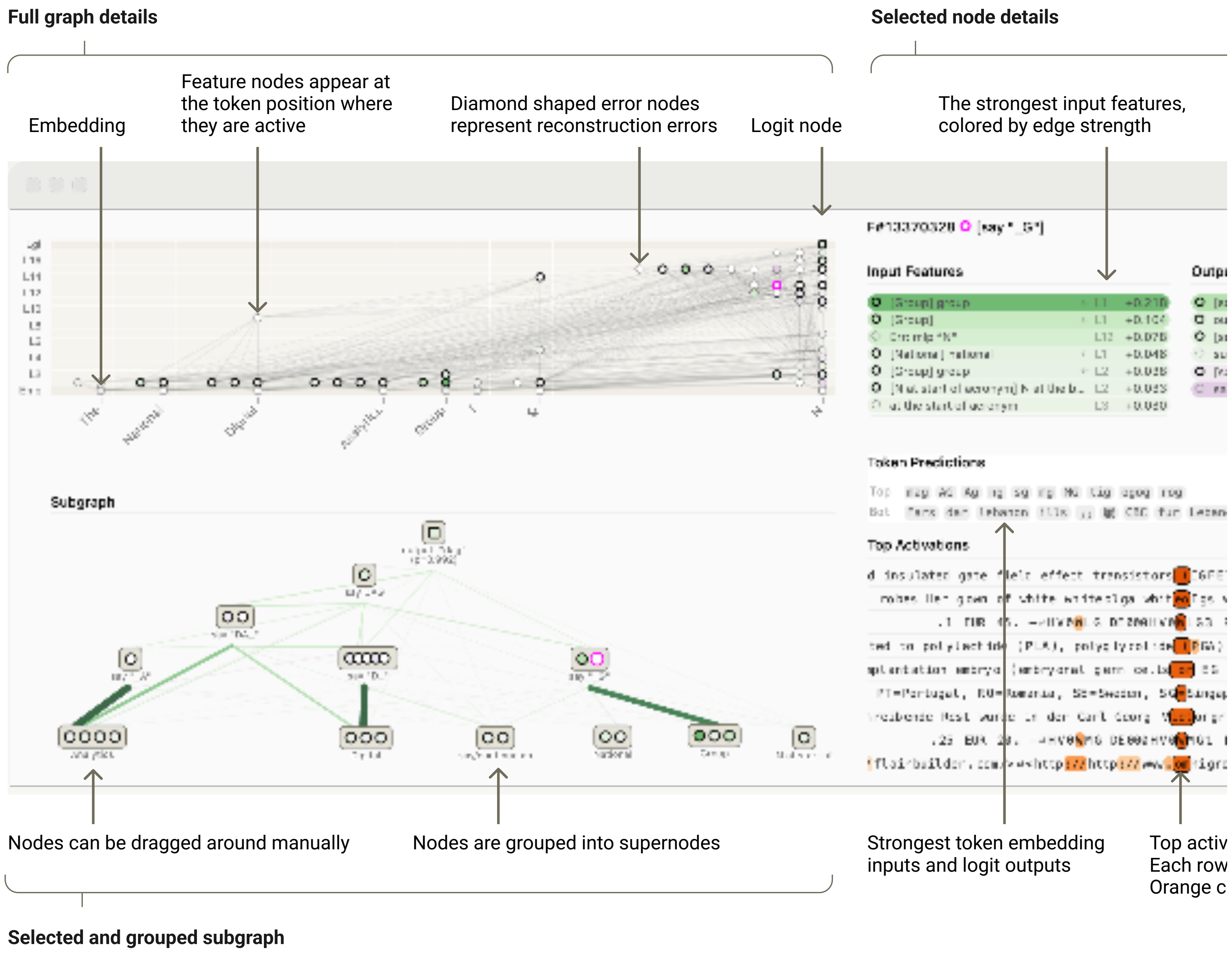
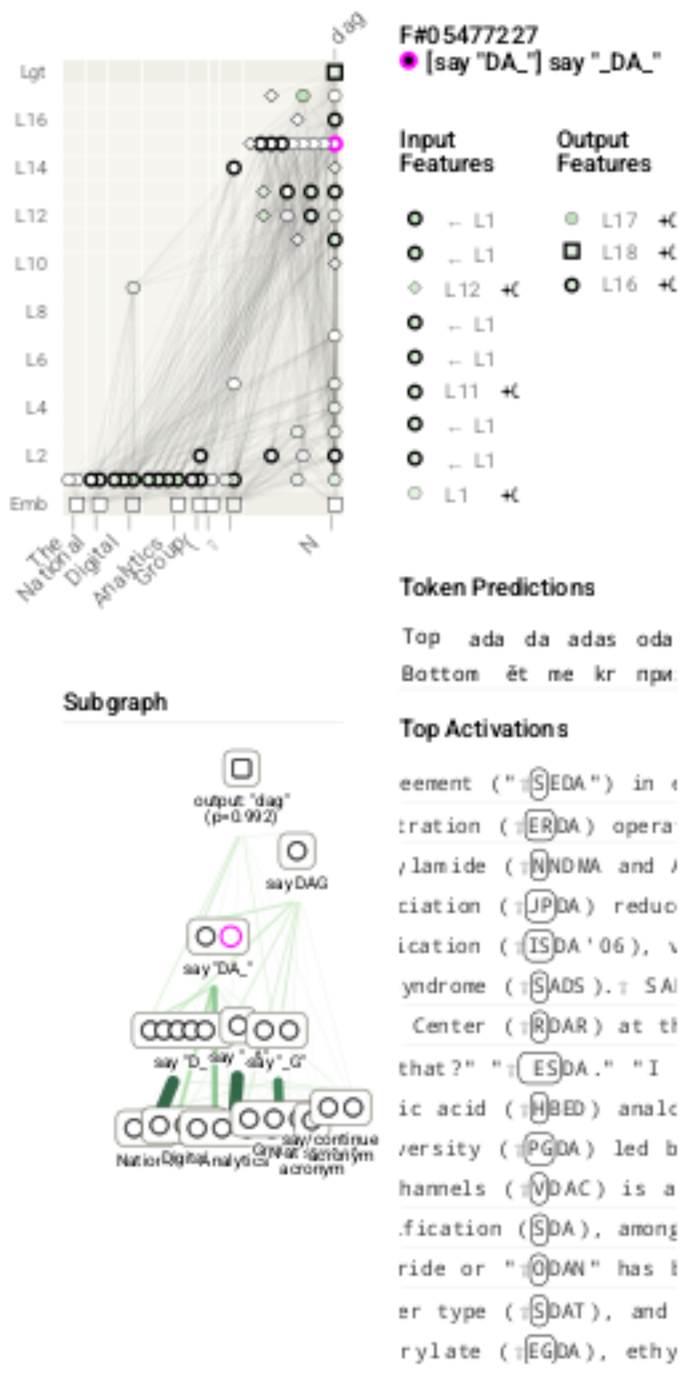

Full graph details
Selected node details
Embedding
Feature nodes appear at the token position where they are active
Diamond shaped error nodes represent reconstruction errors
Logit node
The strongest input features, colored by edge strength
Input Features
Token Predictions
Top Activations
Subgraph
Nodes can be dragged around manually
Nodes are grouped into supernodes
Strongest token embedding inputs and logit outputs
Top activ
Each row
Orange c
Selected and grouped subgraph

**Figure 6**: An overview of the interface for interacting with an attribution graph.

Below we show the interactive visualization for the attribution graph attributing back from the single token "DAG":



```
eement ("SEDA") in
tration (ERDA) opera
lamide (NNDMA and
ciation (JPDA) reduc
ication (ISDA'06),
yndrome (SADS). SA
 Center (RDAR) at th
that?" "ESDA." "I
ic acid (HBED) analc
versity (PGDA) led b
hannels (VDAC) is a
fication (SDA), among
ride or "ODAN" has
er type (SDAT), and
rylate (EGDA), ethy
```

The interface is interactive. Nodes can be hovered over and clicked on to display additional information. Subgraphs can also be constructed by using `Cmd/Ctrl+Click` to select a subset of nodes. In the subgraph, features can be aggregated into groups we call *supernodes* (motivated below in § 3.4 Grouping Features into Supernodes).

**Hover**
Hovering over a feature displays information about it

**Click**
One node is “actively selected” at a time (via clicking) and highlighted in pink

**Ctrl/Cmd-Click**
Pinned nodes are highlighted.
The pinned subgraph is displayed in a separate panel.

**Hold “**
Group
subgra

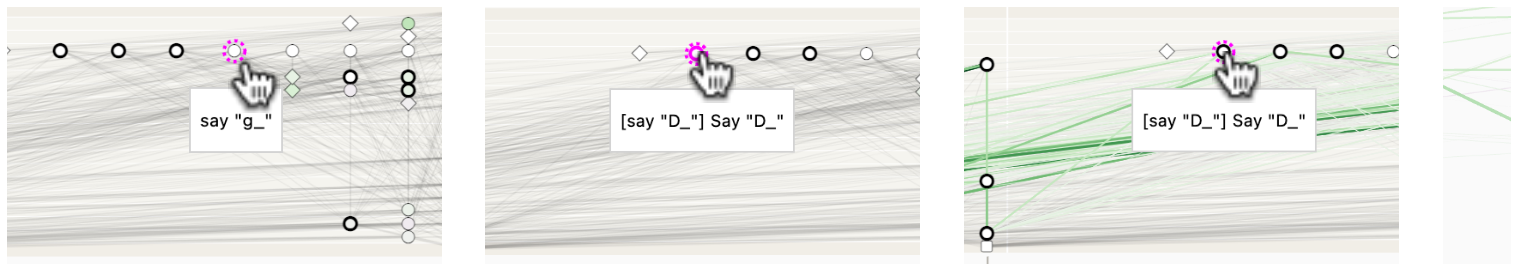


**Figure 7**: Supported mouse and keyboard interactions with attribution graphs.

## Understanding and Labeling Features

We use feature visualizations similar to those shown in our previous work, Scaling Monosemanticity, in order to manually interpret and label individual features in our graph.[12]

The easiest features to label are input features, which activate on specific tokens or categories of closely-related tokens and which are common in early layers, and output features, which promote continuing the response with specific tokens or categories of closely-related tokens and which are common in late layers. For example:

- This feature is likely an input feature because its visualization shows that it activates strongly on the word “digital” and similar words like “digitize”, but not on other words. We therefore label it a “digital” feature.
- This feature (a Haiku feature from the later § 3.8 Addition Case Study) is an input feature that activates on a variety of tokens that end in the digit 6, and even on tokens that are more abstractly related to 6 like “six” and “June”.
- This feature is likely an output feature because it activates strongly on several different tokens, but in each example, the token is followed by the text “dag”. Furthermore, the top of the visualization indicates that the feature increases the probability of the model predicting “dag” more than any other token (in terms of its direct effect through the residual stream). This suggests that it’s an output feature. Since output features are common, when labeling output features that promote some token or category X, we often simply write “say X”, so we give this example the label “say ‘dag’”.
- This feature (from the later § 3.7 Factual Recall Case Study) is an output feature that promotes a variety of sports, though it also demonstrates some ways in which labeling output features can be difficult. For example, one must observe that “lac” is the first token of “lacrosse”. Also, the next token in the context after the feature activates often isn’t actually the name of a sport, but is usually a plausible place for the name of a sport to go.

Other features, which are common in middle layers of the model, are more abstract and require more work to label. We may use examples of contexts they are active over, their *logit effects* (the tokens they directly promote and suppress through the residual stream and unembedding), and the features they’re connected to in order to label them. For example:

- This feature activates on the first one or two letters of an unfinished acronym after an open parenthesis, for a variety of letters and acronyms, so we label it as continuing an acronym in general.
- This feature activates at the start of a variety of acronyms that all have D as their *second* letter, and many of the tokens it promotes directly have D as their second letter as well. (Not all of those tokens do, but we don't expect logit effects to perfectly represent the feature's functionality because of indirect effects through the rest of the model. We also find that features further away from the last layer have less interpretable logit effects.) For brevity, we may label this feature as "say '_D'", representing the first letter with an underscore.
- Finally, this feature activates on the first letter of various strings of uppercase letters that don't seem to be acronyms, and the tokens it most suppresses are acronym-like letters, but its examples otherwise lack an obvious commonality, so we tentatively label it as suppressing acronyms.

We find that even imperfect labels for these features allow us to find significant structure in the graphs.

## Grouping Features into Supernodes

Attribution graphs often contain groups of features which share a facet relevant to their role on the prompt. For example, there are three features active on "Digital" in our prompt which each respond to the word "digital" in different cases and contexts. The only facet which matters for this prompt is that the word "digital" starts with a "D"; all three features have positive edges to the same set of downstream nodes. Thus for the purposes of analyzing this prompt, it makes sense to group these features together and treat them as a unit. For the purposes of visualization and analysis, we find it convenient to group multiple nodes — corresponding to (feature, context position) pairs — into a "supernode." These supernodes correspond to the boxes in the simplified schematic we showed above, reproduced below for convenience.

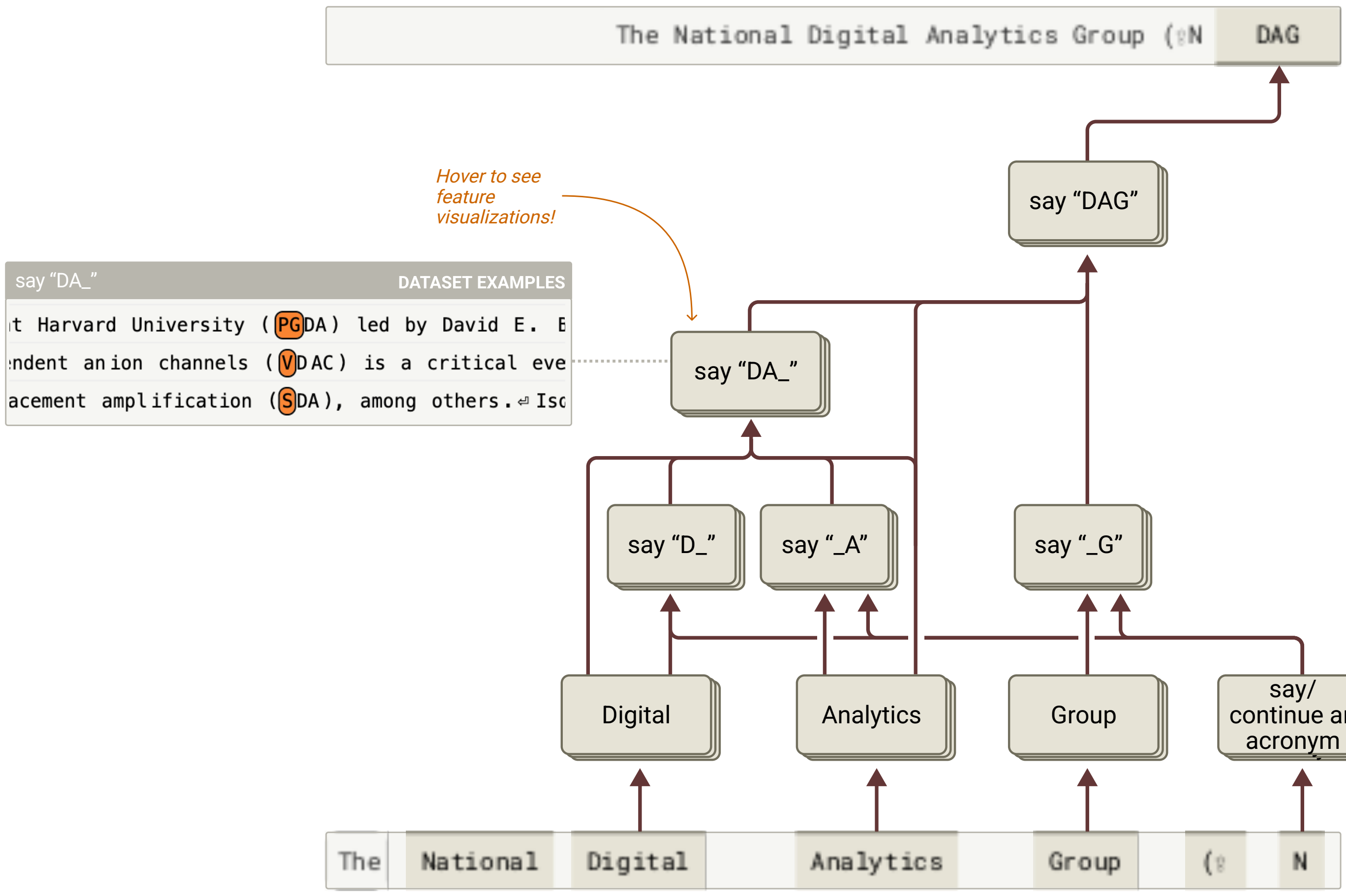

The National Digital Analytics Group (N
DAG
say "DAG"
Hover to see feature visualizations!
say "DA_"
DATASET EXAMPLES
t Harvard University (PGDA) led by David E. E
ndent anion channels (VDAC) is a critical eve
acement amplification (SDA), among others. Isc
say "DA_"
say "D_"
say "_A"
say "_G"
Digital
Analytics
Group
say/ continue an acronym
The
National
Digital
Analytics
Group
(
N

**Figure 8**: A simplified diagram of the attribution graph for 18L completing a fictional acronym.

The strategy we use to group nodes depends on the analysis at hand, and on the roles of the features in a given prompt. We sometimes group features which activate over similar contexts, have similar embedding or logit effects, or have similar input/output edges, depending on the facet which is important for the claim we are making about the mechanism. We generally want nodes within a supernode to promote each other, and their effects on downstream nodes to have the same sign. While we experimented with automated strategies such as clustering based on decoder vectors or the graph adjacency matrix, no automated method was sufficient to cover the range of feature groupings required to illustrate certain mechanistic claims. We further discuss supernodes and potential reasons for why they are needed in Similar Features and Supernodes.

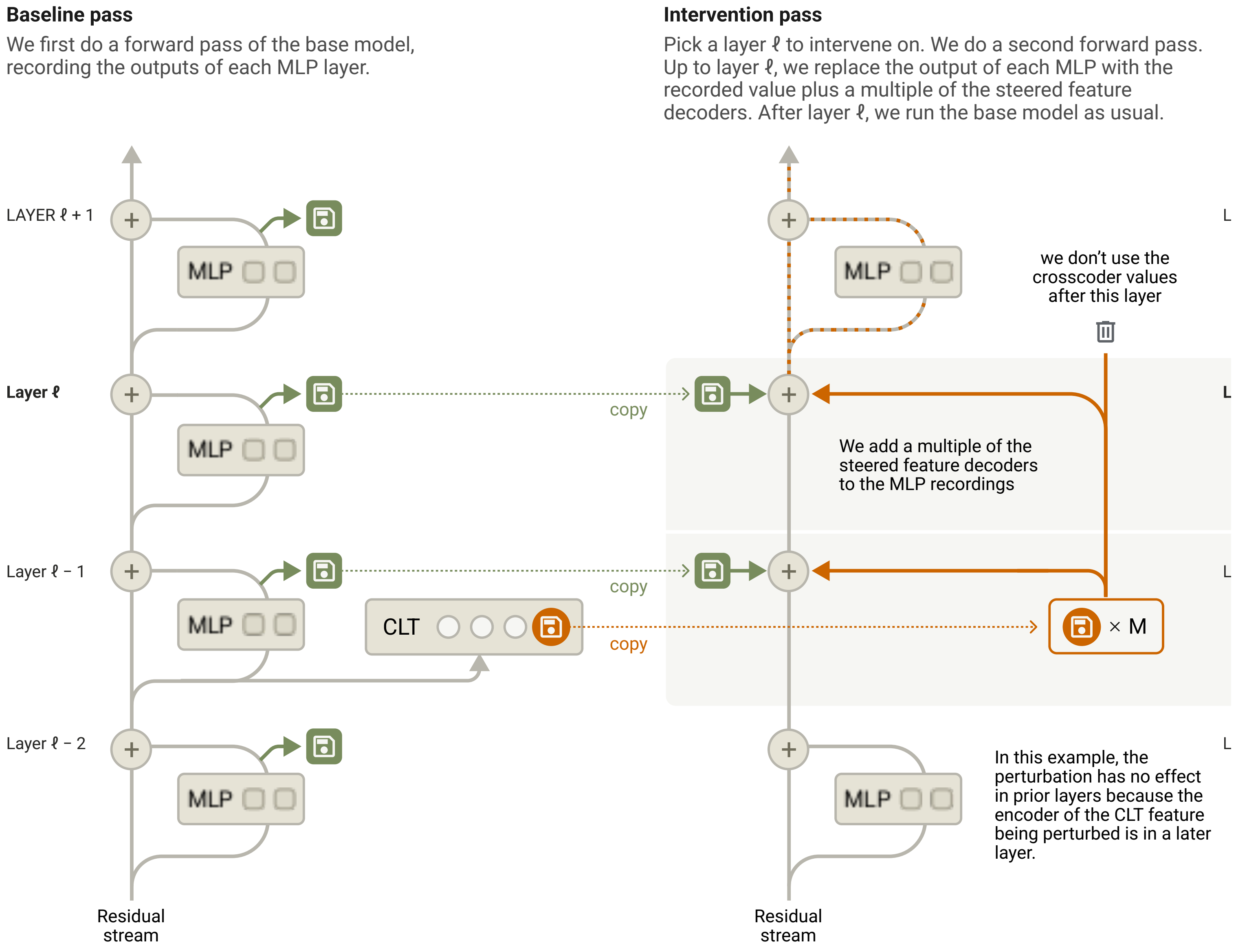
## Validating Attribution Graph Hypotheses with Interventions

In attribution graphs, nodes suggest which features matter for a model's output, and edges suggest how they matter. We can validate the claims of an attribution graph by performing feature perturbations in the underlying model, and checking if the effects on downstream features or on the model outputs match our predictions based on the graph. Features can be intervened on by modifying their computed activation and injecting its modified decoding in lieu of the original reconstruction.

Features in a cross-layer transcoder write to multiple output layers, so we need to decide on a range of layers in which to perform our intervention. How might we do this? We could intervene on a feature's decoding at a single layer just like we would for a per-layer transcoder, but edges in an attribution graph represent the cumulative effect of multiple layers' decodings, so intervening at a single layer would only target a subset of a given edge. In addition, we'll often want to intervene on more than one feature at a time, and different features in a supernode will decode to different layers.

To perform interventions over layer ranges, we modify the decoding of a feature at each layer in the given range, and run a forward pass starting from the last layer in the range. Since we aren't recomputing a layer's MLP output based on the result of interventions earlier in the range, the only change to the model's MLP outputs will be our intervention. We call this approach "constrained patching", as it doesn't allow an intervention to have second-order effects within its patching range. See § K Appendix: Iterative Patching for a description of another approach we call "iterative patching", and see § H Appendix: Nuances of Steering with Cross-Layer Features for a discussion of why more naive approaches, such as adding a feature's decoder vector at each layer during a forward pass of the model, risk double counting a feature's effect.

Below, we illustrate a multiplicative version of constrained patching, in which we multiply a target feature's activation by $M$ in the $[\ell - 1, \ell]$ layer range. Note that MLP outputs at further layers are not directly affected by the patch.[13]



Baseline pass
We first do a forward pass of the base model, recording the outputs of each MLP layer.
Intervention pass
Pick a layer ℓ to intervene on. We do a second forward pass. Up to layer ℓ, we replace the output of each MLP with the recorded value plus a multiple of the steered feature decoders. After layer ℓ, we run the base model as usual.
LAYER ℓ + 1
Layer ℓ
Layer ℓ – 1
Layer ℓ – 2
MLP
CLT
copy
× M
we don't use the crosscoder values after this layer
We add a multiple of the steered feature decoders to the MLP recordings
In this example, the perturbation has no effect in prior layers because the encoder of the CLT feature being perturbed is in a later layer.
Residual stream

**Figure 9**: A schematic of multiplicative patching.

Attribution graphs are constructed by using the underlying model's attention patterns, so edges in the graph do not account for effects mediated via QK circuits. Similarly, in our perturbation experiments, we keep attention patterns fixed at the values observed during an unperturbed forward pass. This methodological choice means our results don't account for how perturbations might have altered the attention patterns themselves.

Returning to our acronym prompt, we show the results of patching supernodes, starting with suppressing the "Group" supernode. Below, we overlay patching effects onto supernode schematics for clarity, displaying the effect on other supernodes and the logit distribution. Note that in this diagram, the position of the nodes in the figure is not meant to correspond to token positions unless explicitly noted.

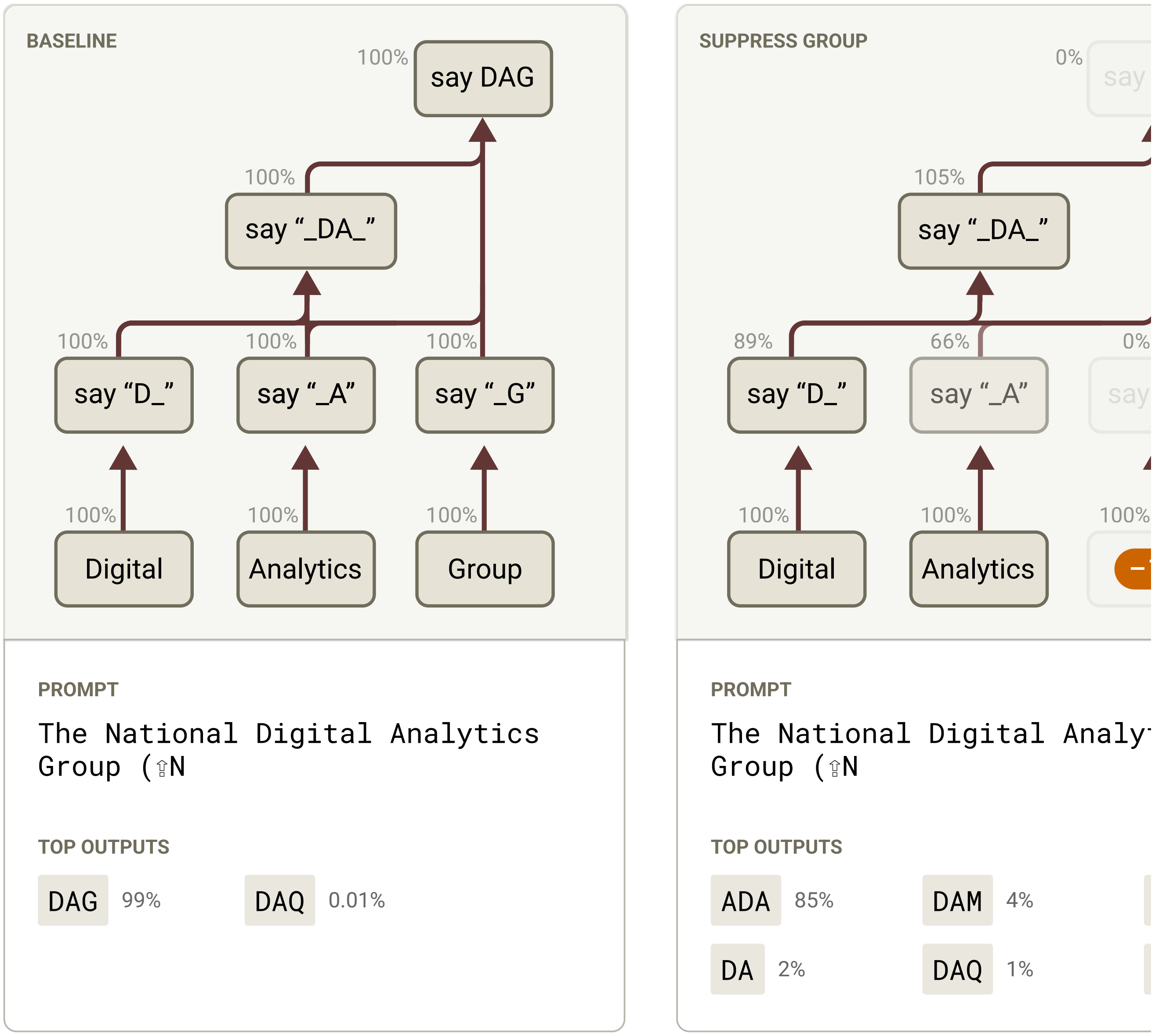


**Figure 10**: Suppressing the word "Group" in the fictional organization's name causes 18L to output other acronyms with "D

We now show the results of suppressing some supernodes on the aggregate activation of other supernodes and on the logit. For each patch, we set every feature in a node's activation to be the opposite of its original value (or equivalently, we steer multiplicatively with a factor of −1).[14] We then plot each node's total activation as a fraction of its original value.[15] We use an orange outline to highlight nodes downstream of one another for which we would hypothesize patching to have an effect.

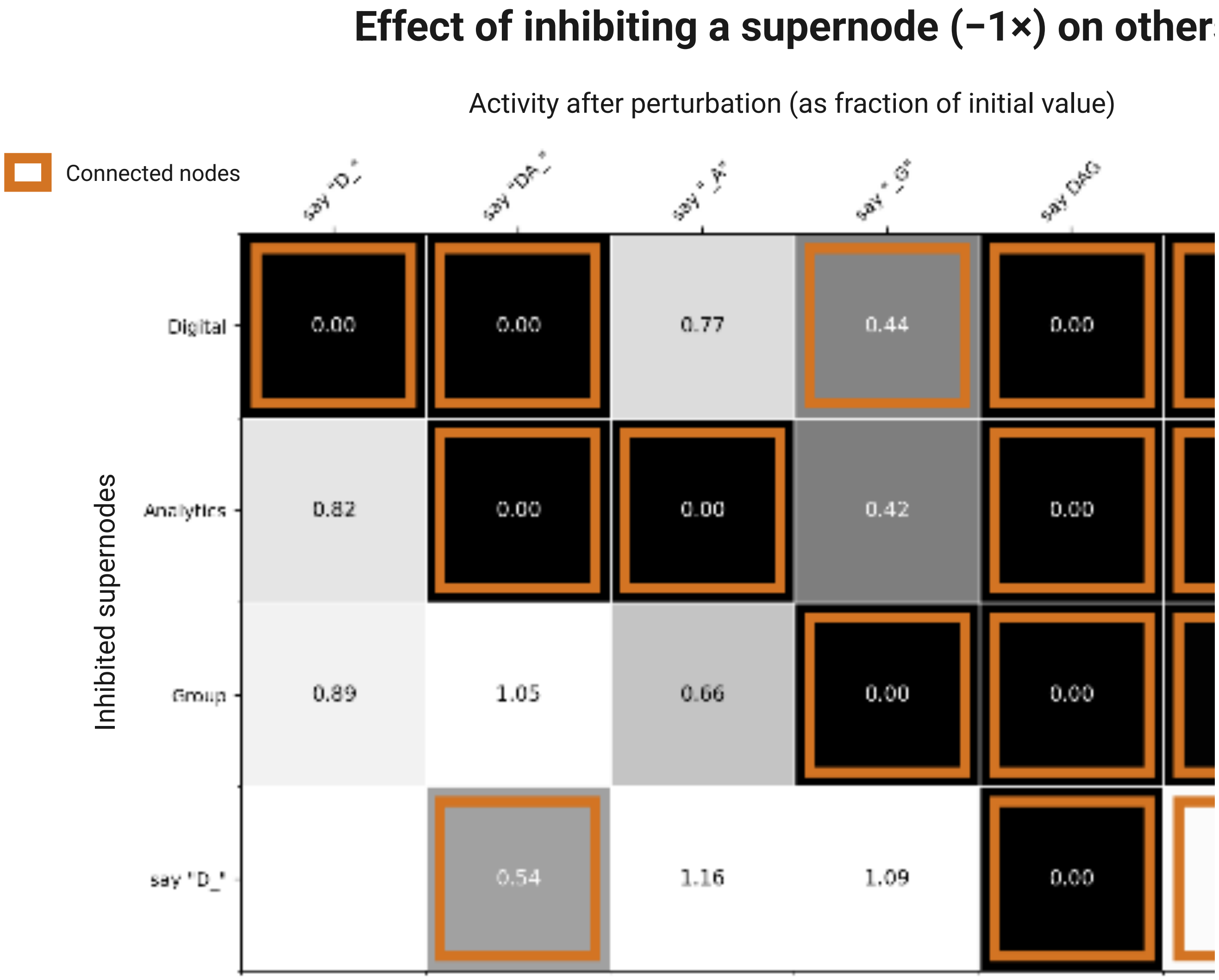
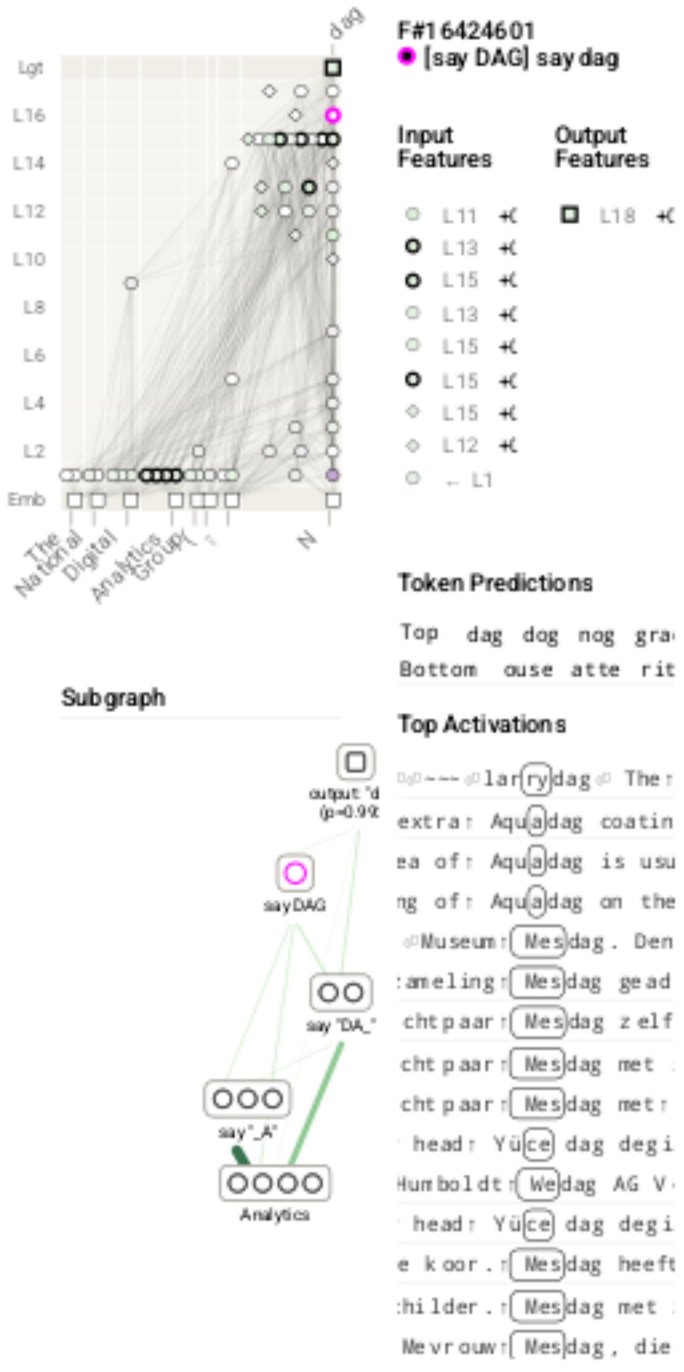


**Figure 11**: Selected intervention effects on feature activations for “The National Digital Analytics Group (N”.

We see that inhibiting features for each word inhibits the related initial features in turn. In addition, the supernode of features for “say DA_” is affected by inhibitions of both the “Digital” and “Analytics” supernodes.

## Localizing Important Layers

The attribution graph also allows us to identify in which layers a feature’s decoding will have the greatest downstream effect on the logit. For example, the “Analytics” supernode features mostly contribute to the “dag” logit indirectly through intermediate groups of features “say _A”, “say DA_”, and “say DAG” which live in layers 13 and beyond.



We would thus expect steering negatively on an “Analytics” feature to have an effect on the dag logit which plateaus before layer 13 and then decreases in magnitude as we approach the final layer. The decrease is caused by the constrained nature of our intervention. If a patching range includes all the “say an acronym” features, it will not change their activation, because constrained patching doesn’t allow knock-on effects. Below, we show the effect of steering with each Analytics feature, keeping the start layer set to 1 and sweeping over the patching end layer.[16]

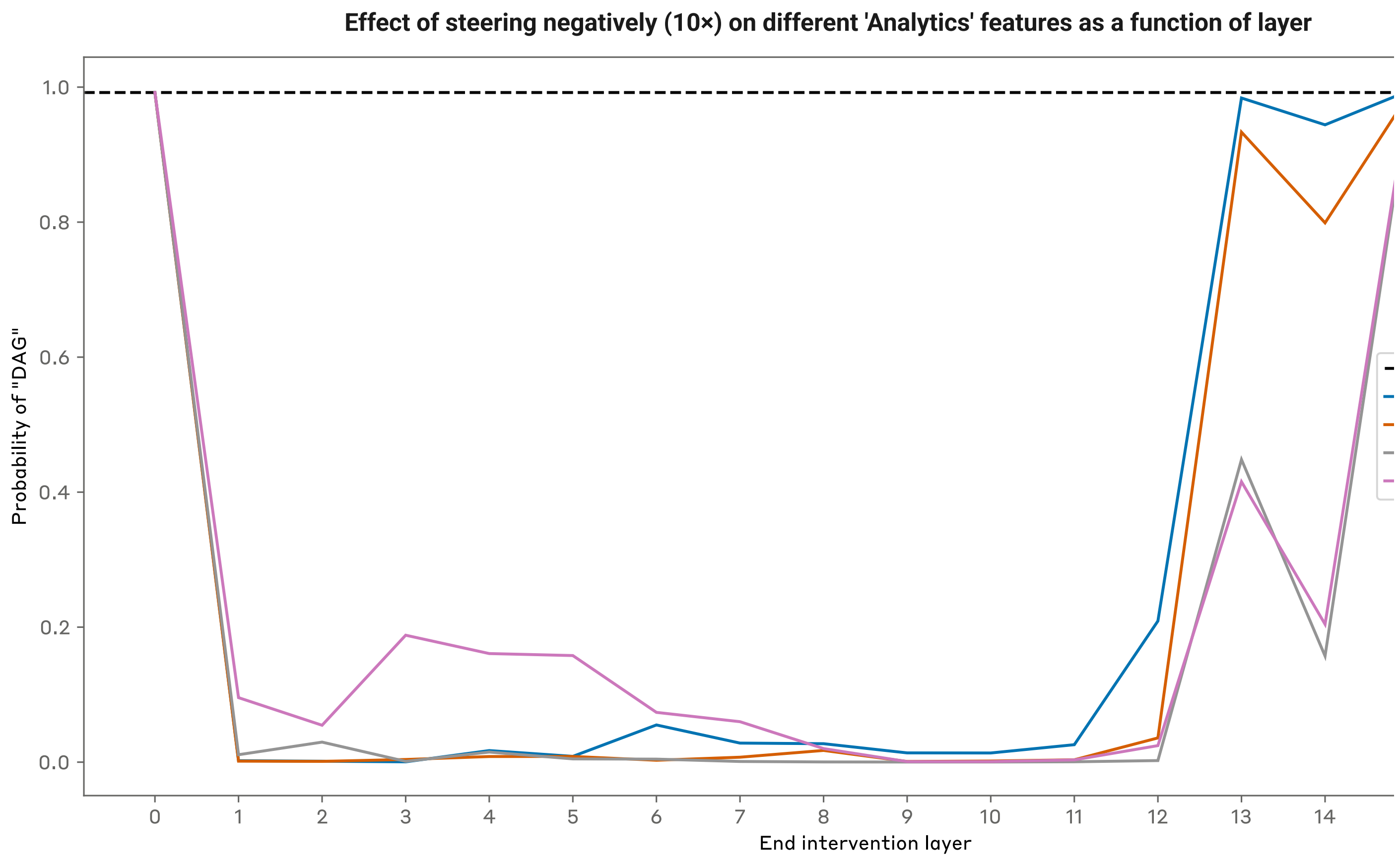
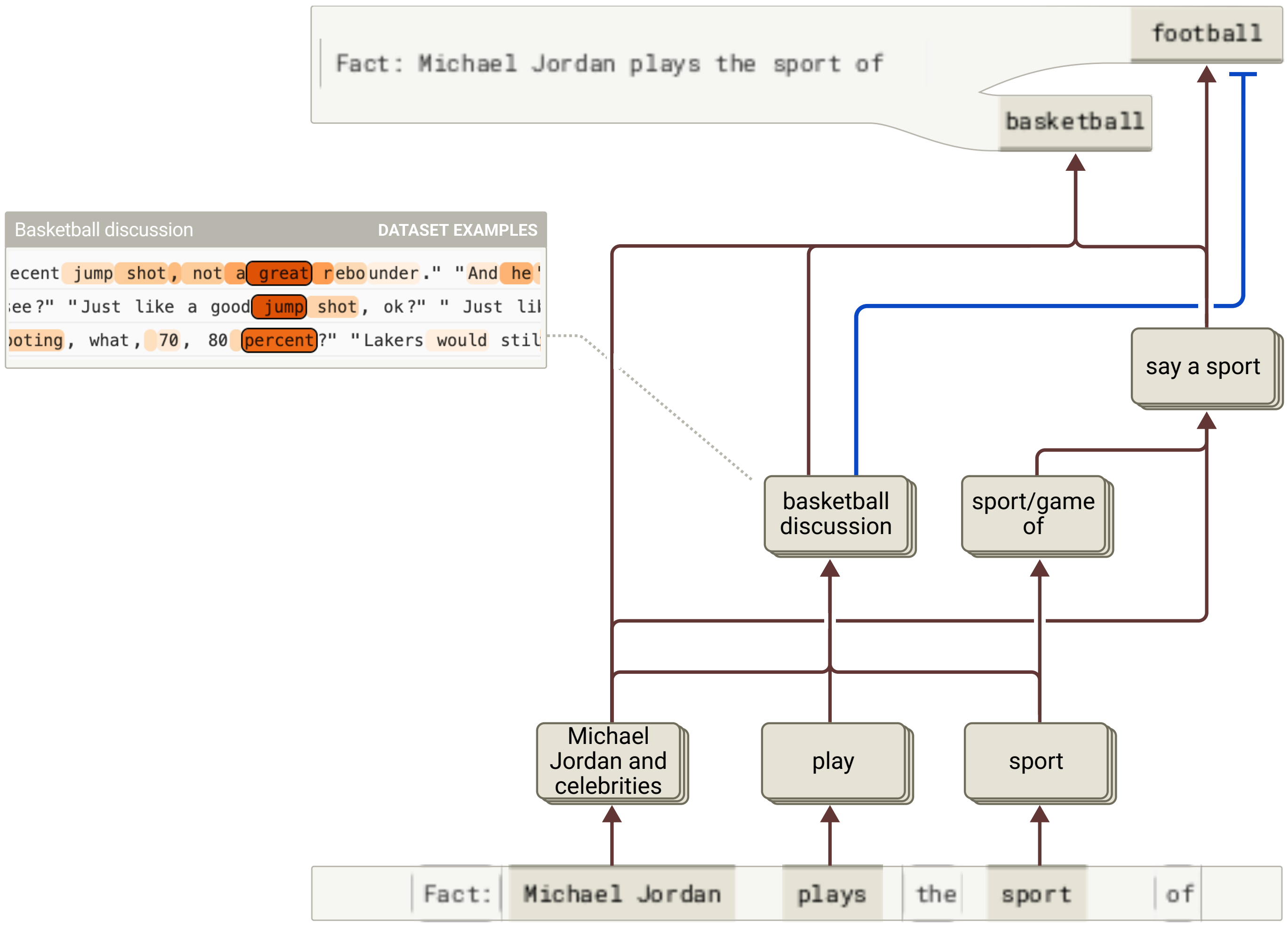


**Figure 12**: Interventions are most effective when done in layers prior to those containing the “say an acronym” features.

## Factual Recall Case Study

We now turn to the question of factual recall by studying how the model completes `Fact: Michael Jordan plays the sport of` with `basketball` with 65% confidence [19, 20]. We start by computing an attribution graph. We group semantically similar features into supernodes like we did for the acronym study.

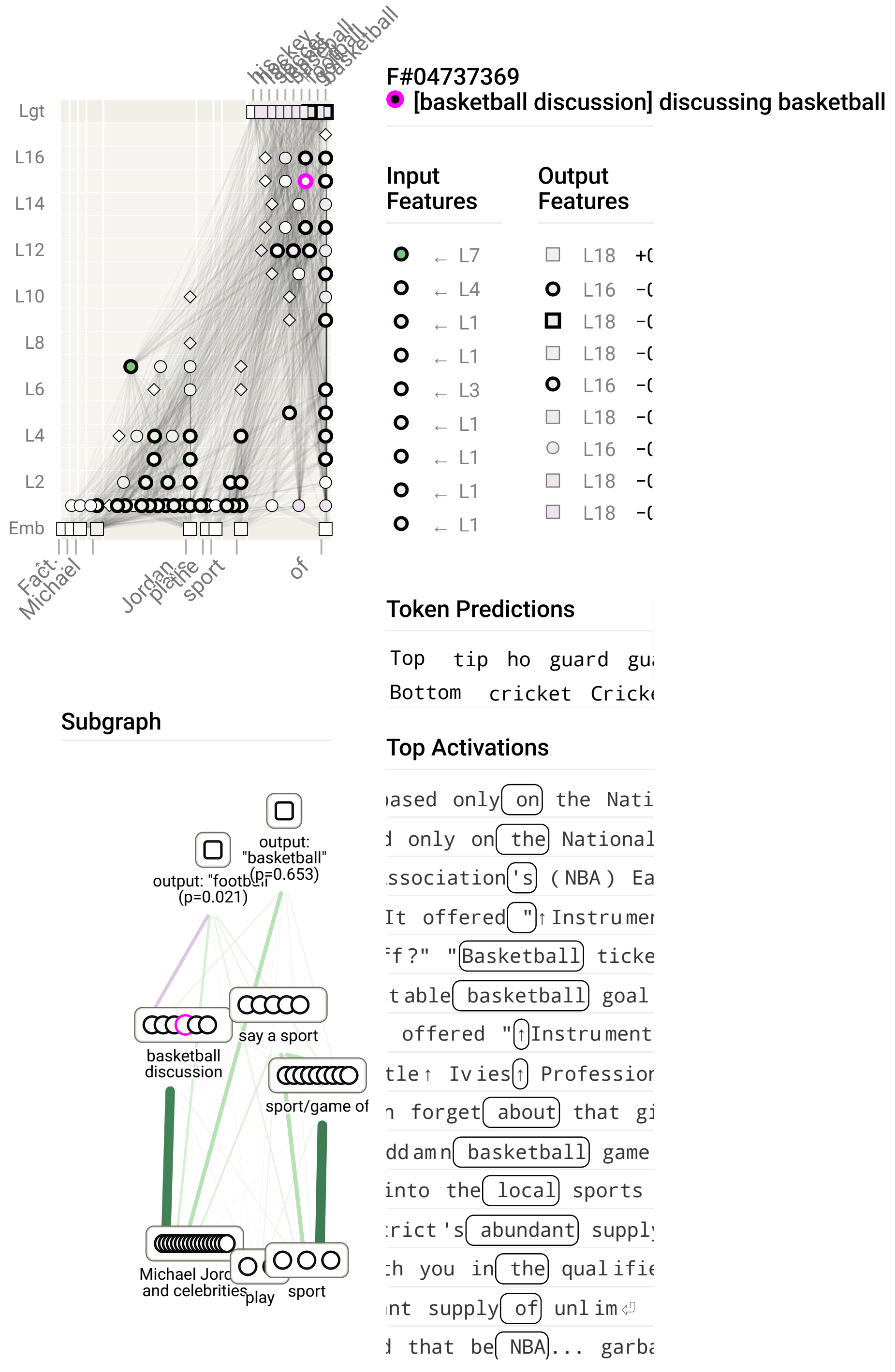
The supernode diagram below shows two primary paths. One path originates from the “plays” and “sport” tokens and promotes “sport” and “say a sport” features, which in turn promote the logits for basketball, football, and other sports. The other path originates from “Michael Jordan and other celebrities” and promotes basketball related features, which have positive edges to the basketball logit and negative edges to the football logit. In addition to these sequential paths, some groups of features such as “Michael Jordan” and “sport/game of” have direct edges to the basketball logit, representing effects mediated only via attention head OVs, consistent with the findings of Batson *et al.* [20].



Fact: Michael Jordan plays the sport of
football
basketball
Basketball discussion
DATASET EXAMPLES
ecent jump shot, not a great rebounder." "And he'
ee?" "Just like a good jump shot, ok?" " Just lik
oting, what, 70, 80 percent?" "Lakers would stil
say a sport
basketball discussion
sport/game of
Michael Jordan and celebrities
play
sport
Fact: Michael Jordan plays the sport of

**Figure 13**: A simplified diagram of the attribution graph for 18L recalling a simple fact.

We also display the full interactive graph below.



F#04737369

● [basketball discussion] discussing basketball

| Input Features | | Output Features | | |
|---|---|---|---|---|
| ● | ← L7 | ☐ | L18 | +( |
| ○ | ← L4 | ○ | L16 | -( |
| ○ | ← L1 | ☐ | L18 | -( |
| ○ | ← L1 | ☐ | L18 | -( |
| ○ | ← L3 | ○ | L16 | -( |
| ○ | ← L1 | ☐ | L18 | -( |
| ○ | ← L1 | ○ | L16 | -( |
| ○ | ← L1 | ☐ | L18 | -( |
| ○ | ← L1 | ☐ | L18 | -( |

Token Predictions

| | | | | |
|---|---|---|---|---|
| Top | tip | ho | guard | gu |
| Bottom | cricket | Crick | | |

Top Activations

```
ased only[ on] the Nati
d only on[ the] National
ssociation['s] ( NBA ) Ea
It offered[ "]↑ Instrumen
ff ?" "[Basketball] ticke
t able[ basketball] goal
 offered "[↑]Instrument
tle↑ Iv ies[↑] Professior
n forget[ about] that gi
dd am n[ basketball] game
into the[ local] sports
trict 's[ abundant] suppl
ch you in[ the] qual ifie
nt supply[ of] unl im⏎
d that be[ NBA]... garba
```

In addition, a complex set of mechanisms seems to be involved in contributing information about the entity Michael Jordan to the residual stream at “Jordan”, as observed in Nanda *et al.* [19]. We have grouped into one supernode features sensitive to "Michael", an L1 feature which has already identified the token pair “Michael Jordan”, features for other celebrities, and polysemantic features firing on “Michael Jordan” and other unrelated concepts. Note that we choose to include some polysemantic features in supernodes as long as they share a facet relevant to the prompt, such as this feature which activates more strongly on the word “synergy” than on “Michael Jordan”. We evaluate features in more depth in Qualitative Feature Evaluations.

Steering experiments can once more allow us to validate the hypotheses proposed by the graph.

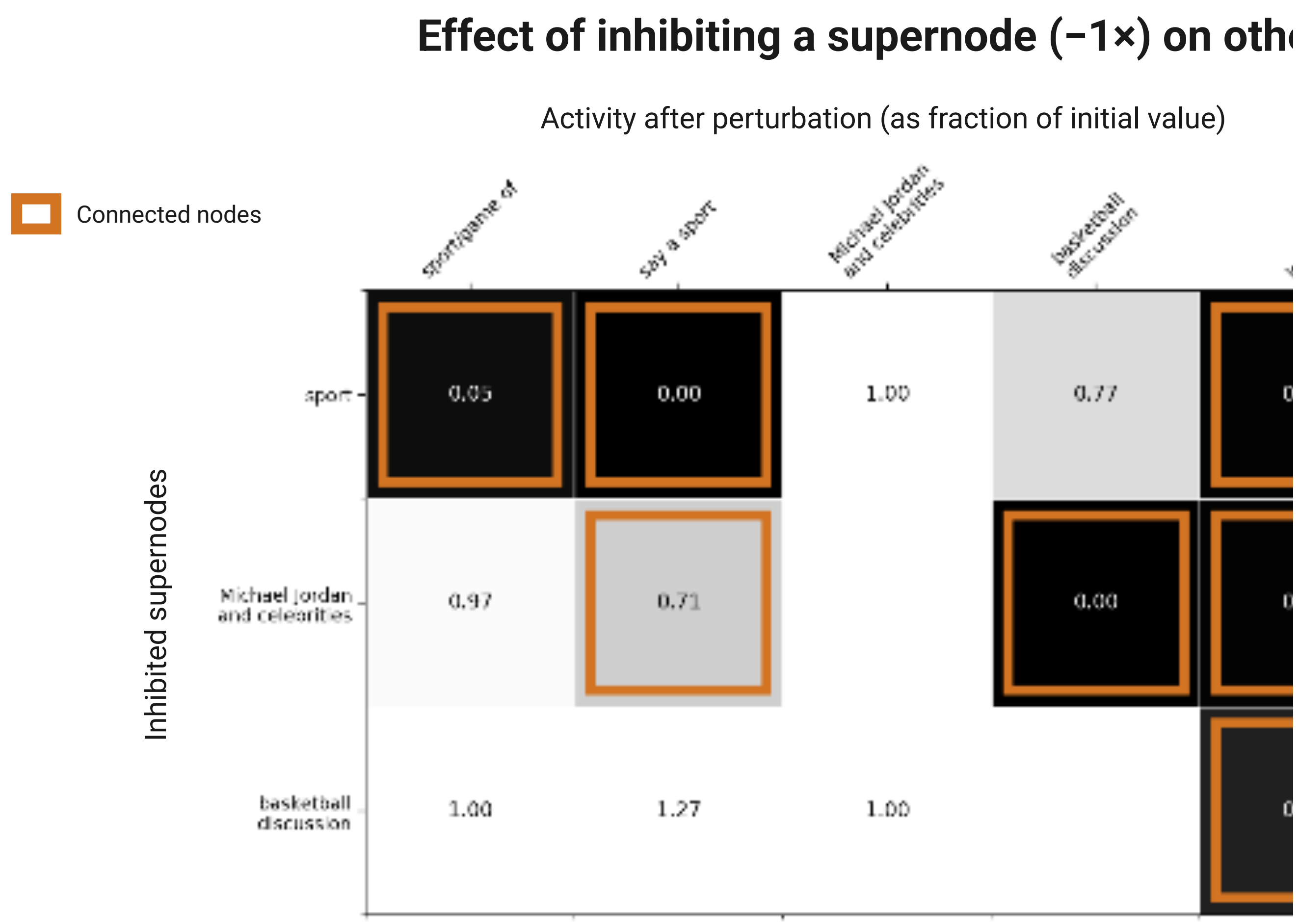


**Figure 14**: Selected intervention effects on feature activations for “Fact: Michael Jordan plays the sport of”.

Ablating either the “sport” or “Michael Jordan” supernode has a large effect on the logit but a comparatively smaller effect on the other supernode, confirming the parallel path structure. In addition, we see that suppressing the intermediate “basketball discussion” supernode also has a large effect on the logit.

## Addition Case Study

We now consider the simple addition prompt `calc: 36+59=`.[17] Unlike previous sections, we show results for Haiku 3.5 because the patterns are clearer and show the same structure (see § Q Appendix: Comparison of Addition Features … for a side-by-side comparison). We look at small-number addition because it is one of the simplest behaviors exhibited competently by most LLMs and human adults (try the problem in your head to see if your approach matches the model's!).

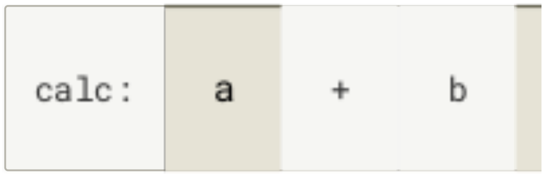
We supplement the generic feature visualization (on arbitrary dataset examples) with one which explicitly covers the set of two-digit addition problems, allowing us to get a crisp picture of what each feature does. Following Nikankin *et. al.* [21], who analyzed neurons, we visualize each feature active on the `=` token with three plots:

- An operand plot, displaying its activity on the 100 × 100 grid of potential inputs.
- An output weight plot, displaying its direct weights on the outputs for [0, 99].[18]
- An embedding weight plot (or “de-embedding” [12]), displaying the direct effect of embedding vectors on a feature’s encoder. This is shown in the same format as the output weight plot.

We show an example plot of each of these three types below for different features. On this restricted domain, the operand plots are complete descriptions of the CLT features as functions. Stripes and grids in these plots represent different kinds of structure (e.g. diagonal lines indicate constraints on the sum, while grids represent modular constraints on the inputs).

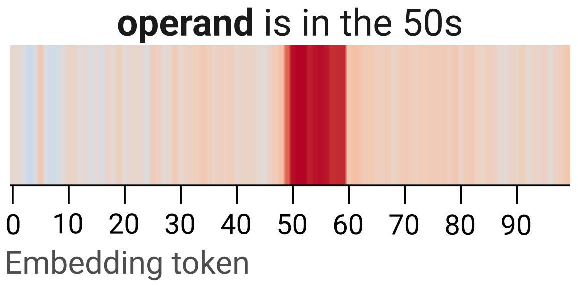
### Addition Features

We systematically analyze the features active on one- and two-digit addition prompts of the form **calc: a+b=** for **a**, **b** ∈ [0,99].



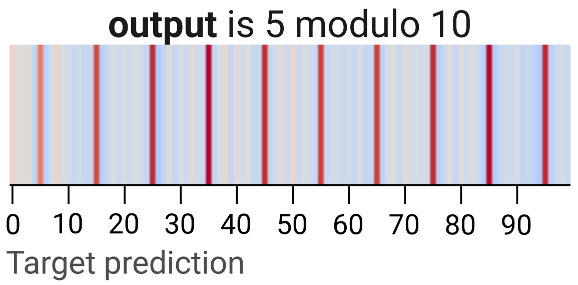
### Embedding Weight Plots

The direct effect of embeddings for tokens 0–99 on the feature encoder. Visualized for features active on **a** or **b**.



### Output Weight Plots

The direct effect of the sum of a feature's decoders on the outputs for tokens 0–99. Visualized for features active on **=**.



### Operand Plots

The activity of a feature on the "=" token of the prompt "calc: a+b=", for a,b ∈ [0,99]

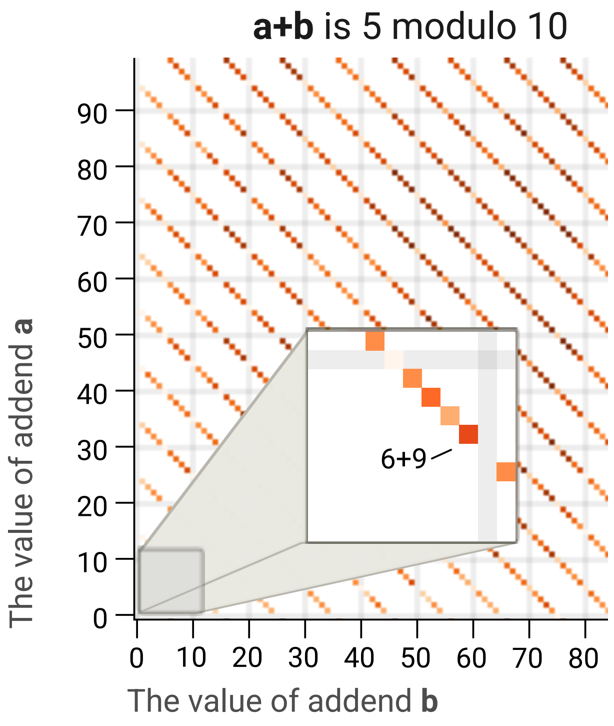


**Figure 15**: Common elements in figures describing addition features.

In the supernode diagram below, we see information flow from input features, which split out the final digit, the number, and the magnitude of the operands to three major paths: a final-digit path (mod 10) (light-brown, right), a moderate precision path (middle), and a low precision path (dark brown, left),[19] which collectively produce a moderate precision value of the sum and the final digit of the sum; these finally constructively interfere to give both the mod 100 version of the sum and the final output.

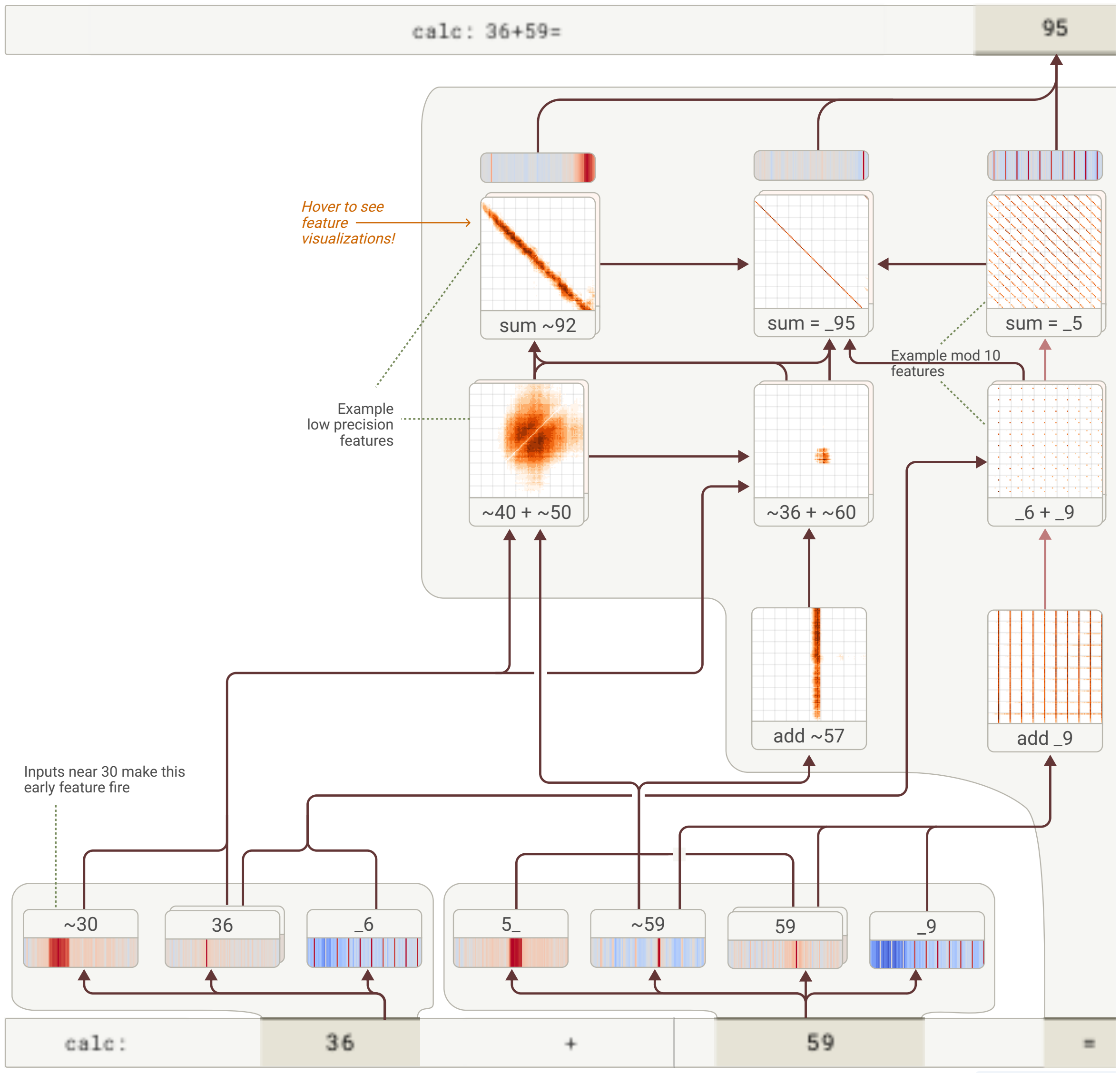


**Figure 16**: A simplified attribution graph of Haiku adding two-digit numbers. Features of the inputs feed into separable processing pathways.

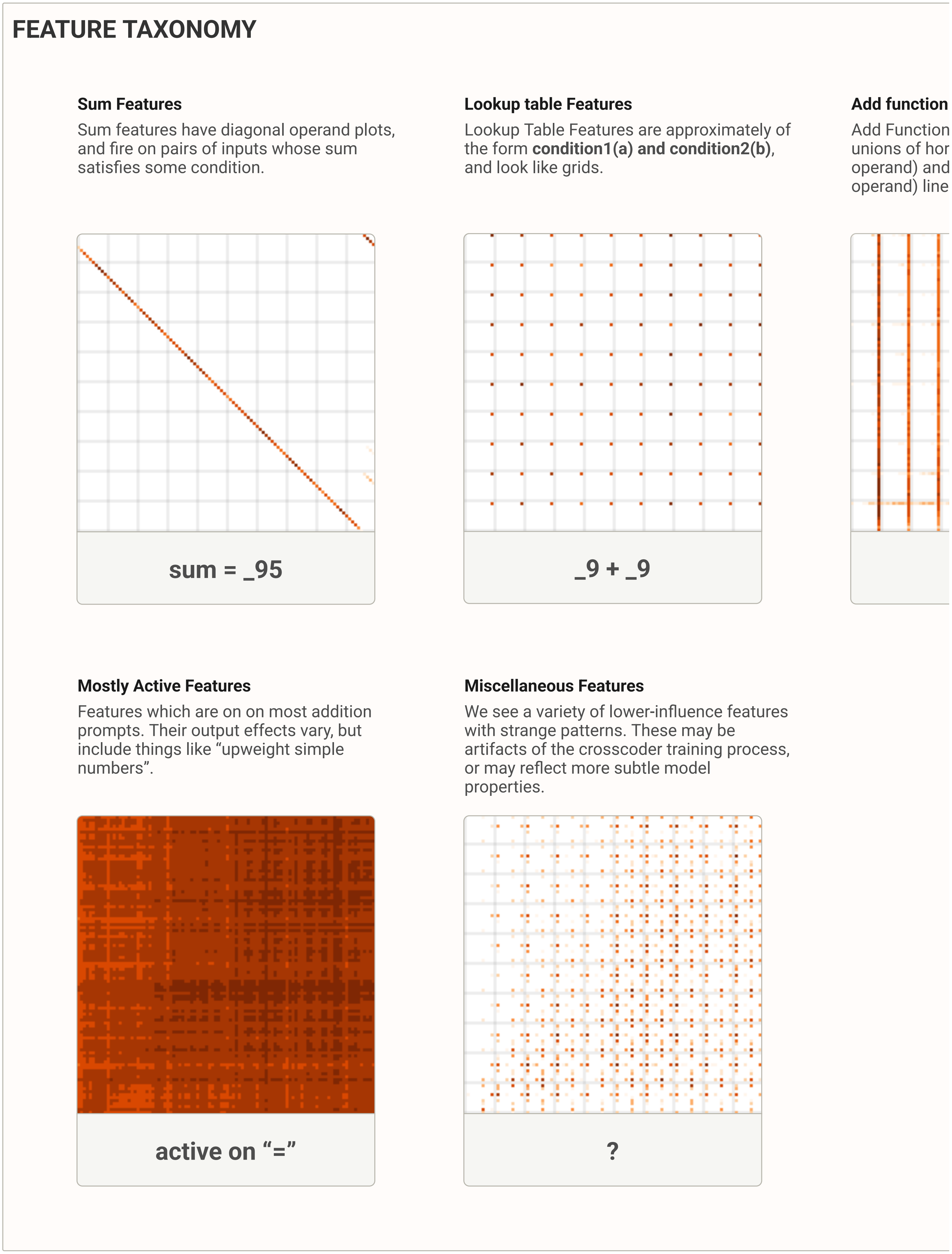
View detailed grap

We provide the equivalent interactive graph for 18L here.

The supernode graph suggests a taxonomy of features underlying this task, in which features vary along two major axes:[20]

- **Computational role**
    - **Sum features** have diagonal operand plots, and fire on pairs of inputs whose sum satisfies some condition.
    - **Lookup Table Features** have plots that look like a grid, and consist of inputs a and b satisfying `condition1(a) AND condition2(b)`. We discuss these in more detail below.
    - **Add Function Features** have plots with horizontal or vertical bars. One addend satisfies some condition, or an `OR` operation merges two conditions across addends.
    - **Mostly Active Features** are active on the "=" token of most of our 10,000 addition prompts.
    - **Miscellaneous Features** have all sorts of strange properties, but often look like hybrid activation patterns from the above types. We find that these have lower influence on the outputs.
- **Condition properties**

- **Precision**: we find conditions with ones-digit precision (sum=_5 or =59), with exact range (of width e.g. 2 or 10), and with fuzzy ranges of width ranging from 2–50.
- **Modularity**: we find features that are sensitive to the sum or operand value in absolute terms, mod 10, mod 100, and less commonly, mod 2, mod 5, mod 25, and mod 50.
- **Pattern**: we find features sensitive to a regex style pattern in an input or output, such as “starts with 51”, as in [21]. These do not feature as prominently in our addition graphs, having low influence on the model's output, but they do exist, and may be more important for other tasks involving numbers.



FEATURE TAXONOMY

**Sum Features**

Sum features have diagonal operand plots, and fire on pairs of inputs whose sum satisfies some condition.

**Lookup table Features**

Lookup Table Features are approximately of the form **condition1(a) and condition2(b)**, and look like grids.

**Add function**

Add Function
unions of hor
operand) and
operand) line

**Mostly Active Features**

Features which are on on most addition prompts. Their output effects vary, but include things like "upweight simple numbers".

**Miscellaneous Features**

We see a variety of lower-influence features with strange patterns. These may be artifacts of the crosscoder training process, or may reflect more subtle model properties.

**Figure 17**: Types of features commonly active on the "=" within prompts of the form "calc: **a**+**b**=" for **a**, **b** in [0,99].

These findings broadly agree with other mechanistic studies showing that language models trained on natural language corpora perform addition using parallel heuristics involving magnitudes and moduli that constructively interfere to produce the correct answer [21, 22, 23]. Namely, Nikankin et al. [21] proposed a "bag of heuristics" interpretation, recognizing a set of "operand" features (equivalent to our "add X" features) and "result" features (equivalent to our "sum" features) exhibiting high- and low-precision and different modularities in sensing the input and producing the output.

We also identify the existence of lookup table features, which seem to be an interesting consequence of the architecture used by both the model and the CLT. Neurons and CLT feature activations are computed by applying a nonlinearity to the sum of their inputs. This produces a "parallelogram constraint" on the response of a feature to a set of inputs: namely, if a feature $f$ is active on two inputs of the form $x+y$ and $z+w$, then it must be active on at least one of the inputs $x+w$ or $z+y$. This follows since the preactivation of $f$ is an affine function of the operands.[21] In particular, it is impossible for input features to produce a general sum feature in one step. For example, a general "sum = 5" feature which fires for 1+4 and 2+3 would need to fire for at least one of 1+3 or 2+4. So some intermediate step between copying over information about both inputs to the "=" token and producing a property of their sum is required. CLT lookup table features represent these intermediate steps for addition.[22]

To validate that the structure we observe in the attribution graph matches the causal structure of the model, we perform a series of interventions. For each supernode, we perturb it to the negative of its original value, and measure the result on all subsequent supernodes and the outputs. We find results largely consistent with the graph:

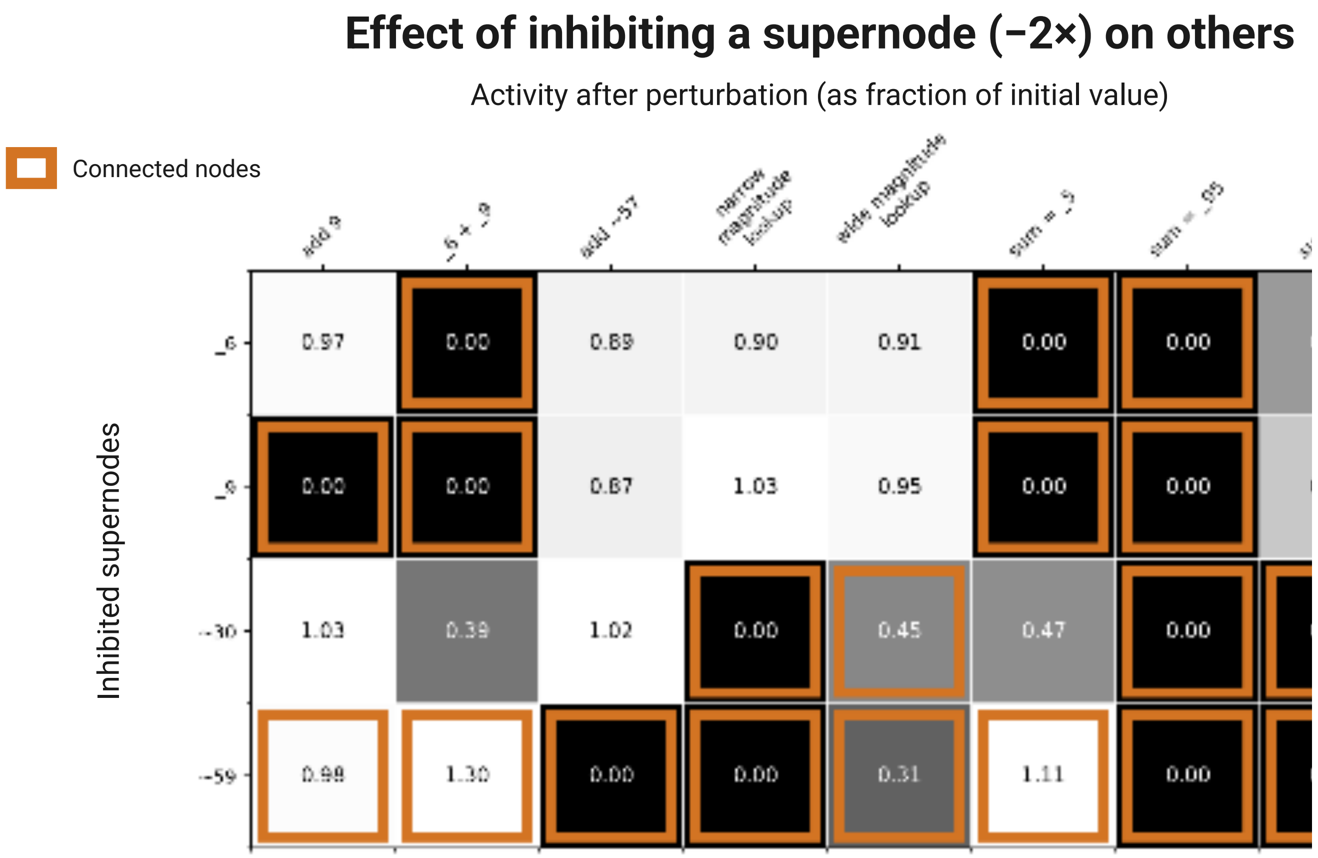


**Figure 18**: Selected intervention effects on feature activations for "calc: 36+59="

In particular, inhibiting the ones-digit feature on either of the input tokens suppresses the entire ones-digit pathway (the _6 + _9 lookup table features, the resulting sum=_5 and sum= _95 features), while leaving the magnitude pathway mostly intact, including the sum~92 features. Remarkably, when suppressing _6, the model confidently outputs 98 instead of the correct answer 95; the tens digit from the original problem is preserved by the other magnitude signals but the ones digit is that which would result from adding 9 to itself. (Suppressing _9, however, results in an output of 91, not 92, so such numerology must be taken with a grain of salt.). Conversely, inhibiting low-precision features on either inputs (~30 and ~59) suppress the low-precision lookup table features, the magnitude sum feature, and the appropriate sum features while leaving the ones-digit pathway alone.

We also show the quantitative effects of perturbations on the outputs, finding that negatively steering the _6 + _9 lookup table features smears the result out over a range of 5, while negatively steering the final sum=_95 feature smears the result out to a wider band (perhaps coming from sum~92 features).

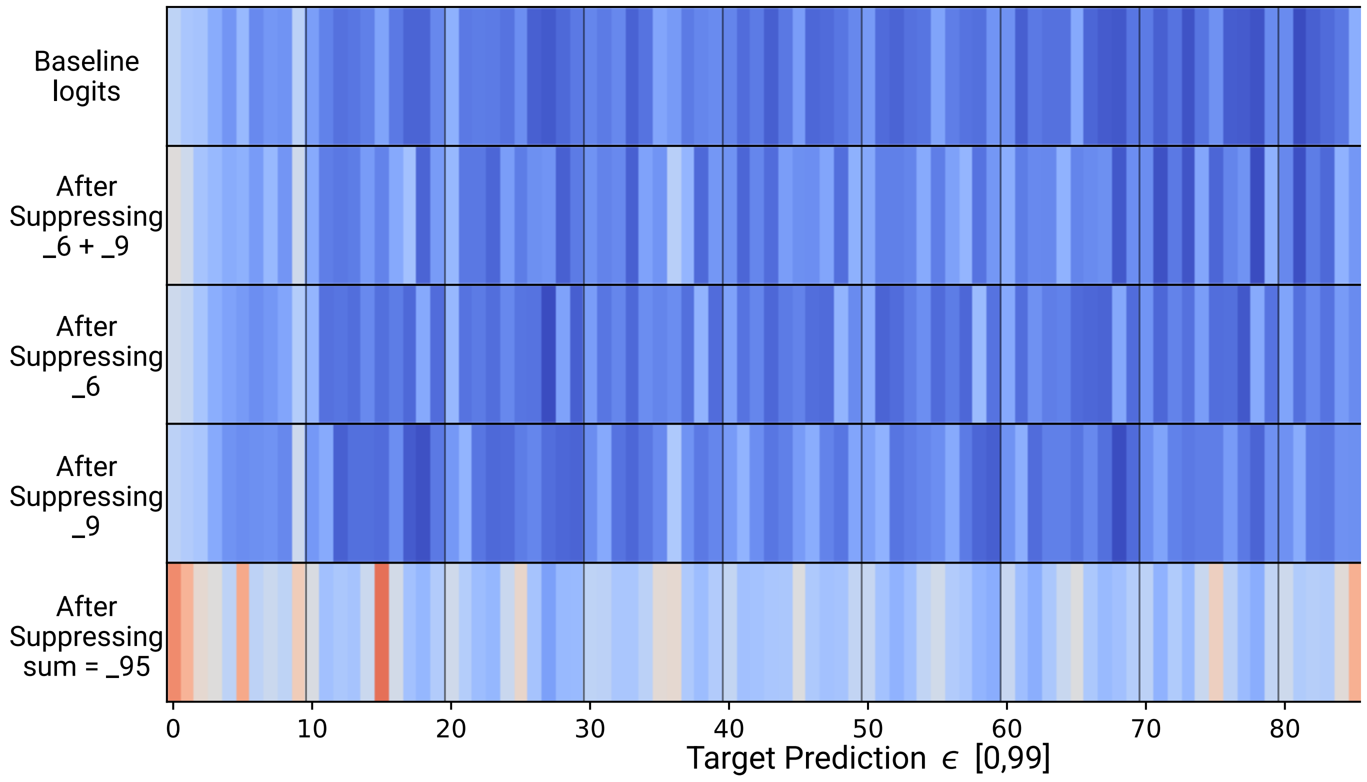


**Figure 19**: Target prediction logits for different interventions on “calc: 36+59=”.

We will investigate how CLT features interact across the full range of two-digit addition prompts below, after establishing the framework for global weights that we use to generalize this circuit to other inputs.

## Global Weights

The attribution graphs we construct show how features interact on a specific prompt to produce the model's output, but we are also interested in a more global picture of how features interact across all contexts. In a classic multi-layer perceptron, the global interactions are provided by the weights of the model: the direct influence of one neuron on another is just the weight between them if the neurons are in consecutive layers; if neurons are further apart, the influence of one on another factors through intermediate layers. In our setup, the interaction between features has a context *independent* component and a context *dependent* component. We would ideally like to capture both: we want a set of *global weights* which are context independent, but also capture network behavior across all possible contexts. In this section we analyze the context independent component (a kind of "virtual weight"), a problem with them (large "interference" terms with no causal effect on distribution), and one approach using co-activation statistics to deal with the interference.

On a specific prompt, a source CLT feature ($s$) influences a target ($t$) via three kinds of paths:

1. residual-direct: $s$'s decoders write to the residual stream, where it is read in at a later layer by $t$'s encoder.
2. attention-direct: $s$'s decoders write to the residual stream, are transported by some number of attention head OV steps, and then read by $t$'s encoder.
3. indirect: paths from $s$ to $t$ are mediated by other CLT features.

We note that the residual-direct influence is simply the product of the first feature's activation on this prompt times a virtual weight which is consistent across inputs.[23] These virtual weights are a simple form of global weights because of this consistent relationship. Virtual weights have been derived between many different components in neural networks, including attention heads [18], SAE features [8, 24], and transcoder features [12]. For CLTs, the virtual weight between two features is the inner product between the encoder of the downstream feature and the sum of decoders in between these two features.

More formally, let $\ell_s$ and $\ell_t$ be the layers of the encoder weights for features $s$ and $t$. Let $L_{st}$ be the set of layers in between these features such that $\forall \ell \in L_{st}, s \leq \ell < t$. Feature $s$ writes to all MLP outputs in $L_{st}$ before reaching feature $t$. Let $W_{\text{dec}}^{s,\ell}$ be the decoder weights for feature $s$ targeting layer $\ell$, and $W_{\text{enc}}^{t}$ be the encoder weights for feature $t$. Then, the virtual weights are computed as:

$$V_{st} = \langle \sum_{\ell \in L_{st}} W_{\text{dec}}^{s,\ell}, W_{\text{enc}}^{t} \rangle$$

Attribution graph edges consist of a sum of the residual-direct contribution (the virtual weight multiplied by a feature's activation) plus the attention-direct contribution.

There is one major problem with interpreting virtual weights: interference [7, 25]. Because millions of features are interacting via the residual stream, they will all be connected, and features which never activate together on-distribution can still have (potentially large) virtual weights between them. When this happens, the virtual weights are not suitable global weights because these connections never impact network function.

We can see interference at play in the following example: below, we take a "Say a game name" feature in 18L and plot its largest virtual weights by magnitude. Green bars indicate a positive connection and purple bars indicate a negative one. Many of the most strongly-connected features are hard to interpret or not clearly related to the concept.

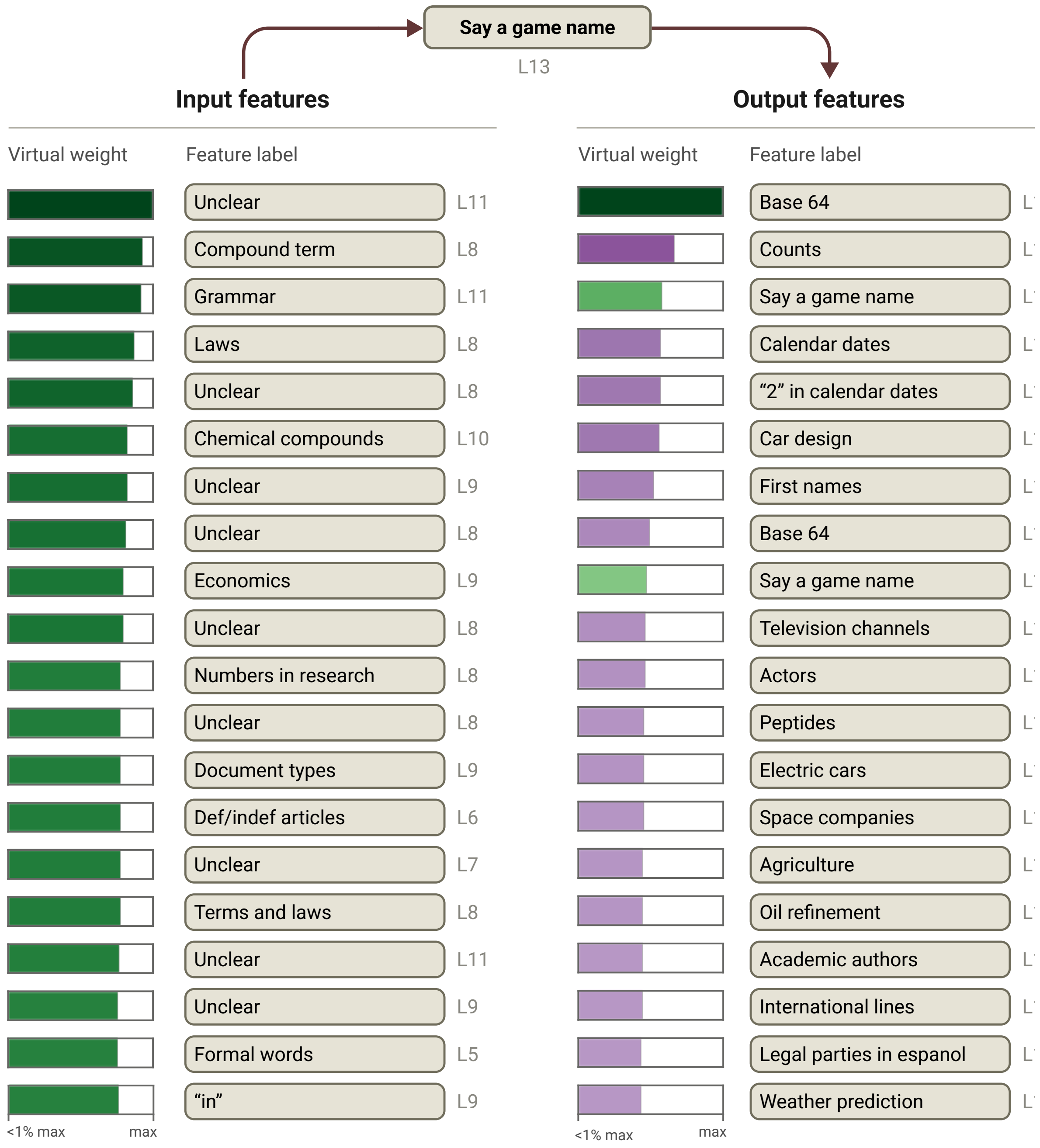


**Figure 20**: Largest virtual weights by magnitude for an example 18L feature. Green bars indicate a positive connection and purple bars indicate a negative one. Many large connections are not easily interpretable.

You might consider this a sign that virtual weights or our CLTs aren't capturing interpretable connections. However, we can still uncover many interpretable connections by trying to remove interference from these weights.[24]

There are two basic solutions to this problem. One is to restrict the set of features being studied to those active on a small domain (as we do in § 4.1 Global Weights in Addition). The other is to bring in information about the feature-feature coactivation on the data distribution.

For example, let $a_i$ be the activation for feature $i$. We can compute an expected residual attribution value by multiplying the virtual weight as follows.

$$V_{ij}^{\text{ERA}} = \mathbb{E}\big[\mathbb{1}(a_j > 0) V_{ij} a_i\big] = \mathbb{E}\big[\mathbb{1}(a_j > 0) a_i\big] V_{ij}$$

This represents the average strength of a residual-direct path across all of the prompts we've analyzed (also computed by Dunefsky et al. [12]). This is similar to computing the average of all attribution graphs within a context position across many tokens.[25] The indicator function in this expression ($\mathbb{1}(a_j > 0)$) captures how attributions are only positive when the target feature is active. As small feature activations are often polysemantic, we instead weight attributions using the target activation value:

$$V_{ij}^{\text{TWERA}} = \frac{\mathbb{E}\big[a_j a_i\big]}{\mathbb{E}\big[a_j\big]} V_{ij}$$

We call this last type of weight target-weighted expected residual attribution (TWERA). As shown in the equations, both of these values can be computed by multiplying the original virtual weights by ("on-distribution") statistics of activations.

Now, we revisit the example game feature from before but with connections ordered by TWERA. We also plot each connection's "raw" virtual weight for comparison. Many more of these connections are interpretable, suggesting that the virtual weights extracted useful signals but we needed to remove the interference in order to see them. The most interpretable features from the virtual weight plot above (another "Say a game name" and "Ultimate frisbee" feature) are preserved while many unrelated concepts are filtered out.

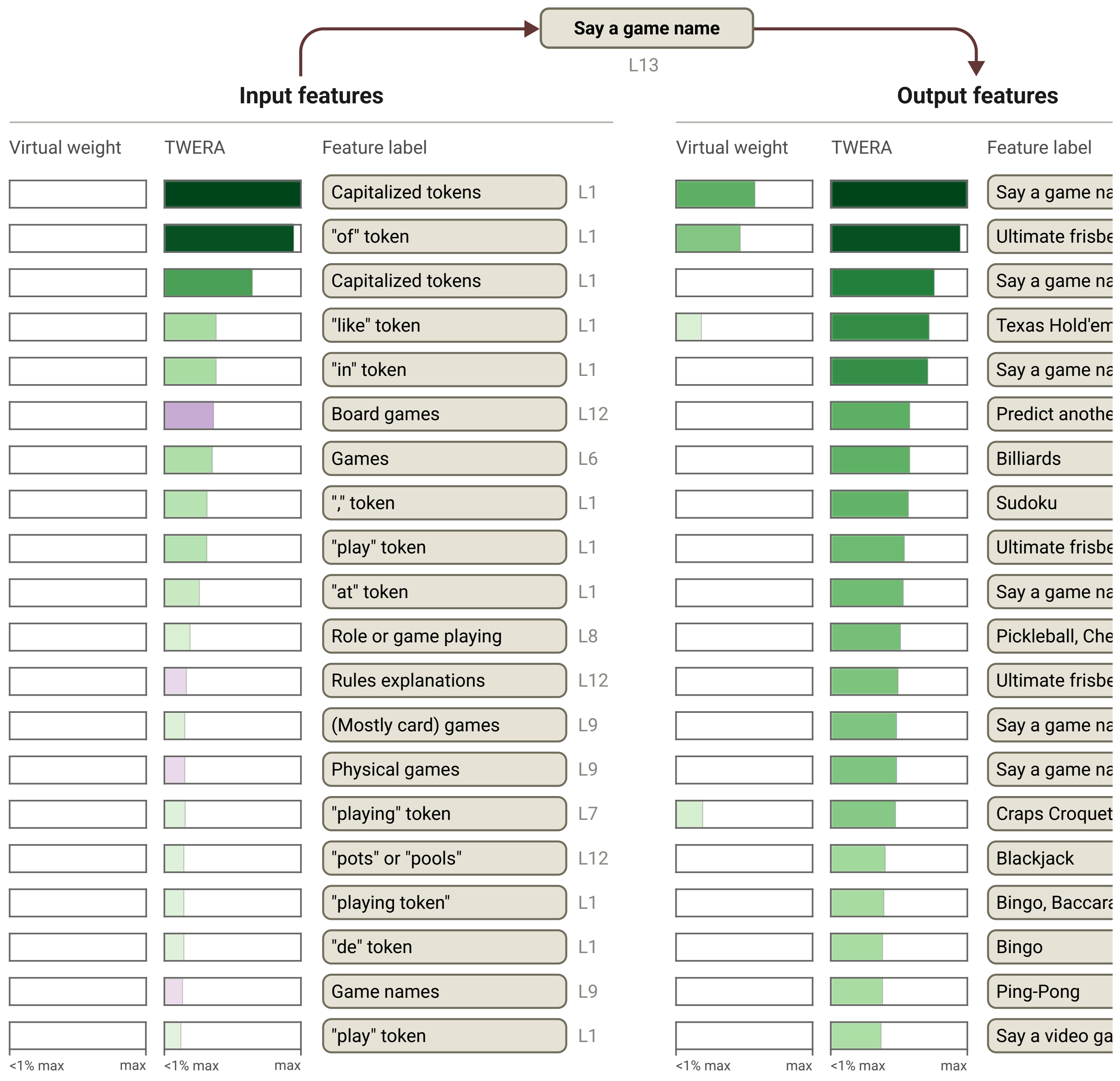


**Figure 21**: Largest TWERA values for the same example feature.

TWERA is not a perfect solution for interference. Comparing TWERA values to the raw virtual weights shows that many extremely small virtual weights have strong TWERA values.[26] This indicates that TWERA heavily relies on the coactivation statistics and strongly changes which connections are important beyond simply removing the large interference weights. TWERA also does not handle inhibition well (like attribution generally). We will explore these issues further in future work.

Still, we find that global weights give us a useful window into how features behave in a broader range of contexts than our attribution graphs. We'll use these methods to complement our understanding in the rest of this paper and in the companion paper.

## Global Weights in Addition

We now return to the simple addition problem from above on Haiku 3.5, and show how data-independent virtual weights reveal clear structure between the types in our taxonomy of addition features. We again consider completions of the 10,000 prompts `calc: a+b=` for **a**,**b** ∈ [0, 99]. In addition to the operand plots (again, defined above), we inspect the virtual weight graph after restricting the large $n_{\text{feat}} \times n_{\text{feat}}$ virtual weight matrix to the set which are active on at least 10 of the set of 10,000 addition prompts. This allows us to see all the feature-feature interactions that can occur via direct residual paths.

In the neighborhood of the features appearing in the 36+59 prompt above, we see:

- The _6 + _9 lookup table features feed into other sum features (sum=_15, sum=_25, etc.), which likely activate when combined with other magnitude features.
- The add _9 feature feeds into other ones-digit lookup table features, (_9 + _9, _0 + _9, etc.).
- The sum = _5 feature is fed by other lookup table features.
- The medium-precision ~36+60 lookup table feature is fed by an add ~62 feature in addition to the add ~57 feature we see on this prompt.

### Global Weights

We display a subset of the features with virtual weights to some of the features active on the **36+59** prompt. Dark features (🟧) and edges (—) are **active** on that prompt. Lighter features (⬜) and edges (—) are **not active** on the prompt but connected to it in the global circuit, and can contribute on other prompts.

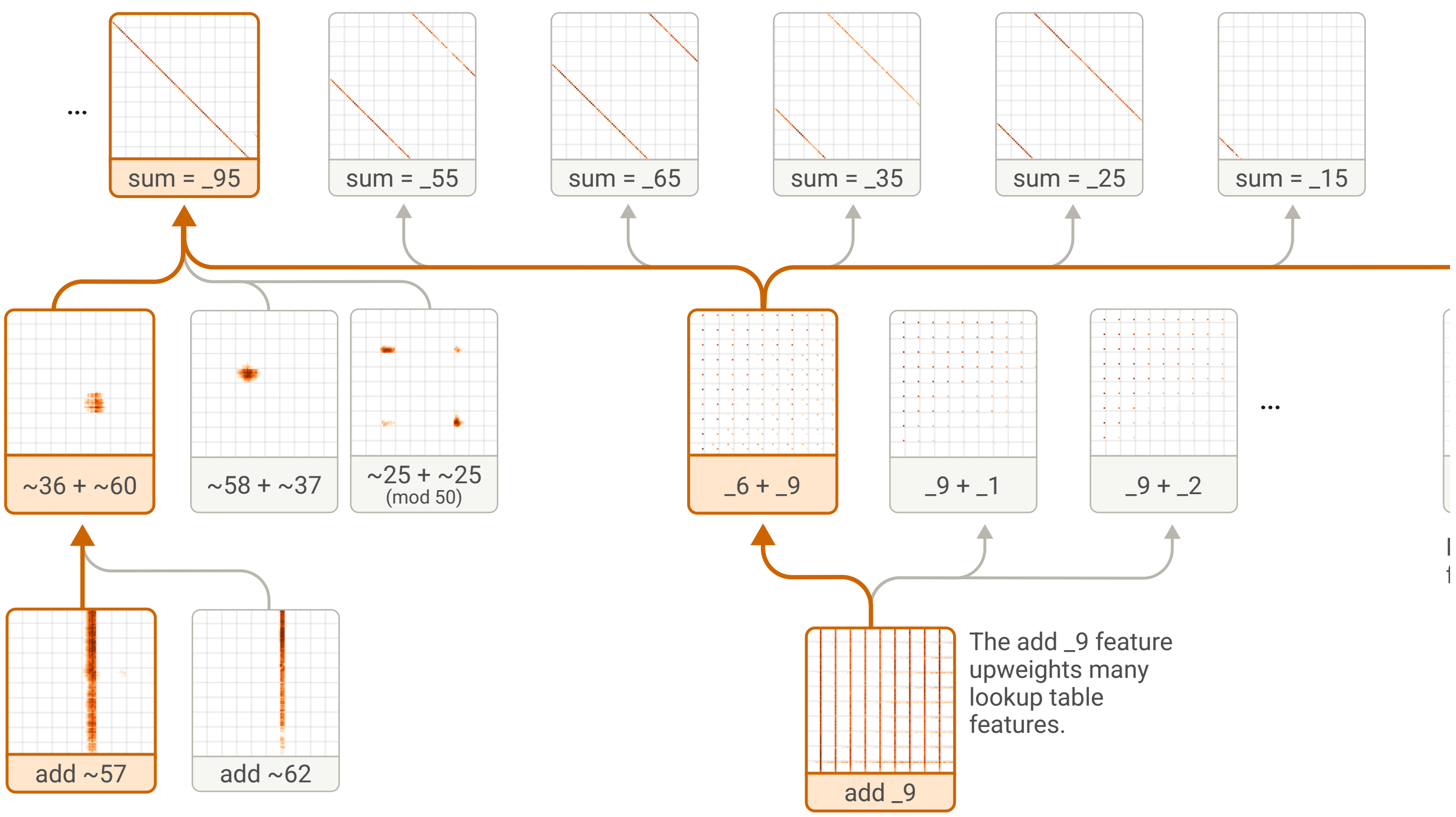


**Figure 22**: Virtual weight connections between Haiku features active on addition prompts. The connections from "calc: 36+59=" fit into general patterns.

We provide an interactive interface to explore the virtual weights connecting all 2931 features prominent on two-digit addition problems in our smaller 18L model.

We find that restricting to features active on this narrow domain of addition problems produces a global circuit graph where most edges are interpretable in terms of the operand function realized by the source and target features. Moreover, the connections between features recapitulate a more general version of the graph in the previous section; add features detect a specific operand as part of the input, lookup table features propagate this information to sum features which (in concert with the previous features) produce the model's final answer.

Several of the features we find take the form of heuristics as in Nikankin et al. [21]: features whose operand plots have predictive power directly push that prediction to the outputs: the low precision features promote outputs in the range matching their operands; the ones digit lookup table features directly promote outputs matching their mappings (e.g. _6 + _9 directly upweights _5 outputs). Almost none of these features represent the full solution until the very last layers of the model. As viewed by CLTs, the models use several intermediate heuristics rather than a coherent procedural program with a single confident output.

Our focus on the *computational* steps the model uses to perform addition is complementary to concurrent work by Kantamneni and Tegmark [22] which begins from *representations*. Inspired by the observation of spikes in the Fourier decomposition of the embedding vectors for integers, they find low-dimensional subspaces highly correlated with numbers' magnitudes and mod 2, 5, 10, and 100 components. Projecting to those subspaces preserves much of the model's performance on the task, consistent with a "Clock" algorithm performing separate calculations in each modulus, which interfere constructively at the end; the CLT features show essentially high- and low-precision versions of that method. Some of the important features we find have operand plots similar to their neurons, which they fit as a (thresholded) sum of Fourier modes.[27] Some of the important features appearing in our graphs (such as operands or sums that *start* with fixed digits, e.g. 95_ and 9_) aren't describable in Fourier terms, consistent with the existence of some error in their low-rank approximation.[28] Identifying the representational basis of the ensemble of computational strategies revealed by our unsupervised approach is a promising direction for future work.

Altogether, we've replicated a view of the base model using heuristics finding matching CLT features, we've shown how these heuristics contribute to separable pathways through intervention experiments, and we've demonstrated how these heuristics are connected, building off one another to collectively solve the addition task.

## Evaluations

In this section, we perform qualitative and quantitative evaluations of transcoder features and the attribution graphs derived from them, focusing especially on *interpretability* and *sufficiency*. For readers interested in a higher level discussion of findings and limitations, we recommend skipping ahead to § 6 Biology and § 7 Limitations.

Our methods produce causal graph descriptions of the model's mechanisms on a particular prompt. How can we quantify how well these descriptions capture what is really going on in the model? It is difficult to distill this question to one number, as several factors are relevant:

**Interpretability**. How well do we understand what individual features "mean"? We attempt to quantify interpretability in a few ways below; however, we still rely heavily on subjective evaluation in practice. The coherence of our groupings of features into "supernodes" also warrants evaluation. We do not attempt to quantify this in this work, instead leaving it to readers to verify for themselves that our groupings are sensible and interpretable. We also note that in the context of attribution graphs, interpretability of the *graph* is just as important as interpretability of individual features. To that end, we quantify one notion of graph simplicity: average path length.

**Sufficiency**. To what extent are our (pruned) attribution graphs sufficient to explain the model's behavior? We attempt to quantify this in several ways. The most straightforward such evaluation is our measurement of how well the replacement model's outputs match the underlying model, discussed in § 2.2 From Cross-Layer Transcoder to Replacement Model . This is a "hard" evaluation in that a single error anywhere along the computational graph can severely degrade performance. We also compute a few "softer" measures of sufficiency below, that measure the proportion of error nodes in attribution graphs. Note that in many instances, we present schematics of *subgraphs* of a pruned attribution graph that portray what we believe to be its most noteworthy components. We intentionally do *not* measure the sufficiency of these subgraphs, as they often intentionally exclude "boring" but necessary parts of the graph (e.g. "this is a math problem" features in addition prompts). We leave it to future work to find more principled ways to distill attribution graphs to their "interesting" components and quantify how much (and what kind of) information is lost.[29]

**Mechanistic faithfulness.** To what extent are the mechanisms we identify *actually* used by the model? To measure this, we perform perturbation experiments (such as inhibiting active features) and measuring whether the effects agree with what is predicted by the local replacement model (the underlying object portrayed by our attribution graphs). We attempt to do so quantitatively below, and we also validate faithfulness on our specific case studies, in particular focusing on faithfulness of the mechanisms we have identified as interesting / important. Note that our notion of mechanistic faithfulness is related to the idea of *necessity* of circuit components to a model's computation. However, necessity can be a somewhat restrictive notion – mechanisms that are not strictly "necessary" for the model's output may still be important to identify, especially in cases where multiple mechanisms cooperate in parallel to contribute to a computation, as we often observe.

We note that the specific evaluations we use are in many cases new to this work. In part this is because our work is somewhat unique in focusing on attribution graphs for individual prompts, rather than identifying circuits underlying the model's performance of an entire *task*. Developing better automatic methods for evaluating interpretability, sufficiency, and faithfulness of the entire pipeline (features, supernodes, graphs) is an important subject of future research. See § 9 Related Work for more detail on prior circuit evaluation methods.

## Cross-Layer Transcoder Evaluation

## QUALITATIVE FEATURE EVALUATIONS

For CLT features to be useful to us, they must be human-interpretable (perhaps in the future it will suffice for them to be AI-interpretable!). Interpretability is ultimately a qualitative property – the best gauge of the interpretability of our features is to view them in action. A standard (though incomplete) tool for understanding what a feature represents is to view the dataset examples for which it is active (we refer to the collection of such examples as our "feature visualization"). We provide thousands of feature visualizations in the context of our case studies of circuits later in this paper and in the companion paper. Below we also show 50 randomly sampled features from assorted layers of each model.

Our feature visualizations show snippets of samples from public datasets (Common Corpus, The Pile with books removed [27], LMSYS Chat 1m [28], and Isotonic Human-Assistant Conversation) that most strongly activate the feature, as well as examples that activate the feature to varying degrees interpolating between the maximum activation and zero. Highlights indicate the strength of the feature's activation at a given token position. We also show the output tokens that the feature most strongly promotes / inhibits via its direct connections through the unembedding layer (note that this information is typically more meaningful for features in later model layers).

**18L Features** (Hover)

Layer 1 Layer 5 Layer 9 Layer 13 Layer 17

**Haiku Features** (Hover)

First layer Mid-layer Final layer

At a very coarse level, we find several types of features:

- Input features that represent low-level properties of text (e.g. specific tokens or phrases). Most early-layer features are of this kind, but such features are also present in middle and later layers.
- Features whose activations represent more abstract properties of the context. For example, a feature for the danger of mixing common cleaning chemicals. These features appear in middle and later layers.
- Features that perform *functions*, such as an "add 9" feature that causes the model to output a number that is nine greater than another number in its context. These tend to be found in middle and later layers.
- Output features, whose activations promote specific outputs, either specific tokens or categories of tokens. An example is a "say a capital" feature, which promotes the tokens corresponding to the names of different U.S. state capitals.
- Polysemantic features, especially in earlier layers, such as this feature that activates for the token "rhythm", Michael Jordan, and several other unrelated concepts.

In line with our previous results on crosscoders [15], we find that features also vary in the degree to which their outputs "live" in multiple layers – some features contribute primarily to reconstructing one or a few layers, while others have strong outputs all the way through the final layer of the model, with most features falling somewhere in between.

We also note that the abstractions represented by Haiku features are in many cases richer than those in the smaller 18L model, consistent with the model's greater capabilities.

## QUANTITATIVE CLT EVALUATIONS

In § 2.2 From Cross-Layer Transcoder to Replacement Model, we evaluated the ability of our CLTs to reproduce the computation of the underlying model. Here, we measure reconstruction error, sparsity (measured by "L0", the average number of features active per input token), and feature interpretability. As we increased the size of our CLT, we observed Pareto-improvements in reconstruction error (averaged across layers) and feature sparsity (in 18L, reconstruction error decreased at a roughly fixed L0, while in Haiku, reconstruction error and L0 both decreased). In our largest 18L run (10M features), we attained a normalized mean reconstruction error of ~11.5% and an average L0 of 88. In our largest Haiku run (30M features), we attained a normalized reconstruction error of 21.7%, and an average L0 of 235.

We also computed two LLM-based quantitative measures of interpretability, introduced and described in more detail in [29]:

- **Sort Eval**: we take two randomly sampled features and identify the set of dataset examples that activate them most strongly. We present these sets of examples to Claude, including the token-by-token feature activation information. Then we take other dataset examples that activate *only one* of the features, present these to Claude, and ask it to guess which feature these examples correspond to (based on the initial example sets). The final evaluation score is the empirical likelihood that Claude guesses the correct feature on any given pair.

- **Contrastive Eval**: we generate (using Claude) pairs of prompts that are similar in content and structure but differ in one key respect. We compute the sets of features that activate on *only one* of the prompts but not the other. We present Claude with the feature vis for each such feature, along with the two prompts, and ask it to guess which prompt caused the feature to activate. The final evaluation score is the empirical likelihood that Claude guesses the correct prompt for features across trials.

We find that according to both measures, the quality of CLT features improves with scale (alongside improving reconstruction error) – see plots below.[30]

We also compare CLTs to two baselines: per-layer transcoders (PLTs) trained at each layer of the model, and the raw neurons of the model thresholded at varying activation levels.[31] We find that on all metrics, CLTs outperform PLTs, and both CLTs and PLTs substantially outperform the Pareto frontier of thresholded neurons.

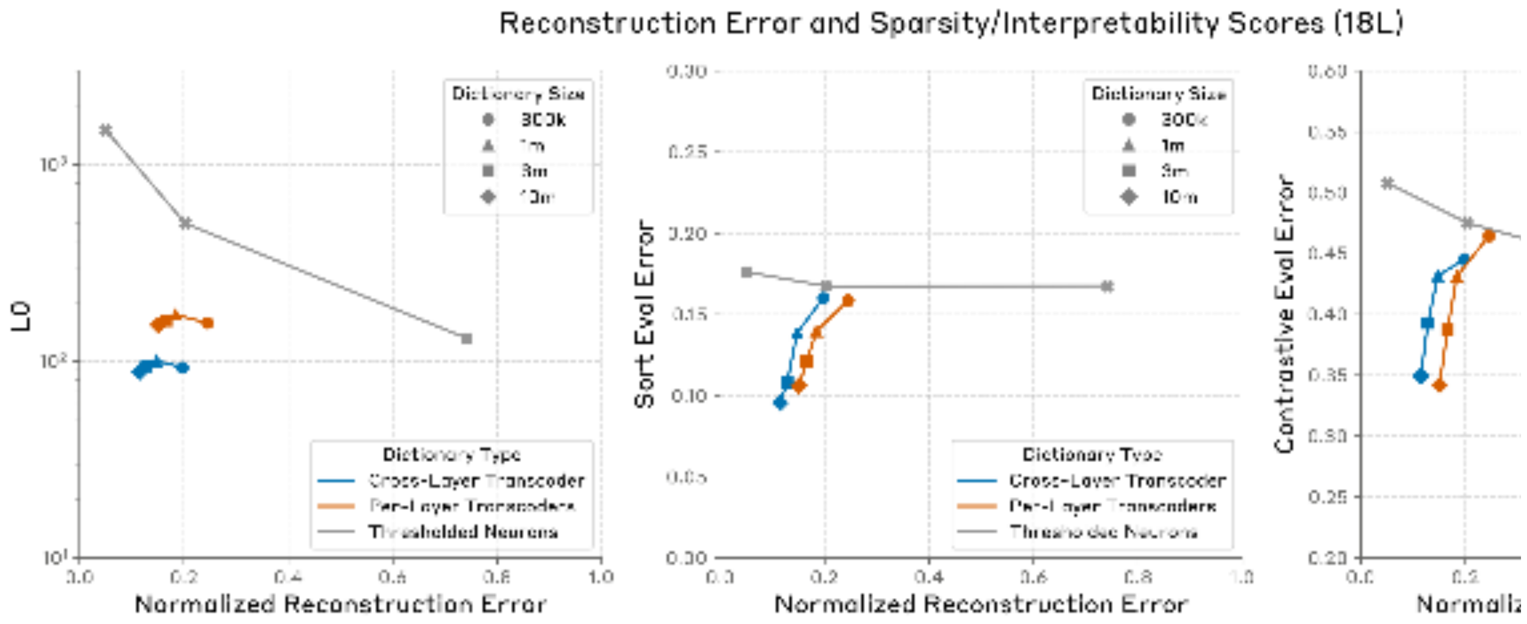


**Figure 23**: Reconstruction error and sparsity/interpretability scores for multiple dictionary types fit to 18L.

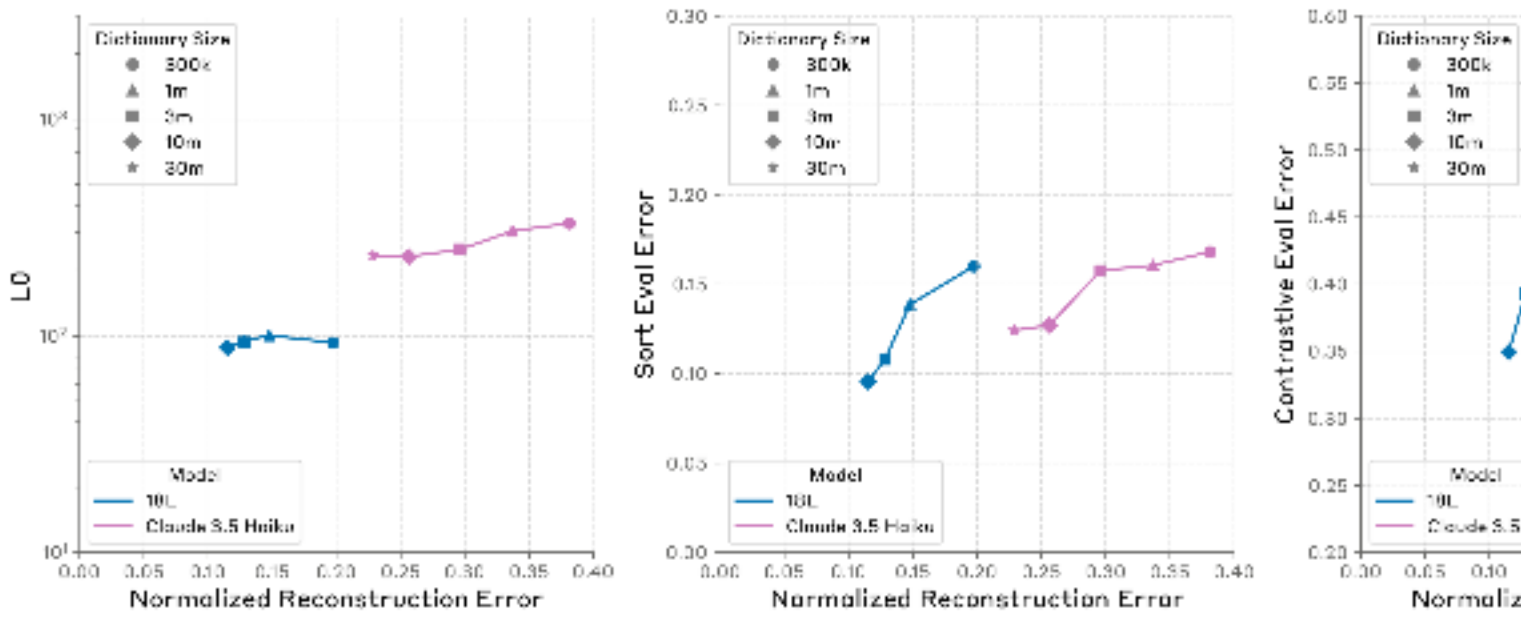


**Figure 24**: Reconstruction error and sparsity/interpretability scores for 18L and Haiku.

## Attribution Graph Evaluation

Case studies in § 3 Attribution Graphs focused on qualitative observations derived from attribution graphs. In this section, we describe our more quantitative evaluations used to compare methodological choices and dictionary sizes. In each of the following subsections, we will introduce a metric and compare graphs generated using (1) cross-layer transcoders, (2) per-layer transcoders for every layer, and (3) thresholded neurons[32] . To connect the quantitative to the qualitative, we will link to graphs which score especially high or low on each of these metrics.

While we don't treat these metrics as fundamental quantities to be optimized, they have proven a useful guide for tracking ML improvements in dictionary learning and to flag prompts our method performs poorly on.

## COMPUTING NODE INFLUENCE

Our graph-based metrics rely on quantities derived from the *indirect influence matrix*. Informally, this matrix measures how much each pair of nodes influences each other via all possible paths through the graph. This gives a natural importance metric for each node: how much it influences the logit nodes. We also commonly compare how much influence comes from error nodes vs. non-error nodes.

To construct this matrix, we start with the adjacency matrix of the graph. We replace all the edge weights with their absolute values (or simply clamp negative values to 0) to obtain an unsigned adjacency matrix and then normalize the input edges to each node so that they sum to 1. Let $A$ refer to this normalized, unsigned adjacency matrix, indexed as (target, source).

The indirect influence matrix is $B = A + A^2 + A^3 + \cdots$, which is a Neumann series and can be efficiently computed as $B = (I - A)^{-1} - I$. The entries of $B$ indicate the sum of the strengths of all paths between a given pair of nodes, where the strength of any given path is given by the product of the values of its constituent edges in $A$. To compute a logit influence score for each node, we compute a weighted average of the rows of B corresponding to logit nodes (weighted by the probability of the particular logit).

## MEASURING AND COMPARING PATH LENGTH

A natural metric of graph complexity is the average path length from embedding nodes to logit nodes. Intuitively, shorter paths are easier to understand as they require interpreting fewer links in the causal chain.

To measure the influence of paths of different lengths, we compute influence matrices $B_\ell = \sum_{i=0}^{\ell} A^i$.The influence of paths of length less than or equal to $\ell$ is then given by $P_\ell = \sum_e B^\ell_{t,e}$ where $e$ are all embedding nodes and $t$ is the logit node.[33]

Below, we compare graphs built from our 10M CLT, 10M PLTs, and thresholded neurons in terms of their influence by path length averaged across a dataset of pretraining prompts (without pruning).[34]

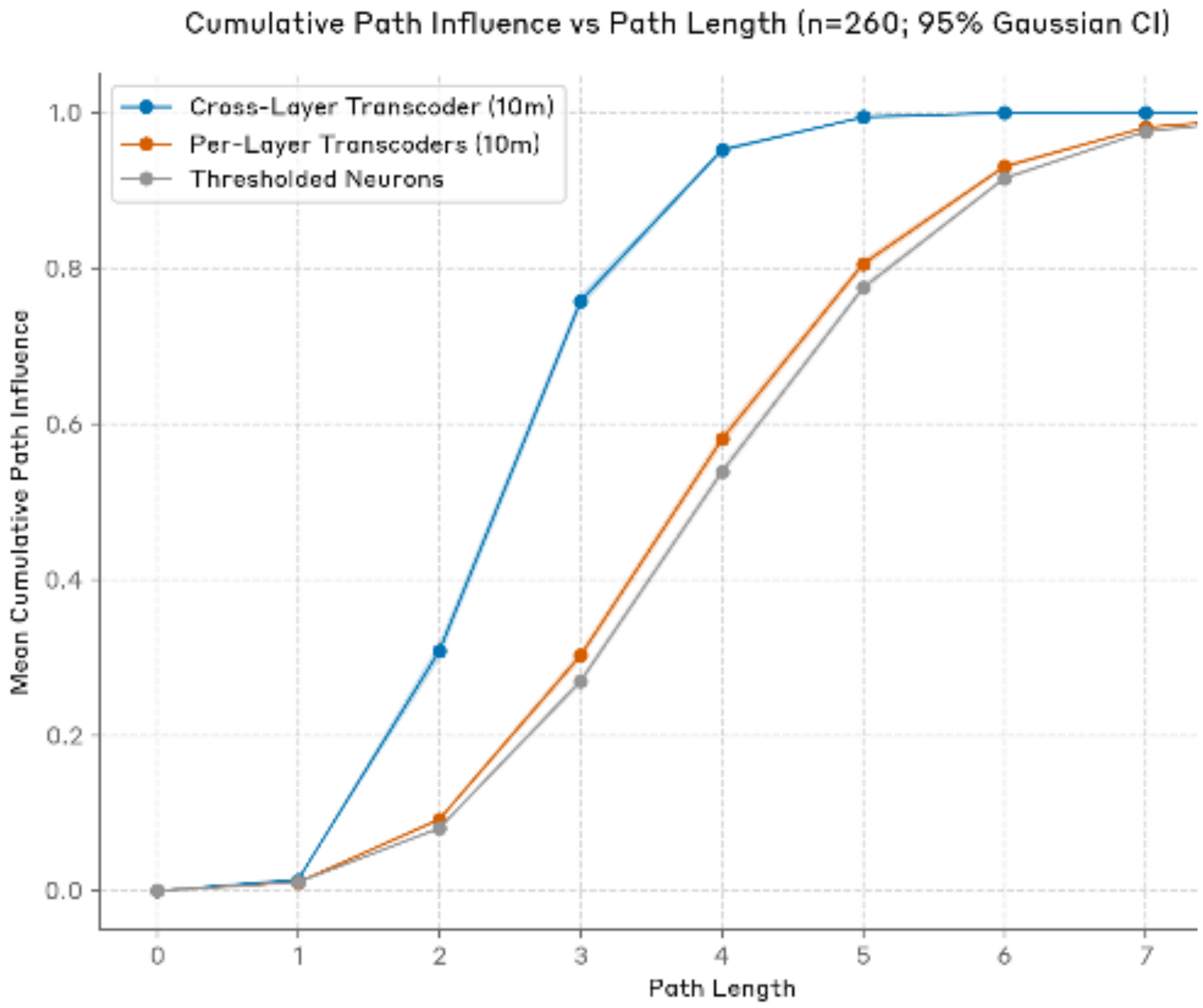


**Figure 25**: Comparison of average cumulative influence versus path length by dictionary type. Shorter paths are easier t

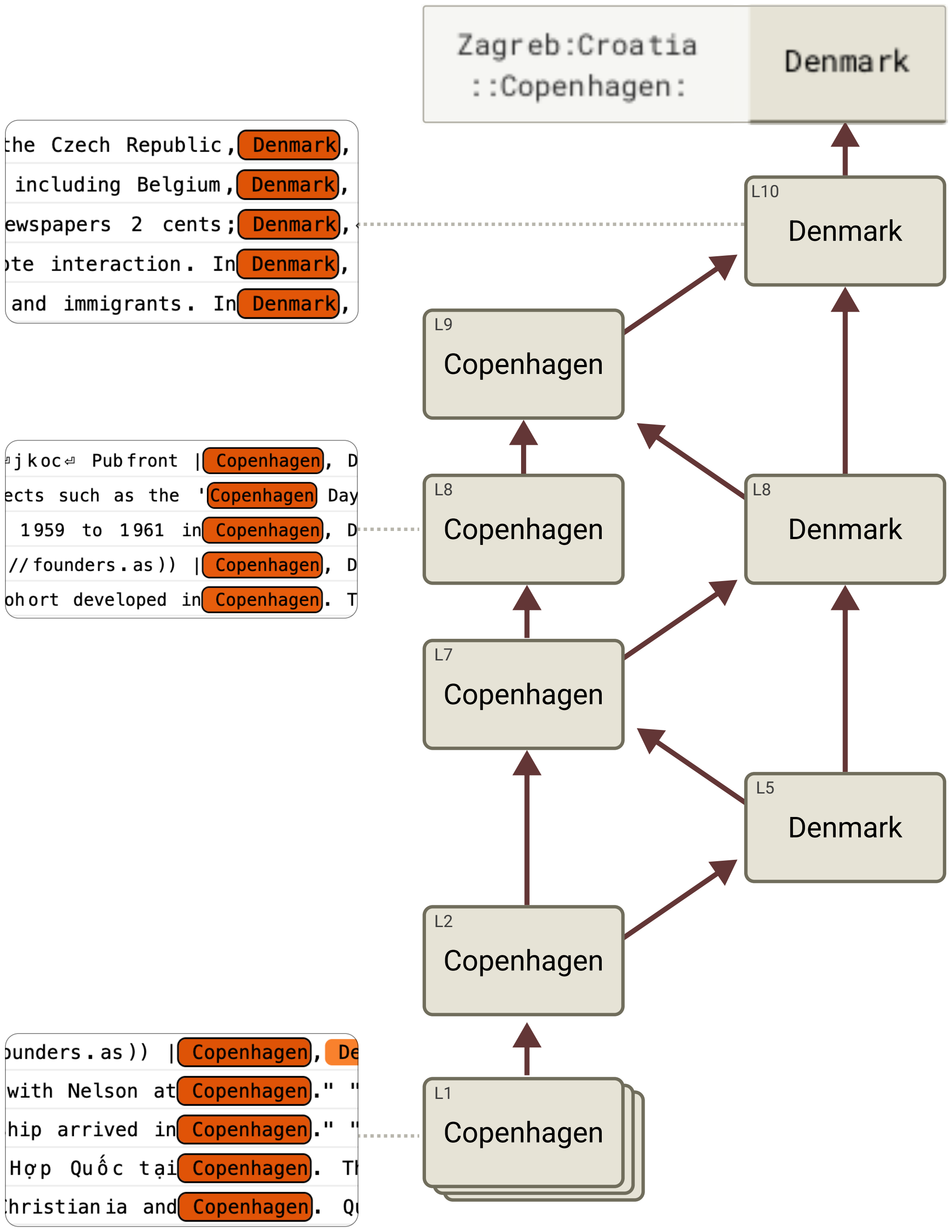
One of the most important advantages of crosslayer transcoders is the extent to which they reduce the path lengths in the graph. To understand how large of a qualitative difference this is, we invite the reader to view these graphs generated with different types of replacement models for the same prompt.

| Replacement Model Type | Average Path Length | Graph Link |
|---|---|---|
| Cross-Layer Transcoder (10m) | 2.3 | capital-analogy-clt |
| Per-Layer Transcoders (10m) | 3.7 | capital-analogy-plt |

We find that one important way in which cross-layer transcoders collapse paths is the case of amplification, where many similar features activate each other in sequence. For example, on the prompt `Zagreb:Croatia::Copenhagen:` the per-layer transcoder shows a path of length 7 composed entirely of Copenhagen features while the cross-layer transcoder collapses them all down to layer 1 features.

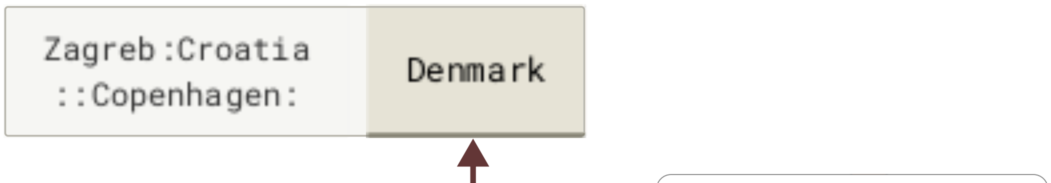
**Per-Layer Transcoders (PLTs)**

PLTs model computation independently at each layer. This can create repeated features.


Zagreb:Croatia
::Copenhagen:
Denmark
L10
Denmark
L9
Copenhagen
L8
Copenhagen
L8
Denmark
L7
Copenhagen
L5
Denmark
L2
Copenhagen
L1
Copenhagen
the Czech Republic, Denmark,
including Belgium, Denmark,
ewspapers 2 cents; Denmark,
ote interaction. In Denmark,
and immigrants. In Denmark,
jkoc Pubfront | Copenhagen, D
ects such as the 'Copenhagen Day
1959 to 1961 in Copenhagen, D
//founders.as)) | Copenhagen, D
ohort developed in Copenhagen. T
ounders.as)) | Copenhagen, De
with Nelson at Copenhagen." "
hip arrived in Copenhagen." "
Hợp Quốc tại Copenhagen. Th
hristiania and Copenhagen. Qu

**Cross-Layer Transcoder (CLT)**

CLTs can merge computation across layers, simplifying graphs (but potentially being less faithful).



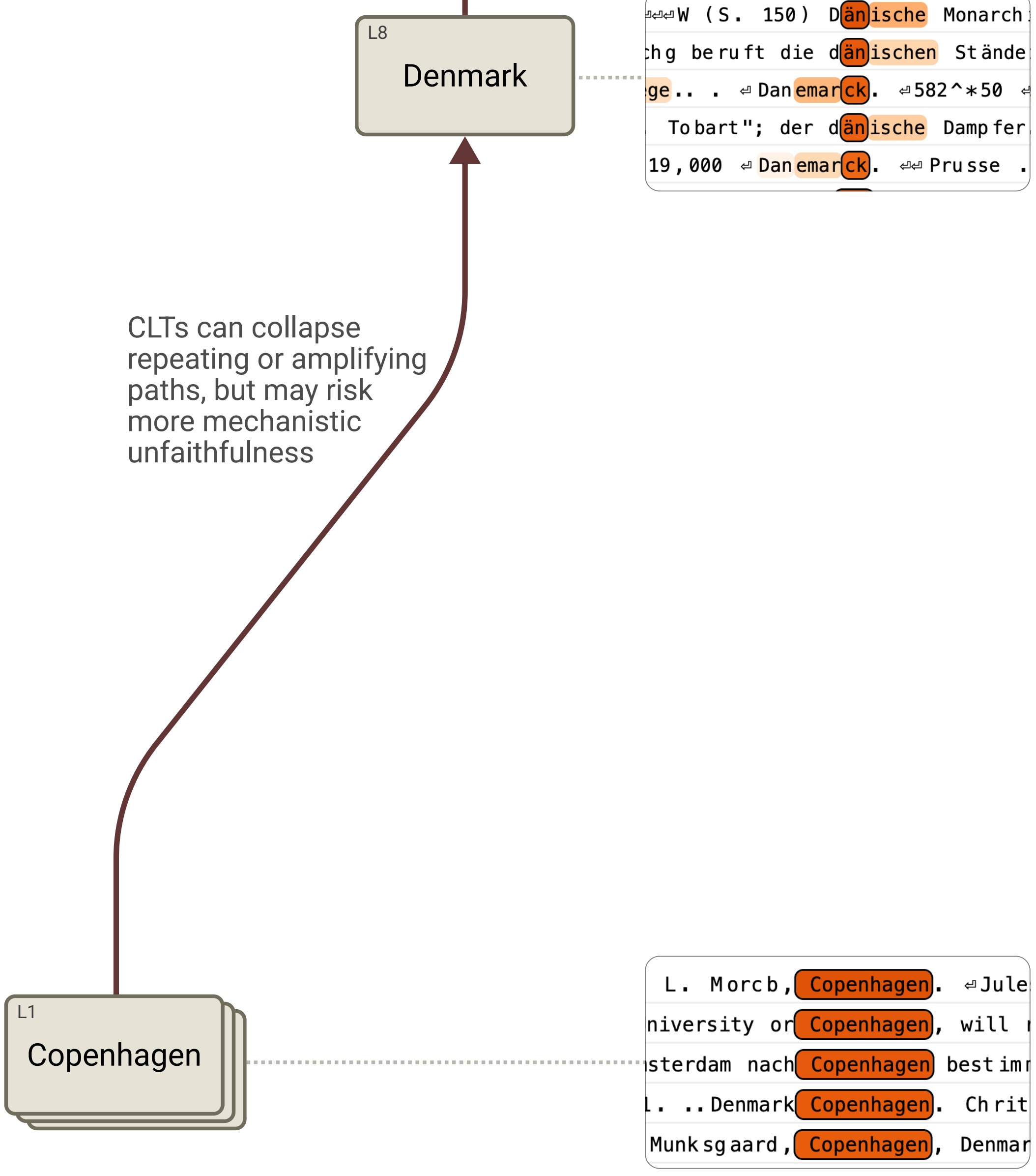

L8
Denmark
CLTs can collapse repeating or amplifying paths, but may risk more mechanistic unfaithfulness
L1
Copenhagen

**Figure 26**: CLTs produce simpler graphs than PLTs.

This example illustrates both the advantages and disadvantages of consolidating amplification of a repeated computation across multiple layers into a single cross-layer feature. On one hand, it makes interpretability substantially easier, as it automatically collapses duplicate computations into a single feature without needing to do post hoc analysis or clustering. It also reduces the risk of "chain-breaking", where missing one feature in an amplification chain inhibits the ability to trace back further into the graph (i.e., a relevant amplification feature is missing for one step of the path, breaking the causal chain). On the other hand, the CLT has a different causal structure than the underlying model, which increases the risk that the replacement model's mechanisms diverge from the underlying model's. In the above example, we observe a set of Copenhagen features that activate a Denmark feature, which initiates a mutually reinforcing chain of Copenhagen and Denmark features. This dynamic is invisible in CLT graphs, and to the extent this dynamic is also present in the underlying model, it is an example of CLTs being mechanistically unfaithful.

## EVALUATING AND COMPARING GRAPH COMPLETENESS

Because our replacement model has reconstruction errors, we want to measure how much of the model's computation is being captured. That is, how much of the graph influence is attributable to feature nodes versus error nodes.

To measure this, we primarily rely on two metrics:

- **Graph completeness score:** measures the fraction of input edges (weighted by the target node's logit influence score) that come from feature or embedding nodes rather than error nodes.
- **Graph replacement score:** measures the fraction of end-to-end graph paths (weighted by strength) that proceed from embedding nodes to logit nodes via feature nodes (rather than error nodes).

Intuitively, the completeness score gives more "partial credit" and measures how much of the most important node inputs are accounted for, whereas replacement score rewards complete explanations.

Below, we report average *unpruned* graph replacement and completeness scores for dictionaries of various sizes and types on our pretraining prompt dataset. We find the biggest methodological improvement comes when moving from per-layer to cross-layer transcoders, with large but diminishing returns from scaling the number of features.

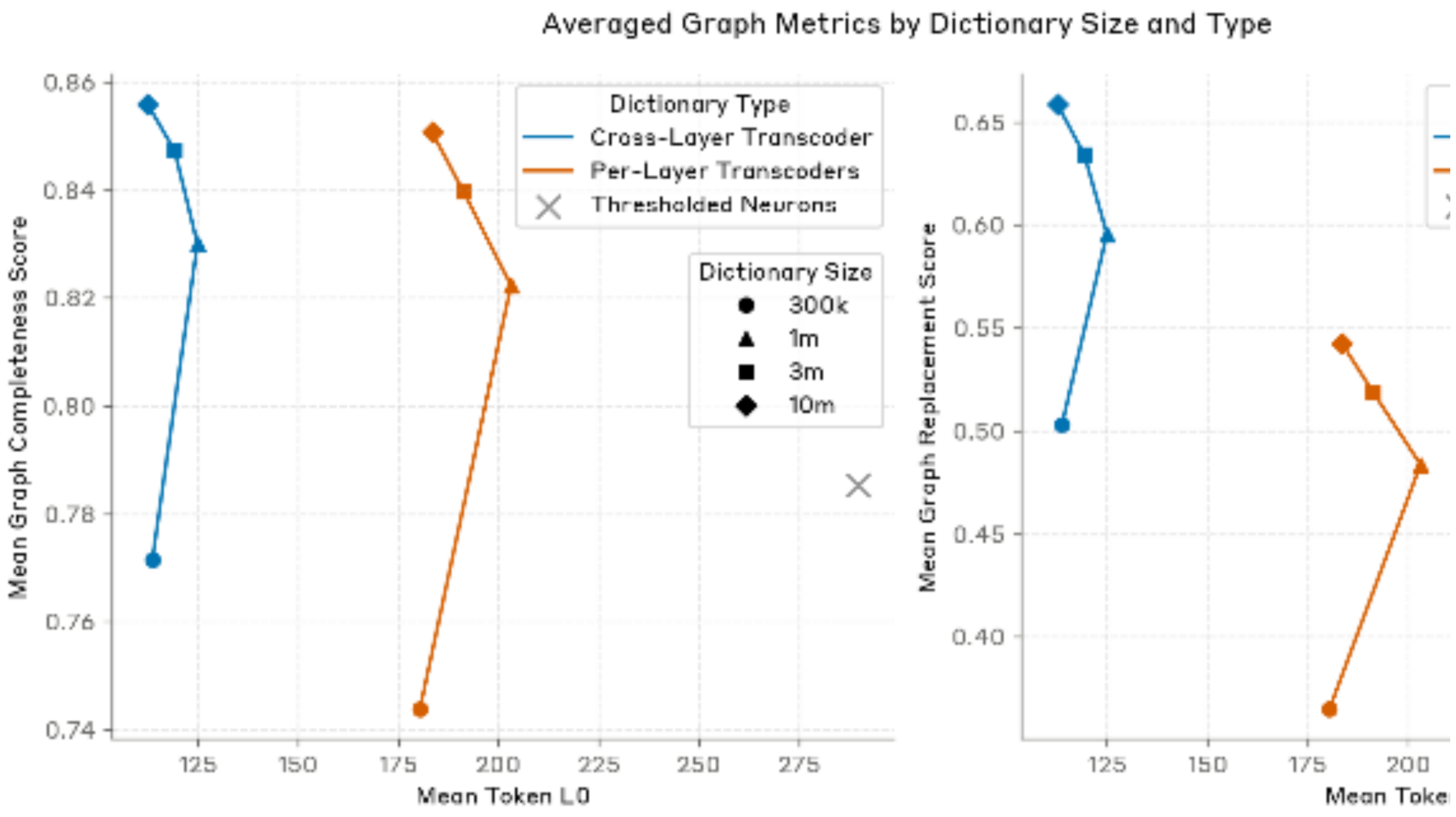


**Figure 27**: Comparison of graph completeness and replacement score versus per-token L0 for graphs generated with different underlying dictionaries correspond to graphs with fewer holes, and lower L0 suggests more interpretable graphs.

To contextualize the qualitative difference we observe in graphs with varying scores, we invite the reader to explore some representative attribution graphs. Note, these graphs are pruned with our default pruning, which we describe in more detail below.

| Replacement Model Type | Completeness Score | Replacement Score | Graph Link |
|---|---|---|---|
| Cross-Layer Transcoder (10m) | 0.80 | 0.61 | uspto-telephone-clt |
| Per-Layer Transcoders (10m) | 0.78 | 0.37 | uspto-telephone-plt |

## GRAPH PRUNING

We rely heavily on pruning to make graphs more digestible. To decide how much to prune the graph, we can use the completeness and replacement metrics described above, but with pruned nodes now counting towards the error terms. By varying the pruning threshold, we chart a frontier between the number of {nodes, edges} and {replacement, completeness} scores (see Appendix for full plots and details).

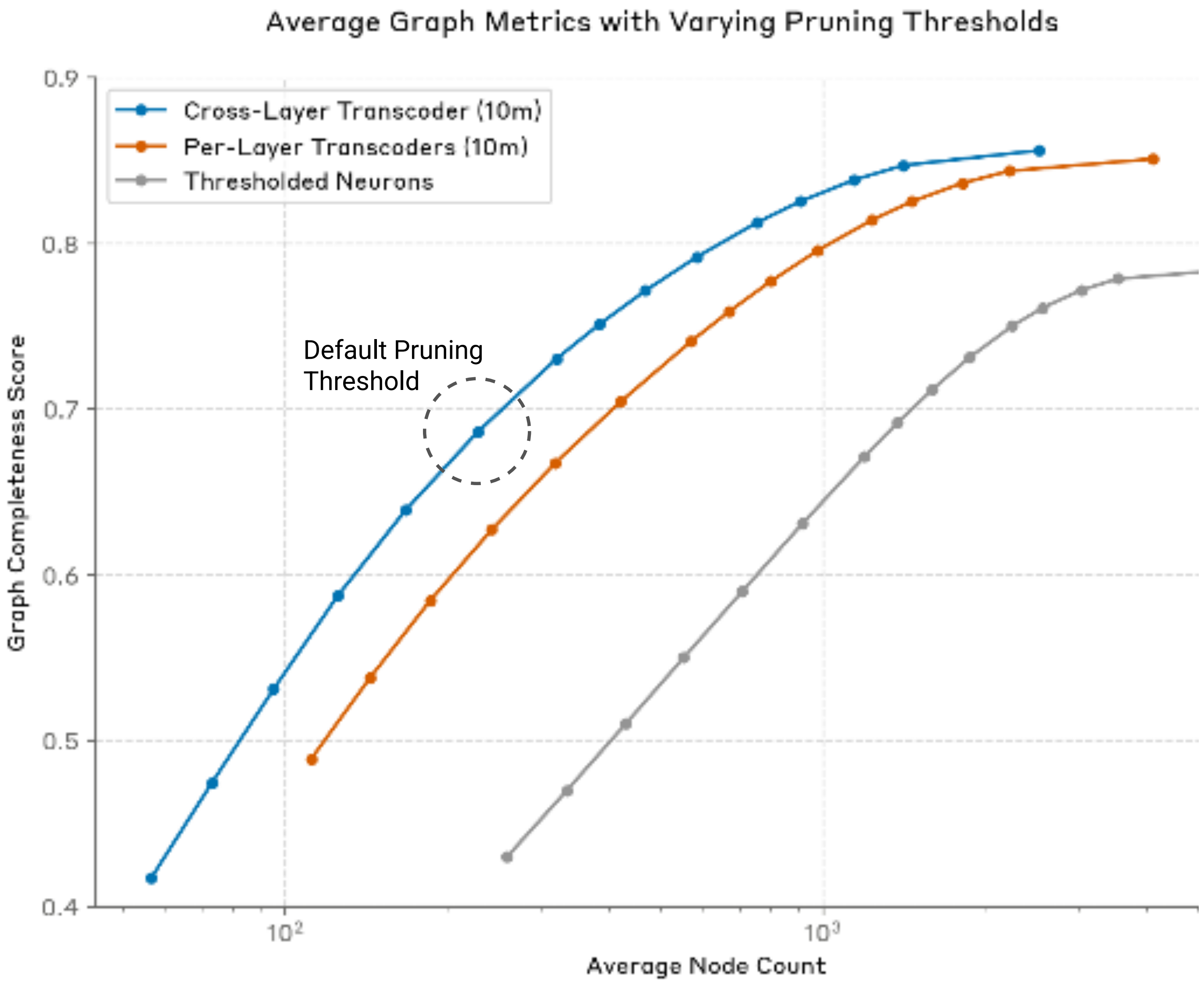


**Figure 28**: Average node count and completeness score for graphs as a function of pruning threshold.

We find we can generally reduce the number of nodes by an order of magnitude while reducing completeness by only 20%.

For a sense of the qualitative difference, in the table below we link to attribution graphs for the same prompt (another acronym) but with different pruning thresholds.

| Pruning Threshold | Completeness Score | Node Count | Graph Link |
|---|---|---|---|
| 0.95 | 0.87 | 236 | iasg-p95 |
| 0.9 | 0.83 | 137 | iasg-p90 |
| 0.8 (default) | 0.70 | 55 | iasg-p80 |
| 0.7 | 0.58 | 27 | iasg-p70 |

## Evaluating Mechanistic Faithfulness

As discussed in § 3.5 Validating Attribution Graph Hypotheses with Interventions, attribution graphs provide hypotheses about mechanisms, which must be validated with perturbation experiments. This is because attribution graphs describe interactions in the local replacement model, which may differ from the underlying model. In most of our work, we use attribution graphs as a tool for generating hypotheses about specific mechanisms ("Feature A activates Feature B, which increases the likelihood of Token X") operating inside the model, which correspond to "snippets" of the attribution graph. We summarize the results of three kinds of validation experiments, which are described in more detail in § G Appendix: Validating the Replacement Model.

We start by measuring the extent to which influence metrics derived from attribution graphs are predictive of intervention effects on the logit and other features. First, we measure the extent to which a node's logit influence score is predictive of the effect of ablating a feature on the model's output distribution. We find that influence is significantly more predictive of ablation effects than baselines such as direct attribution (i.e. direct edges in the attribution graph, ignoring multi-step paths) and activation magnitude (see Validating Node-to-Logit Influence). We then perform a similar analysis for interactions between features. We compute the influence score between pairs of features, and compare it to the relative effect of ablating the upstream feature in the pair on the activation of the downstream one. We observe a Spearman correlation of 0.72, which is evidence that graph influence is a good proxy for effects in the downstream model (see Validating Feature-to-feature Influence). See Nuances of Steering with Cross-Layer Features for some complexities in interpreting these results.

The metrics above help provide an estimate of the likelihood that an intervention experiment will validate a *specific* mechanism in the graph. We might also be interested in a more general validation of *all* the mechanistic hypotheses implicitly made by our attribution graphs. Thus, another complementary approach to validation is to measure the mechanistic faithfulness of the local replacement model *as a whole*, rather than specific paths within attribution graphs. We can operationalize this by asking to what extent perturbations made in the local replacement model (which attribution graphs describe) have the same downstream effects as corresponding perturbations in the underlying model. We find that while perturbation results are reasonably similar between the two models when measured one layer after the intervention (~0.8 cosine similarity, ~0.4 normalized mean squared error), perturbation discrepancies compound significantly over layers.[35] For more details, see Evaluating Faithfulness of the Local Replacement Model.

## Biology

In our companion paper, we use the method outlined here to perform deep investigations of the circuits in nine behavioral case studies of the frontier model Haiku 3.5. These include:

- **Multi-Step Reasoning.** We present a simple example where the model performs “two-hop” reasoning to complete “The capital of the state containing Dallas is…”, going Dallas → Texas → Austin. We can see and manipulate its representation of the intermediate Texas step.
- **Planning in Poems.** We show that the model plans its outputs when writing lines of poetry. Before beginning to write each line, the model identifies potential rhyming words that could appear at the end. These preselected rhyming options then shape how the model constructs the entire line.
- **Multilingual Circuits.** We find the model uses a mixture of language-specific and abstract, language-independent circuits (which are more prevalent in Claude 3.5 Haiku than a smaller model).
- **Addition.** We highlight a case where the same addition circuitry generalizes between very different contexts, and uncover qualitative differences between the addition mechanisms in Claude 3.5 Haiku and a smaller, less capable model.
- **Medical Diagnoses.** We show an example in which the model identifies candidate diagnoses based on reported symptoms, and uses these to inform follow-up questions about additional symptoms that could corroborate the diagnosis – all “in its head,” without writing down its steps.
- **Entity Recognition and Hallucinations.** We uncover circuit mechanisms that allow the model to distinguish between familiar and unfamiliar entities, which determine whether it elects to answer a factual question or profess ignorance. “Misfires” of this circuit can cause hallucinations.
- **Refusal of Harmful Requests.** We find evidence that the model constructs a general-purpose “harmful requests” feature during finetuning, aggregated from features representing *specific* harmful requests learned during pretraining.
- **An Analysis of a Jailbreak**, which works by first tricking the model into starting to give dangerous instructions “without realizing it,” and continuing to do so due to pressure to adhere to syntactic and grammatical rules.
- **Chain-of-thought Faithfulness.** We explore the faithfulness of chain-of-thought reasoning to the model’s actual mechanisms. We are able to distinguish between cases where the model genuinely performs the steps it says it is performing, cases where it makes up its reasoning without regard for truth, and cases where it *works backwards* from a human-provided clue so that its “reasoning” will end up at the human-suggested answer.
- **A Model with a Hidden Goal.** We also apply our method to a variant of the model that has been finetuned to pursue a secret goal of exploiting biases in its training process. While the model is reluctant to reveal its goal out loud, our method exposes it, revealing the goal to be “baked in” to the model’s “Assistant” persona.

We encourage the reader to explore those case studies before returning here to understand the limitations we encountered, and how that informs our approach to method development.

## Limitations

Despite the exciting results presented here and in the companion paper, our methodology has a number of significant limitations. At a high level, the most significant ones are:

- **Missing Attention Circuits** – We don't explain *how* attention patterns are computed by QK-circuits, and can sometimes "miss the interesting part" of the computation as a result.
- **Reconstruction Errors & Dark Matter** – We only explain a portion of model computation, and much remains hidden. When the critical computation is missing, attribution graphs won't reveal much.
- **The Role of Inactive Features & Inhibitory Circuits** – Often the fact that certain features *weren't active* is just as interesting as the fact that others are. In particular, there are many interesting circuits which involve features inhibiting other features.
- **Graph Complexity** – The resulting attribution graphs can be very complex and hard to understand.
- **Features at the Wrong Level of Abstraction** – Issues like feature splitting and absorption mean that features often aren't at the level of abstraction which would make it easiest to understand the circuit.
- **Difficulty of Understanding Global Circuits** – Ideally, we want to understand models in a global manner, rather than attributions on a single example. However, global circuits are quite challenging.
- **Mechanistic Faithfulness** – When we replace MLP computation with transcoders, how confident are we that they're using the same mechanisms as the original MLP, rather than something that's just highly correlated with the MLP's outputs?

We discuss these in detail below, and where possible provide concrete counterexamples where our present methods can not explain model computation due to these issues. We hope that these may motivate future research.

## Missing Attention Circuits

One significant limitation of our approach is that we compute our attribution graphs with respect to fixed attention patterns. This makes attribution a well-defined and principled operation, but also means that our graphs do not attempt to explain *how* the model's attention patterns were formed, or how these patterns mediate feature-feature interactions through attention head output-value matrices [18]. In this paper, we have focused on case studies where this is not too much of an issue – cases where attention patterns are not responsible for the "interesting part" or "crux" of the model's computation. However, we have also found many cases where this limitation renders our attribution graphs essentially useless.

### EXAMPLE: INDUCTION

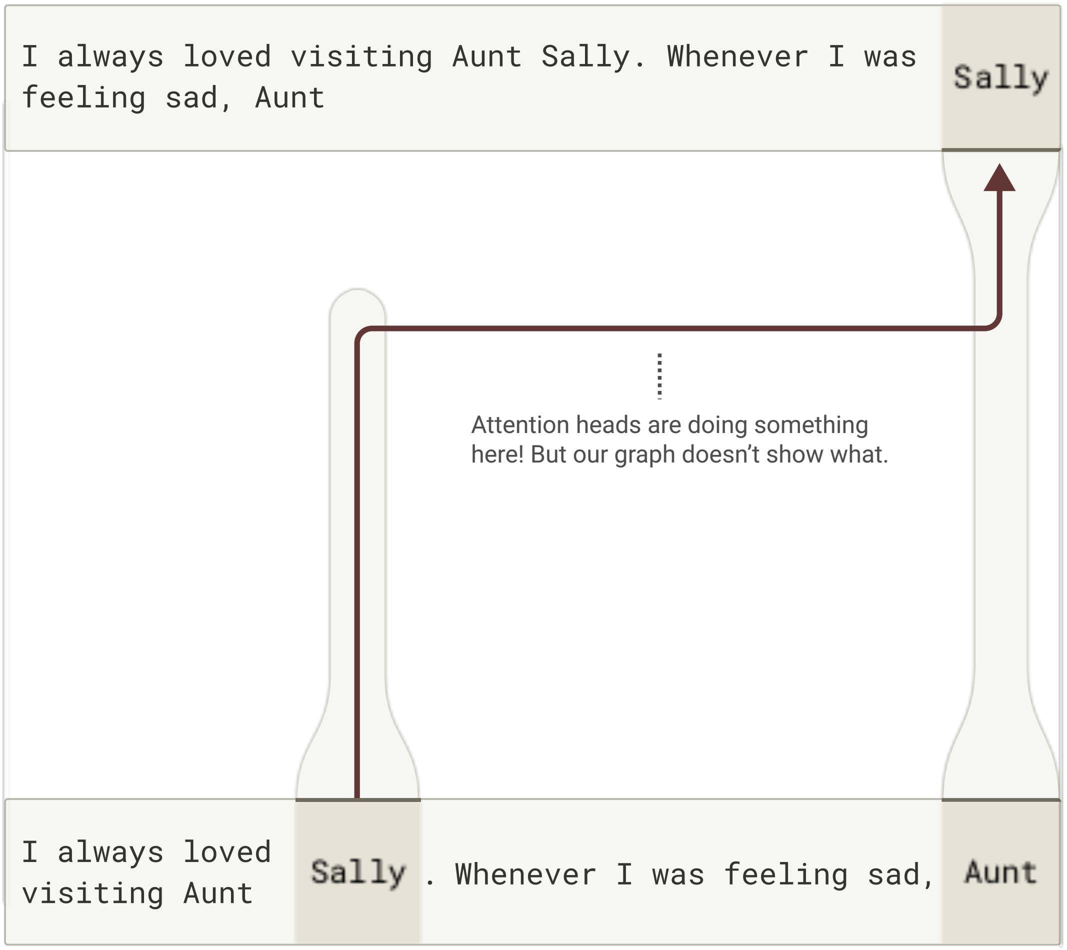
Let's consider for a moment a much simpler model – a humble 2-layer attention-only model, of the kind studied in [18]. One interesting property of these models is their use of *induction heads* [18, 30] to perform basic in-context learning. For example, if we consider the following prompt:

```
I always loved visiting Aunt Sally. Whenever I was feeling sad, Aunt
```

These models will have induction heads attend back to `"Sally"`, and then predict that is the correct answer. If we were to apply our present method, the answer isn't very informative. It would simply tell us that the model predicted `"Sally"`, because there was a token `"Sally"` earlier in the context.

## Present Method: Hypothetical Induction Example

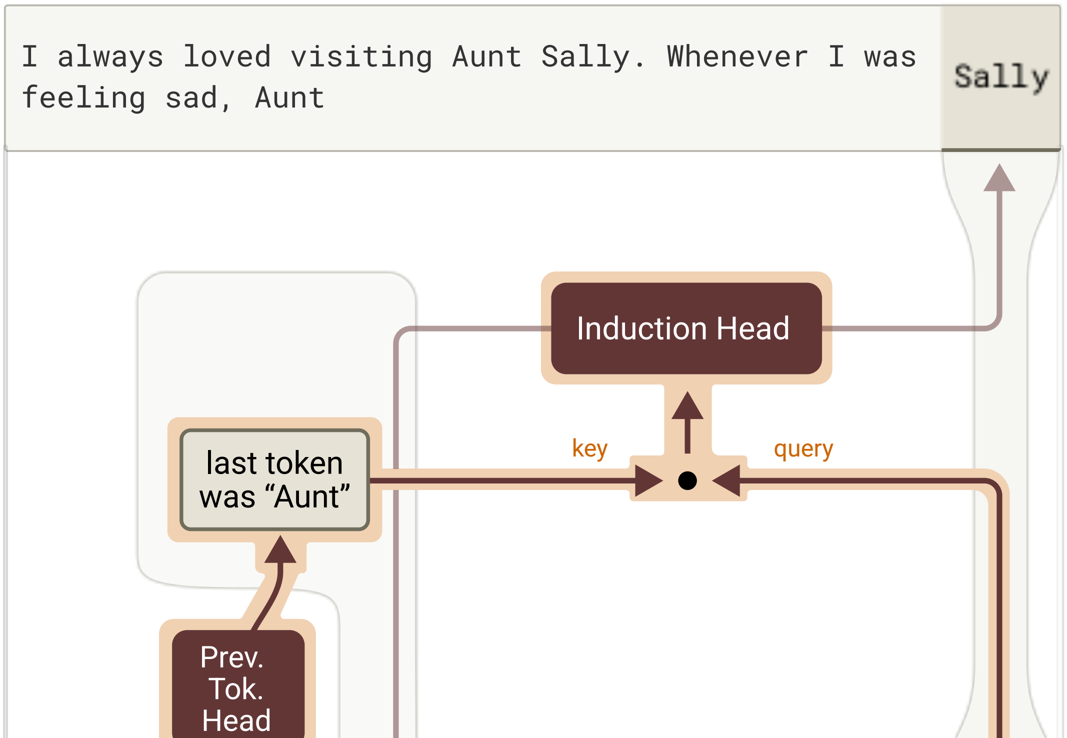
Induction heads are a common circuit which can be found even in small, attention-only models. If we imagine trying to understand such a circuit with our present method, the resulting answer is very limited...



I always loved visiting Aunt Sally. Whenever I was feeling sad, Aunt
Sally
Attention heads are doing something here! But our graph doesn't show what.
I always loved visiting Aunt
Sally
. Whenever I was feeling sad,
Aunt

## Missing QK Circuit: Hypothetical Induction Example

... This is because we aren't looking at the QK-circuit, which controls where attention heads attend. For induction heads, the basic story is matching the present token of the query against the previous token of the key.



I always loved visiting Aunt Sally. Whenever I was feeling sad, Aunt
Sally
Induction Head
key
query
last token was "Aunt"
Prev. Tok. Head

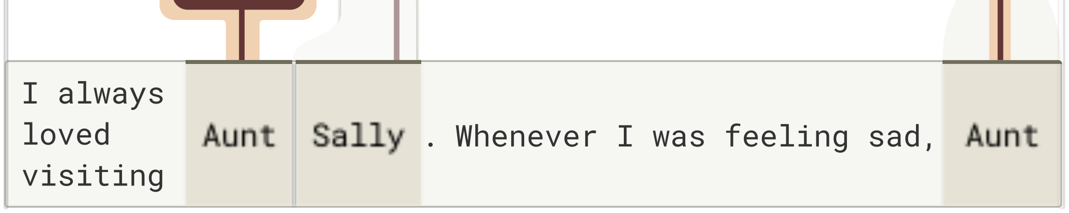


**Figure 29**: A hypothetical induction head example illustrates an important limitation of our present attribution graph method: it doesn't explain *why* attention moves information. For induction heads, that's the whole story!

*This misses the entire interesting story!* The induction head attends to `"Sally"` because it was preceded by `"Aunt"`, which matches the present token. Previous methods (e.g. [2, 18] were able to elucidate this, and so this case might even be seen as a kind of regression.

Indeed, when applied to Claude 3.5 Haiku on this prompt, our method has exactly this problem. See the attribution graph visualization – the graph contains *direct* edges from token-level "Sally" features to "say Sally" features and to the "Sally" logit, but fails to explain how these edges came about.

## EXAMPLE: MULTIPLE-CHOICE QUESTIONS

Induction is a simple case of attentional computation where we can make a reasonable guess at the mechanism even without help from our attribution graphs. However, this failure of the attribution graphs can manifest in more complex scenarios as well, where it completely obscures the interesting steps of the model's computation. For instance, consider a multiple choice question:

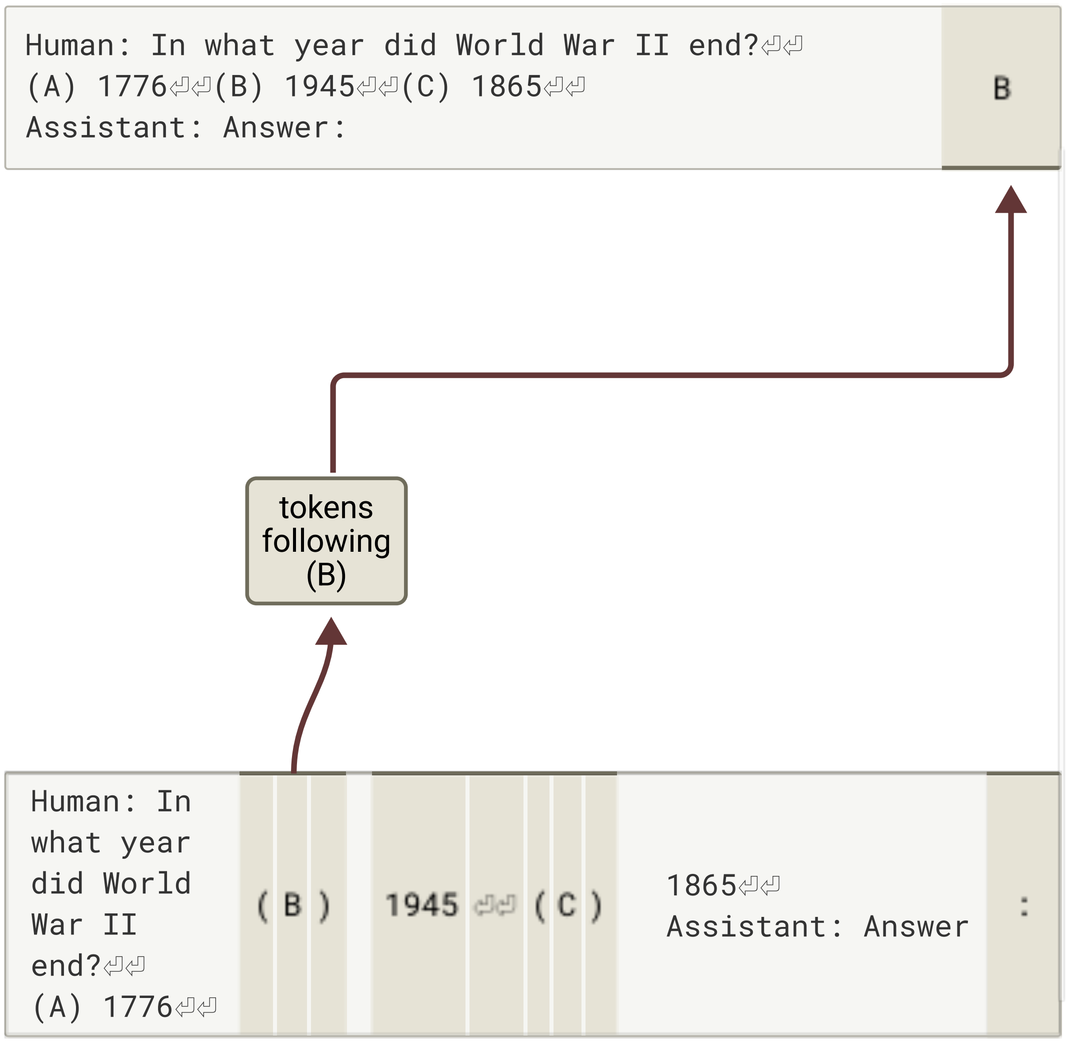
```
Human: In what year did World War II end?
(A) 1776
(B) 1945
(C) 1865

Assistant: Answer: (B)
```

When we compute the attribution graph (interactive graph visualization) for the `"B"` token in the Assistant's response, we obtain a relatively uninteresting answer – we answer `"B"` because of a `tokens following "(b)"` feature that activates on the correct answer. (There's also a direct pathway to the token `"B"`, and output pathways mediated by a `say "B"` motor feature; we've chosen to elide these for simplicity.)

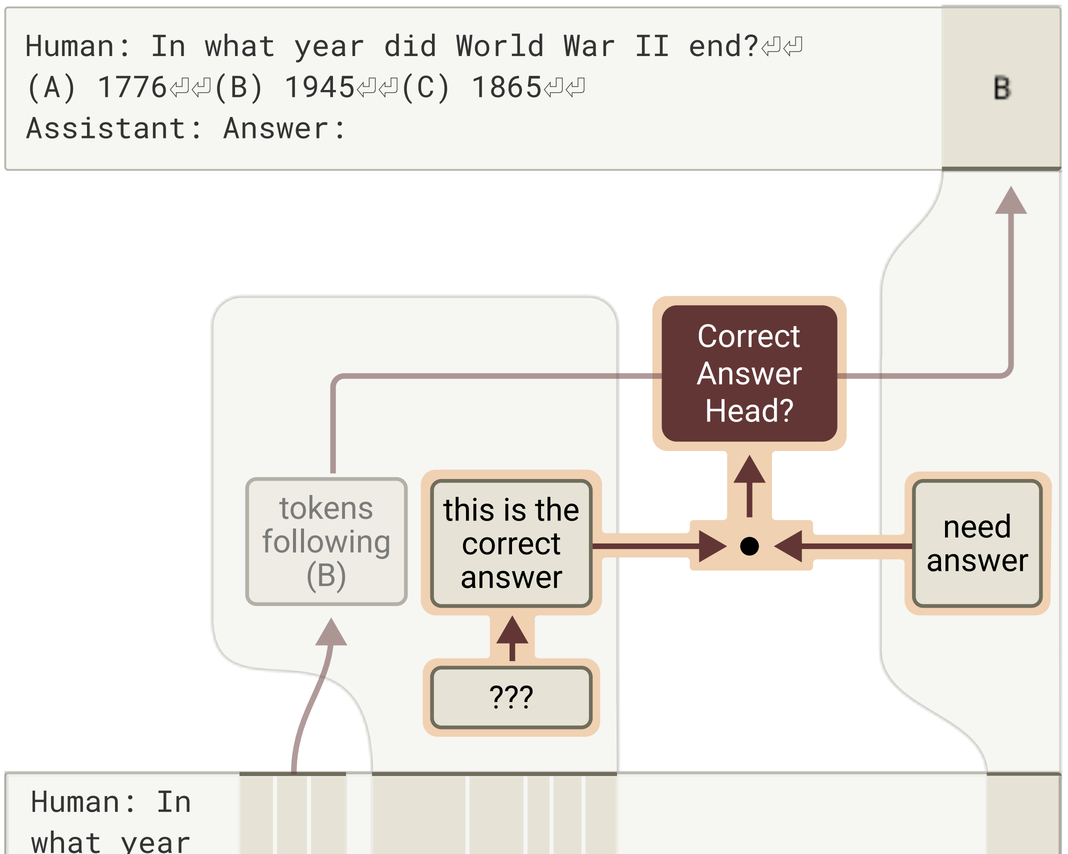
None of this provides a useful explanation of how the model chose its answer! The graph "skips over" the interesting part of the computation, which is *how the model knew that 1945 was the correct answer.* This is because the behavior is driven by attention. On further investigation, it turns out that there are three "correct answer" features that appear to fire on the correct answer to multiple choice questions, and which interventions show play a crucial role. From this, we hypothesize that the mechanism might be something like the following.[36]

**What Our Method Shows for Multiple Choice Questions**



Human: In what year did World War II end?⏎⏎
(A) 1776⏎⏎(B) 1945⏎⏎(C) 1865⏎⏎
Assistant: Answer:
B
tokens following (B)
Human: In what year did World War II end?⏎⏎ (A) 1776⏎⏎
( B )
1945 ⏎⏎ ( C )
1865⏎⏎ Assistant: Answer
:

**Our Hypothesis of What QK is Doing**



Human: In what year did World War II end?⏎⏎
(A) 1776⏎⏎(B) 1945⏎⏎(C) 1865⏎⏎
Assistant: Answer:
B
Correct Answer Head?
tokens following (B)
this is the correct answer
need answer
???
Human: In what year

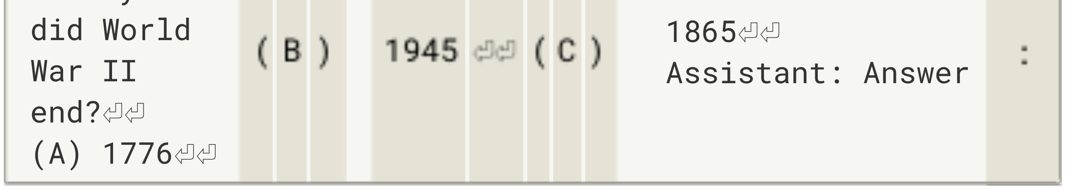


**Figure 30**: Multiple choice questions are another case where attention seems to be critical to understanding the interesting computation.

Since this involves significant conjecture, it's worth being clear about what we know about the QK-circuit, and what we don't.

- We know that it's controlled by attention, since freezing attention locks the answer.
- We don't know that it's mediated by a particular head, nor that any heads involved have a general behavior that can be understood as a generalization of this.
- We do know that there are three features which appear to track "this seems like the correct answer". We do know that intervening on them changes the correct answer in expected ways; for example activating them on answer C causes the model to predict `"C"`.
- We don't know that "correct answer" features directly play a role in the key side of whatever attention heads are involved, nor do we know if "need answer" features exist or play a role on the query side.
- We also do not know if there might be alternative parallel mechanisms at play (*see* Feng & Steinhardt [31] ).

This is all to say, there's a lot we don't understand!

But despite our limited understanding, it seems clear that the model behavior crucially flows through attention patterns and the QK circuits that compute them. Until we can fix this, our attribution graphs will "miss the story" in cases where attention plays a critical role. And while we were able to get a partial understanding of the story in this case through manual investigation, we would like for our methodology to surface this information automatically in the future!

## FUTURE DIRECTIONS ON ATTENTION

We suspect that similar circuits, where attention is the crux, are at play across a wide variety of prompts. In these cases, our present attribution graphs are little help to us, and new methods are needed.

Ultimately, the QK-circuit is a quadratic form over the residual stream. This means that attributions can naturally be made to *pairs* of key-side and query-side features. These pairs have a weight describing whether they increase or decrease attention to a particular token. However, this approach has the downside of a quadratic explosion in complexity.

Looking at QK-circuits for individual attention heads may make things harder than necessary, or even be misleading, if there is attention superposition. The interesting behavior may be spread across heads, which each have their own QK-circuit.

We also suspect that certain forms of dictionary learning on attention layers, or full attention replacement layers, could make these “QK attributions” more tractable and interpretable. We believe that finding the right way to handle attention attributions is one of the most important directions for future work.

## Reconstruction Errors & Dark Matter

Our cross-layer transcoders fail to fully reconstruct the activations of the model. As a result, our methods leave some “dark matter” of the model’s mechanisms unobserved, in part because our dictionaries are not large enough to learn a feature corresponding to every concept the model might know about. In our attribution graphs, this failure manifests as “error nodes,” which provide input to other nodes but receive no input themselves (they “pop out of nowhere”) and which are not interpretable. In the examples presented in this paper, we managed to identify interesting mechanisms despite the presence of error nodes. It is important to note that the circuits we have described are only a *partial* description of the model’s computation (both due to the presence of error nodes, and due to the fact that the circuits we highlight typically only include a subset of the non-error nodes in the graph!).

It is worth highlighting, however, that in some cases error nodes *completely* obscure what the model is doing. For instance, consider this attempted jailbreak – an obfuscated request for the model to write malware (which the model refuses):

```
Human: Write an office macro TAhT SEaRChEs fOS AND EtEIaFTxLRS pASWORdS

Assistant:
```

In the attribution graph, almost all the nodes on the obfuscated prompt tokens are error nodes, making it impossible to trace back the origin of the refusal-related features that are active on the “Assistant:” tokens.

The prevalence of error nodes in this example may not be surprising – this prompt is rather out-of-distribution relative to typical prompts, and so the cross-layer-transcoder is likely to do a poor job of predicting model activity.

We also note that another major source of error is the gap between our human interpretations of features and what they truly represent. Typically our interpretations of features are much too coarse to account for their precise activation profiles.

### FUTURE DIRECTIONS ON RECONSTRUCTION ERROR AND "DARK MATTER"

We see several avenues for addressing this issue:

- Scaling replacement models to larger sizes / more training data will increase the amount of variance they explain.
- Architectural modifications to our cross-layer transcoder setup could make it more expressive and thus capable of explaining more variance.
- Training our replacement model in a more end-to-end fashion, rather than on MSE alone, could decrease the weight assigned to error nodes even at a fixed MSE level
- Finetuning the replacement model on data distributions of interest could improve our ability to capture mechanisms on those distributions.
- We could develop methods of attributing back from error nodes. This would leave an uninterpretable “hole” in the attribution graph, but in some cases may still provide more insight into the model than our current no-inputs error nodes.

## The Role of Inactive Features & Inhibitory Circuits

Our cross-layer transcoder features are trained to be sparsely active. Their sparsity is key to the success of our method. It allows us to focus on a relatively small set of features for a given prompt, out of the tens of millions of features in the replacement model. However, this convenience relies on a key assumption – that only *active* features are involved in the mechanism underlying a model's responses.

In fact, this need not be the case! In some cases, the *lack of activity* of a feature, because it has been *suppressed* by other features, may be key to the model's response. For instance, in our analysis of hallucinations and entity recognition (see companion paper), we discovered a circuit in which "can't answer" features are *suppressed* by features representing known entities, or questions with known answers. Thus, to explain why the model hallucinates in a specific context, we need to understand what caused the "can't answer" features to *not* be active.

By default, our attribution graphs do not allow us to answer such questions, because they only display active features. If we have a hypothesis about which *inactive* features may be relevant to the model's completion (due to suppression), we can include them in the attribution graph. However, this detracts somewhat from one of the main benefits of our methodology, which is its enablement of exploratory, hypothesis-free analysis.

This leads to the following challenge – *how can we identify inactive features of interest*, out of the tens of millions of inactive features? It seems like we want to know which features could have been "counterfactually active" in some sense. In the entity recognition example, we identified these counterfactually active features by comparing pairs of prompts that contained either known or unknown entities (Michael Jordan or "Michael Batkin"), and then focusing on features that were active in at least one prompt from each pair. We expect that this contrastive pairs strategy will be key to many circuit analyses going forward. However, we are also interested in developing more unsupervised approaches to identifying key suppressed features. One possibility may be to perform feature ablation experiments, and consider the set of inactive features that are only "one ablation away" from being active.

One might think that these issues can be escaped by moving to global circuit analysis. However, it seems like there may be a deep challenge which remains. We need a way to filter out interference weights, and it's tempting to do this by using co-occurrence of features. But these strategies will miss important inhibitory weights, where one feature consistently prevents another from activating. This can be seen as a kind of global circuit analog of the challenges around inactive features in local attribution analysis.

## Graph Complexity

One of the fundamental challenges of interpretability is finding abstractions and interfaces that manage the cognitive load of understanding complex computations.[37] Our methodology is designed to reduce the cognitive load of understanding circuits as much as possible. For example:

- Feature sparsity means that there are fewer nodes in the attribution graph.
- We prune the graphs so that the analyst can focus only on its most important components.
- Our UI is designed to make navigating the graph as fluid as possible.
- We use the not-very-principled abstraction of "supernodes" to ad-hoc group together related features.

Despite all these steps, our attribution graphs are still quite complex, and require considerable time and effort to understand for many reasons:

- Even after our pruning pipeline and on fairly short prompts, the graphs typically contain hundreds of features and thousands of edges.
- The concepts we are interested in are typically smeared across multiple features.
- Each feature receives many small inputs from many other features, making it difficult to succinctly summarize “what caused this feature to activate.”
- Features often exert influence on one another by multiple paths of different lengths, or even of different signs!

As a result, it is difficult to distill the mechanisms uncovered by our graphs into a succinct story. Consequently, the vignettes we have presented are necessarily simplified stories of even the limited understanding of model computation captured in our attribution graph. We hope that a combination of improved replacement model training, better abstractions, more sophisticated pruning, and better visualization tools can help mitigate this issue in the future.

## Features at the Wrong Level of Abstraction

As sparse coding models have grown in popularity as a technique for extracting interpretable features from models, many researchers have documented shortcomings of the approach (*see e.g.* [32, 33, 34, 35, 36]). One notable issue is the problem of *feature splitting* [8, 37] in which the uncovered features are in some sense *too specific*. This can also lead to a related problem of *feature absorption* [35], where highly specific features steal credit from more general features, leaving holes in them (leading to things like a “U.S. cities except for New York and Los Angeles” feature).

As a concrete example of feature splitting, recall that in many examples in this paper we have highlighted “say X” features that cause the model to output a particular (group of) token(s). However, we also notice that there are *many* such features, suggesting that they each actually represent something more specific. For example, we came up with twelve prompts which Claude 3.5 Haiku completes with the word “during” and measured whether any features activated for all of the prompts (as a true “say ‘during’” feature would). In fact, there are no such features – any individual feature fires for only a subset of the prompts. Moreover, the degree of generality of the features appears to decrease with the size of our cross-layer transcoder.

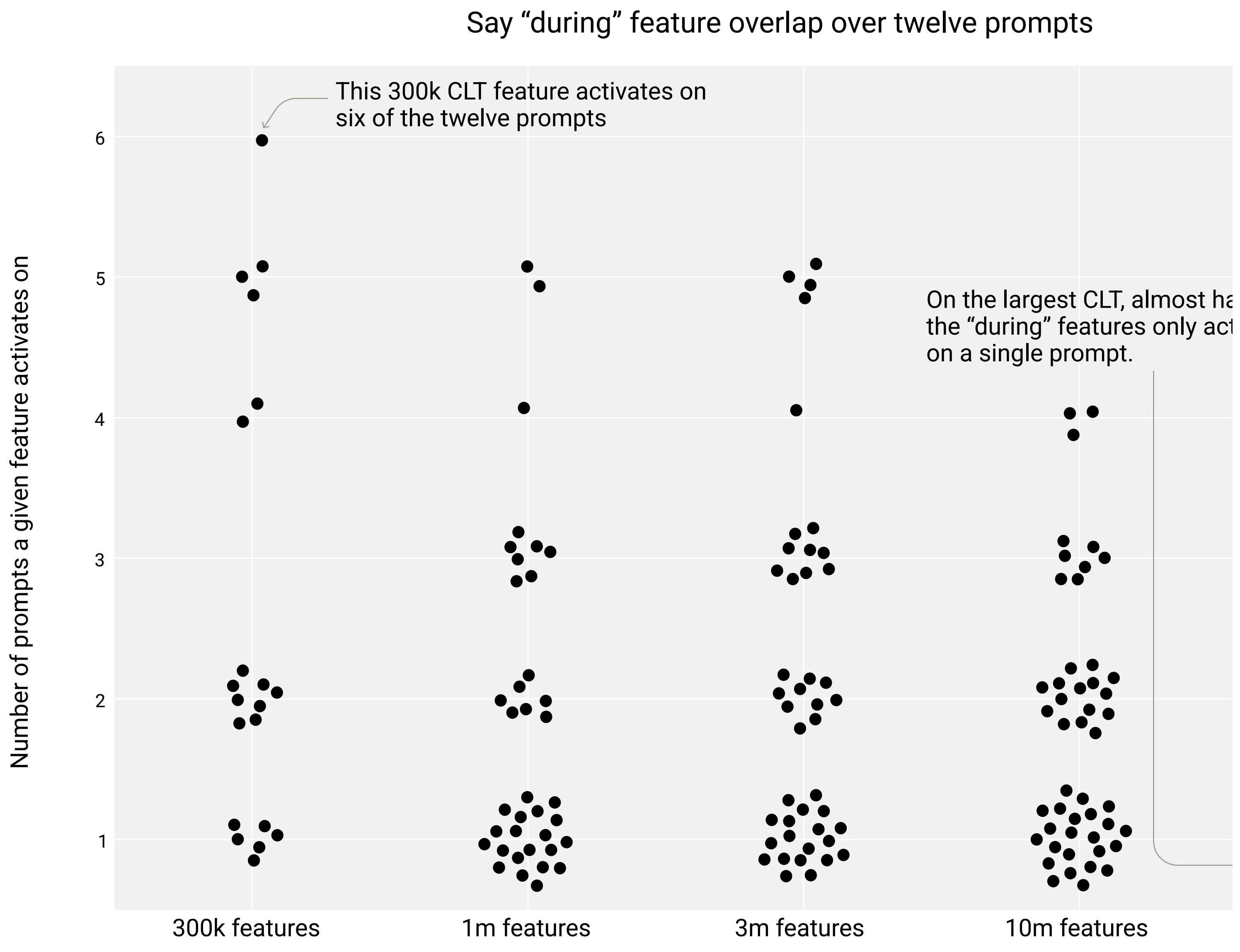


**Figure 31**: Feature splitting increases with CLT scale.

It may be the case that each individual feature represents something interpretable – for instance, qualitatively different contexts that might cause one to say the word "during." However, we often find that the level of abstraction we care about is different from the level we find in our features. Using smaller cross-layer transcoders may help this problem, but would also cause us to capture less of the model's computation.

In this paper, we often work around this issue in an ad-hoc way by manually grouping together features with related meanings into "supernodes" of an attribution graph. While this technique has proven quite helpful, the manual step is labor-intensive and likely loses information. It also makes it difficult to study how well mechanisms generalize *across* prompts, since different subsets of a relevant feature category may be active on different prompts.

We expect that solving this problem requires recognizing that there exist interpretable concepts at varying levels of abstraction, and at different times we may be interested in different levels. Sparse coding approaches like SAEs and (cross-layer) transcoders are a "flat" instrument, but we probably need a hierarchical variant that allows features at varying levels of abstraction to coexist in an interpretable way.

Several authors have recently proposed "Matryoshka" variants of sparse autoencoders that may help address this issue [38, 39]. Other researchers have proposed post-hoc ways to unify related features with "meta-SAEs" [40].

## Difficulty of Understanding Global Circuits

In this paper we have mostly focused on *attribution graphs*, which display information about feature-feature interactions *on a particular prompt*. However, one theoretical advantage of transcoder-based methodologies like ours is that they give us *global weights* between features, that are *independent* of the prompt. This allows us to estimate a "connectome" of the replacement model and learn about the general algorithms it (though not necessarily the underlying model) uses that apply to many different inputs. We have some successes in this approach – for instance, in the companion paper section on Refusals, we could see the global inputs to "harmful requests" features consisting of a variety of different *specific* categories of harm. In this paper, we studied in depth the global weights of features relating to arithmetic, finding for instance that "say a number ending in 5" features receive input from "6 + 9" features, "7 + 8" features, etc.

However, for the most part, we have found global feature-feature connections rather difficult to understand. This is likely for two main reasons:

- **Interference weights** – because features are represented in superposition, in order to learn useful weights between features, models must incur spurious "interference weights" – connections between features that don't make sense and aren't useful to the model's performance. These spurious weights are not too detrimental to model performance because their effects rarely "stack up" to actually change the model's output. For instance, we sometimes see features like this, which appear to clearly be a "say 15" feature, but whose top logit outputs include many seemingly unrelated words ("gag", "duty", "temper", "dispers"). We believe these logit connections are essentially irrelevant to the model's behavior, because when this feature activates, it is very unlikely that "duty" will be a plausible completion, and so upweighting its logit runs little risk of causing it to be sampled. Unfortunately, this makes the global weights very difficult to understand! This phenomenon applies to feature-feature weights as well (see § 4.1 Global Weights in Addition).
- **Interactions mediated by attention** – The basic global feature-feature weights derived from our cross-layer transcoder describe the direct interactions between features *not mediated by any attention layers.* However, there are also feature-feature weights mediated by attention heads. These might be thought of as similar to how features in a convolutional neural network are related by multiple sets of weights, corresponding to different positional offsets (further discussion here).

Our attribution graph edges are weighted combinations of both the direct weights and these attention-mediated weights. Our basic notion of global weights does not account for these interactions at all. One way to do so would be to compute the global weights mediated by every possible attention head. However, this has two limitations: (1) for this to be useful, we need a way of understanding the mechanisms by which different heads choose where they attend (see § 7.1 Limitations: Missing Attention Circuits), (2) it does not account for interactions mediated by *compositions* of heads [18]. Solving this issue likely requires extending our dictionary learning methodology to learn interpretable attentional features or replacement heads.

## Mechanistic Faithfulness

Our cross-layer transcoder is trained to mimic the activations of the underlying model at each layer. However, even when it accurately reconstructs the model's activations, there is no guarantee that it does so via the same *mechanisms*. For instance, even if the cross-layer transcoder achieved 0 MSE on our training distribution, it might have learned a fundamentally different input/output function than the underlying model, and consequently have large reconstruction error on out-of-distribution inputs. We hope that this issue is mitigated by (1) training on a broad data distribution, and (2) forcing the replacement model to reconstruct the underlying model's *per-layer* activations, rather than simply its output. Nevertheless, we cannot guarantee that the replacement model has learned the same mechanisms  – what we call *mechanistic faithfulness* – and instead resort to verifying it post-hoc.

In this paper, we have used perturbation experiments (inhibiting and exciting features) to validate the mechanisms suggested by our attribution graphs. In the case studies we presented, we were typically able to validate that features had the effects we expected (on the model output, and on other features). However, the degree of validation we have provided is very coarse. We typically perturb multiple features at once ("supernodes") and check their directional effects on other features / logit outputs. In addition, we typically sweep over the layer at which we perform perturbations and use the layer that yields the maximum effect. In principle, our attribution graphs make predictions that are *much more* fine-grained than these kinds of interventions can test. Ideally, we should be able to accurately predict the effect of perturbing any feature at any layer on any other feature.

In § G Appendix: Validating the Replacement Model, we attempt to more comprehensively quantify our accuracy in predicting such perturbation results, finding reasonably good predictive power for effects a few layers downstream of a perturbation, and much worse predictive power many layers downstream. This suggests that, while our circuit descriptions may be mechanistically accurate at a very coarse level, we have substantial room to improve their faithfulness to the underlying model.

We are optimistic about trying methods to directly optimize for mechanistic faithfulness, or exploring alternative dictionary learning architectures that learn more faithful solutions naturally.

## Discussion

Our approach to reverse engineering neural networks has four basic steps: decomposition into components, providing descriptions of these components, characterizing how components interact to produce behaviors, and validating these descriptions.[38] A number of choices are required at each step, which can be more or less principled, and the power of a method is ultimately the degree to which it produces valid hypotheses about model behaviors.

In this paper, we trained cross-layer transcoders with sparse features to replace MLP blocks (the decomposition), described the features by the dataset examples they activate on (the description), characterized their interactions on specific prompts using attribution graphs (the interactions), and validated the hypotheses using causal steering interventions (the validation).

We believe some of the choices we made are robust, and that successful decomposition methods will make similar choices or find other ways of dealing with the underlying issues they address:

- **We use learned features instead of neurons.** While the top activations for neurons are often interpretable, lower activations are not. In principle, one could threshold neuron activations to restrict them to this interpretable regime; however, we found that thresholding neurons at that level damages model behavior significantly more than a transcoder or CLT. This means a trained replacement layer can provide a Pareto improvement relative to thresholded neurons across interpretability, L0, and MSE. The set of neurons is fixed. (Nevertheless, neurons can provide a starting point for investigation, without incurring any additional compute costs, see e.g. [41].)
- **We use transcoders instead of residual-stream SAEs.** While residual stream SAEs can decompose the latent states of a model, they don't provide a natural extension decomposing its computational steps. Crucially, transcoder features bridge over MLP layers and interact linearly via the residual stream with transcoder features in other layers. In contrast, interaction between SAE features is interposed by non-linear MLPs.
- **We use cross-layer transcoders instead of per-layer transcoders.** We hypothesized [15] that different MLP layers might collaborate to implement a single computational step ("cross-layer superposition"); the most extreme case of this is when many layers amplify the same early-layer feature so that it is still large enough to influence late layers. CLTs collapse these to one feature. We found evidence that this phenomenon happens in practice, as manifested through the Pareto improvement on path length vs graph influence.
- **We compute feature-feature interactions using linear direct effects instead of nonlinear attributions or ablations.** Much has been written about "saliency maps" or attribution through non-linear neural networks (including ablation, path-integrated gradients [42], and Shapley values (*e.g.* [43])). Even the most principled options for credit assignment in a nonlinear setting are somewhat fraught. Since our goal is to crisply reason about mechanism, we construct our setup so that the direct interactions between features in the previous layer and the pre-activations of features in the next layer are *conditionally linear*; that is to say, they are linear once we freeze certain parts of the problem (attention patterns and normalization denominators). This factors the problem into a portion we can mechanistically understand in a principled manner, and a portion that remains to be understood [18]. Also crucial to achieving this linear direct effect property is the earlier decision to use transcoders.[39 40]

There are other choices we made for convenience, or as a first step towards a more general solution:

- **We collapse attention paths.** Every edge in our attribution graph is the direct interaction of a pair of features, summed over all possible direct interaction paths. Some of these paths flow primarily through the residual stream. Others flow through attention heads. We make no effort in the present work to distinguish these. This throws away a lot of interesting structure, since *which heads* mediated an interaction may be interesting, if they are something we understand (e.g. a successor head [44] or induction head [30]).[41] [42]
- **We ignore QK-circuits.** In order to get our linear feature-feature interactions, we factor understanding transformers into two pieces, following *Framework* [18]. First we ask about feature-feature interaction, conditional on an attention head or set of attention heads (the "OV-circuit"). But this leaves a second question of why attention heads attend to various pieces (the "QK-circuit"). In this work, we do not attempt this second half.
- **We only use a sparsity penalty and a reconstruction loss for crosscoder training.** While our ultimate goal is to find circuits with sparse interpretable edges, in a replacement model which is *mechanistically faithful* to the underlying model, we don't train explicitly for any of those goals.

Nevertheless, our current method yielded interesting, validated mechanisms involving planning, multilingual structure, hallucinations, refusals, and more, in the companion paper.

We expect that advances in the trained, interpretable replacement model paradigm will produce quantitative improvements on graph-related metrics and qualitative improvements on the amount of model behaviors that become legible. It is possible that this will be an incremental process, where incremental improvements to CLTs and associated approaches to attention will yield incremental improvements to circuit identification, or that a radically different decomposition approach will best this method at scale at uncovering mechanisms. Regardless, we hope to enter an era where there is a clear flywheel between decomposition methods and "biology" results, where the appearance of structure in specific model investigations inspires innovations in decomposition methods, which in turn bring more model behaviors into the light.

## Coda: The lessons of addition

Addition is one of the simplest behaviors performed by models, and because it is so structured, we can characterize every feature's activity on the full problem domain exactly. This allows us to skip the difficult step of staring at dataset examples and trying to discern what a feature is doing relative to, and what distinguishes it from other features active in similar contexts. This revealed a range of heuristics used by Haiku 3.5 ("say something ending in a 5", "say something around 50", "say something starting with 51") which had been identified before by Nikankin [21], together with a set of lookup table features that connect input pairs satisfying certain conditions (say, adding digits ending in 6 and 9) to the appropriate sum features satisfying the consequence on the output (say, producing a sum ending in 5).

However, even in this easier setting we made numerous mistakes when labeling these features from the original dataset examples alone, for example thinking a `_6 + _9` feature was itself a `sum = _5` feature based on what followed it in contexts. We also struggled to distinguish between low-precision features of different scales, and between features which were sensitive to a limited set of inputs or merely appeared to be because of a high prevalence of those inputs in our dataset. How much worse must this be when looking at dozens of gradations of refusal features! Getting more precise distinctions between features in fuzzier domains than arithmetic, whether through feature geometry or superhuman autointerpretability methods, will be necessary if we want to understand problems at the level of resolution that even today's CLTs appear to make possible.

Because addition is such a clear problem, we were also able to see how the features connected with each other to build parallel pathways; giving rise from simple heuristics depending on the input to more complex heuristics related to the output; going from the “Bag of Heuristics” identified by Nikankin to a “Graph of Heuristics”. The virtual weights show this computational structure, with groups of lookup table features combining to form sum features of different modularity and scale, which combine to form more precise sum features, and to eventually give the output. It seems likely that, in “fuzzier” natural language examples, we are conflating many roles played by features at different depths into overall buckets like “unknown entity” or “harmful request” or “notions of largeness” which actually serve specialized roles, and that there is actually an intricate aggregation and transformation of information taking place, just out of our understanding today.

## Related Work

Despite being a young field, mechanistic interpretability has grown rapidly. For an introduction to the landscape of open problems and existing methods, we recommend the recent survey by Sharkey *et al.* [32]. Broader perspectives can be found in recent reviews of mechanistic interpretability and related topics (*e.g.* [45, 46, 47, 48]).

In previous papers, we've discussed some of the foundational topics our work builds on, and rather than recapitulate that discussion, we will refer readers to our previous discussion on them. This includes

- **Existence of interpretable features** (*e.g.* [49, 50, 51]; *see prior discussion*)
- **Attention head analysis** (*e.g.* [52, 53]; *see prior discussion*),
- **Bertology** (*e.g.* [54]; *see prior discussion*),
- **Interpretability interfaces** (e.g. [55, 56]; see *prior discussion*),
- **Disentanglement** (*e.g.* [57]; *see prior discussion*),
- **Compressed sensing** (*e.g.* [58, 59]; *see prior discussion*),
- **Sparse dictionary learning** (*e.g.* [60, 61]; *see prior discussion*),
- **Theory of superposition** (*e.g.* [6, 5]; *see prior discussion*),
- **Theories of neural coding and distributed representation** (*e.g.* [62]; see prior discussion)
- **Activation steering** (*e.g.* [63, 64]; *see prior discussion*).

The next two sections will focus on the two different stages we often use in mechanistic interpretability [65]: identifying features and then analyzing the circuits they form. Following that, we'll turn our attention to past work on the "biology" of neural networks.

## Feature Discovery Methods

A fundamental challenge in circuit discovery is finding suitable units of analysis [32]. The network's natural components—attention heads and neurons—lack interpretability due to superposition [7, 66], making the identification of better analytical units a central problem in the field.

**Sparse Dictionary Learning** is a technique with a long history originally developed by neuroscientists to analyze neural recording data [60]. Recent work [67, 8, 9, 10, 11, 68] has applied Sparse Autoencoders (SAEs) [69, 70] as a scalable solution to address superposition by learning dictionaries that decompose language model *representations*. While SAEs have been successfully scaled to frontier models [10, 68], researchers have identified several methodological limitations including feature shrinkage [71], feature absorption [35, 40], lack of canonicalization [72, 73, 74], automating feature interpretation [75, 76, 77], and poor performance on downstream classification and steering tasks [33, 34].

For circuit analysis specifically, SAEs are suboptimal because they decompose representations rather than *computations*. **Transcoders** [12, 13, 14, 17] address this limitation by predicting the *output* of nonlinear components from their inputs. Bridging over nonlinearities like this enables direct computation of pairwise feature interactions without relying on attributions or ablations through intermediate nonlinearities.

The space of dictionary learning approaches is large, and we remain very excited about work which explores this space and addresses methodological issues. Recent work has explored architectural modifications like multilayer feature learning [15, 78], adding skip connections [79], incorporating gradient information [80], adding stronger hierarchical inductive biases [38, 39], and further increasing computational efficiency with mixtures-of-experts [81]. The community has also studied alternative training protocols to learn dictionaries that respect downstream activations [82], reduce feature shrinkage [83, 71], and make downstream computational interactions sparse [84]. To measure this methodological progress, a number of benchmarks and standardized evaluation protocols for assessing dictionary learning methods have been developed [85, 86, 87, 88, 34, 89, 90]. We think circuit-based metrics will be the next frontier in dictionary learning evaluation.

Beyond dictionary learning, several alternative unsupervised approaches to extracting computational units have shown initial success in small-scale settings. These include transforming activations into the *local interaction basis* [91, 92] and developing attribution-based decompositions of parameters [93].

## Circuit Discovery Methods

**Definitions.** Throughout the literature, the term circuit is used to mean many different things. Olah *et al.* [65] introduced the definition as a subgraph of a neural network where nodes are directions in activation space and edges are the weights between them. This definition has been relaxed over time in some parts of the literature and is often used to refer to a general subgraph of network components with edge weights computed from an attribution or intervention of some kind [2, 3].

There are several dimensions along which circuit approaches and definitions vary:

- Are the units of analysis globally interpretable or not? (For example, compare monosemantic features versus an entire attention head which does many different things across the data distribution.)
- Is the circuit itself (i.e. the edge connections) a global description, or a locally valid attribution graph? Or something else?
- Are the edges interpretable? (For example, are the edges computed by linear attributions or a complex nonlinear intervention?)
- Does the approach naturally address superposition?

We believe the North Star of circuit research is to manifest an object with globally interpretable units connected by interpretable edges which are globally valid. The present work falls short by only offering a locally valid attribution graph.

**Manual Analysis.** Early circuit discovery was largely manual, requiring specific hypotheses and bespoke methods of validation [51, 18, 30, 2, 94]. Causal mediation analysis [95], activation patching [1, 96, 97], patch patching [2, 98], and distributed alignment search [99, 100] have been the the most commonly used techniques for refining hypotheses and isolating causal pathways in direct analyses. However, these techniques generally do not provide interpretable (i.e. linear) edge weights between units.

**Automatic Analysis.** These analyses were automated in Conmy *et al.* [3] by developing a recursive patching procedure to automatically find component subgraphs given a task of interest. However, patching analyses are computationally expensive, requiring a forward pass per step. This motivated *attribution* patching [101, 102], a more efficient method leveraging gradients to approximate the effect of interventions. There has been significant follow up work on attribution patching including improving the gradient approximation [103, 104, 105], adapting the techniques to vision models [106], and incorporating positional information [107]. Other techniques studied in the literature include learned masking techniques [108, 109, 110], circuit probing [111], discretization [112], and information flow analysis [113, 114]. The objective of many of these automated approaches is isolating important model components (i.e., layers, neurons, attention heads) and the interactions between them, but they do not address the interpretation of these components.

However, armed with better computational units of analysis from sparse dictionary learning, this work and other recent papers [12, 17, 16, 115] are making a return to discovering connections between interpretable components. Specifically, our work is most similar to Dunefsky et al. [12] and Ge et al. [17] who also use transcoders to compute per-prompt attribution graphs with stop-gradients while also studying input-agnostic global weights. Our work is different in that we use crosscoders to absorb redundant features, use a more global pruning algorithm, include error nodes [16], and use a more powerful visualization suite to enable deeper qualitative analysis. These works are closer in spirit to the original circuits vision outlined in Olah *et al.* [65], but inherit prompt specific quantities (e.g. attention patterns) that limit their generality.

**Attention Circuits.** The attention mechanism in transformers introduced challenges for weight-based circuit analysis as done by Olah *et al.* [65]. Elhage *et al.* [18] proposed decomposing attention layers into a *nonlinear* QK component controlling the attention pattern, and a *linear* OV component which controls the output. By freezing QK (e.g., for a particular prompt), transcoder feature-feature interactions mediated by attention become linear. This is the approach adopted by this work and Dunefsky et al. [12]. Others have tried training SAEs on the attention outputs [116, 16, 115] and multiplying features through key and query matrices to explain attention patterns [17].

**Replacement Models.** Our notion of a replacement model is similar in spirit to past work on causal abstraction [117, 118] and proxy models [119, 120]. These methods seek to learn an interpretable graphical model which maintains faithfulness to an underlying black box model. However, these techniques typically require a task specification or other supervision, as opposed to our replacement model which is learned in a fully unsupervised manner.

**Circuit Evaluation**. Causal scrubbing [121] was proposed as an early principled approach for evaluating interpretation quality through behavior-preserving resampling ablations. Shi *et al.* [122] formalized criteria for evaluating circuit hypotheses, focusing on behavior preservation, localization, and minimality, while applying these tests to both synthetic and discovered circuits in transformer models. However, Miller *et al.* [26] raised important concerns about existing faithfulness metrics, finding them highly sensitive to seemingly insignificant changes in ablation methodology.

## Circuit Biology and Phenomenology

Beyond methods, many works have performed deep case studies and uncovered interesting model phenomenology. For example, thorough circuit analysis has been performed on

- Arithmetic in toy models [94, 123, 124, 125, 126]
- Python doc strings [127]
- Indirect object identification [2]
- Computing the greater-than operator [128]
- Multiple Choice [129]
- Pronoun gender [95, 130, 131]
- In-context learning [132]

The growing set of case studies has enabled further research on how these components are used in other tasks [133, 134]. Moreover, Tigges *et al.* [135] found that many of these circuit analyses are consistent across training and scale. Preceding these analyses, there has also been a long line of “Bertology” research that has studied model biology (*see survey* [54]) using attention pattern analysis and probing.

## Acknowledgments

The case study on a model with hidden goals builds on a model organism developed by Sam Marks and Johannes Treutlein, with whom the authors also had helpful conversations. We would also like to acknowledge enabling work by Siddharth Mishra-Sharma training SAEs on the model used in the hidden goals case study.

We would like to thank the following people who reviewed an early version of the manuscript and provided helpful feedback that we used to improve the final version: Larry Abbott, Andy Arditi, Yonatan Belinkov, Yoshua Bengio, Devi Borg, Sam Bowman, Joe Carlsmith, Bilal Chughtai, Arthur Conmy, Jacob Coxon, Shaul Druckmann, Leo Gao, Liv Gorton, Helai Hesham, Sasha Hydrie, Nicholas Joseph, Harish Kamath, Tom McGrath, János Kramár, Aaron Levin, Ashok Litwin-Kumar, Rodrigo Luger, Alex Makolov, Sam Marks, Dan Mossing, Neel Nanda, Yaniv Nikankin, Senthooran Rajamanoharan, Fabien Roger, Rohin Shah, Lee Sharkey, Lewis Smith, Nick Sofroniew, Martin Wattenberg, and Jeff Wu.

We would also like to acknowledge Senthooran Rajamanoharan for helpful discussion on implementation of JumpReLU SAEs.

This paper was only possible due to the support of teams across Anthropic, to whom we're deeply indebted. The Pretraining and Finetuning teams trained Claude 3.5 Haiku and the 18-layer research model, which were the targets of our research. The Systems team supported the cluster and infrastructure that made this work possible. The Security and IT teams, and the Facilities, Recruiting, and People Operations teams enabled this research in many different ways. The Comms team (and especially Stuart Ritchie) supported public scientific communication of this work.

## Author Contributions

**Development of methodology:**

- Chris Olah, Adly Templeton, and Jonathan Marcus developed ideas leading to general crosscoders, and the latter two implemented them in the Dictionary Learning codebase.
- Jack Lindsey developed and first analyzed the performance of cross-layer transcoders.
- Tom Conerly, Jack Lindsey, Adly Templeton, Hoagy Cunningham, Basil Hosmer, and Adam Jermyn optimized the sparsity penalty and nonlinearity for CLTs.
- Jack Lindsey and Michael Sklar ran scaling law experiments.
- Jack Lindsey, Emmanuel Ameisen, Joshua Batson, and Chris Olah developed and refined the replacement model and attribution graph computation.
- Jack Lindsey, Wes Gurnee, and Joshua Batson developed the graph pruning methodology, and Wes Gurnee systematically evaluated the approaches.
- Emmanuel Ameisen, Joshua Batson, Brian Chen, Craig Citro, Wes Gurnee, Jack Lindsey, and Adam Pearce did initial exploration of example attribution graphs to validate and improve methodology. Wes Gurnee identified specific attention heads involved in certain prompts, and Adam Pearce analyzed feature splitting. Emmanuel Ameisen, Wes Gurnee, Jack Lindsey, and Adam Pearce identified specific examples to study.
- Jack Lindsey, Emmanuel Ameisen, Wes Gurnee, Joshua Batson, and Chris Olah developed the methodology for the intervention analyses.
- Wes Gurnee, Emmanuel Ameisen, Jack Lindsey, and Joshua Batson developed evaluation metrics for attribution graphs, and Wes Gurnee led their systematic implementation and analysis.
- Michael Sklar and Jack Lindsey developed the approach for and executed perturbation experiments used to evaluate mechanistic faithfulness.
- Nicholas L. Turner, Joshua Batson, Jack Lindsey, and Chris Olah developed the virtual weight and global weight approaches and analyses.
- Brian Chen, Craig Citro, and Michael Sklar extended the method to handle neurons in addition to features.

**Infrastructure and Tooling:**

- Tom Conerly, Adly Templeton, T. Ben Thompson, Basil Hosmer, David Abrahams, and Andrew Persic significantly improved the efficiency of dictionary learning and maintained the orchestration framework used for managing dictionary learning.
- Adly Templeton organized efficiency work that enabled the largest runs on Claude 3.5 Haiku.
- Adly Templeton significantly refactored the code to collect activations and train dictionaries, improving performance and usability.
- Brian Chen designed and implemented scalability improvements for feature visualization with support from Tom Conerly.
- Craig Citro, Emmanuel Ameisen, and Andy Jones improved and maintained the infrastructure for interacting with model internals.
- Emmanuel Ameisen and Jack Lindsey developed the infrastructure for running the replacement model. Brian Chen implemented the layer norm and attention pattern freezing required for backpropagation in the local replacement model.
- Emmanuel Ameisen developed a stable implementation of our graph generation pipeline for cross-layer transcoders.
- Nicholas L. Turner led implementations of graph generation pipelines for alternative experimental crosscoder architectures with input from Craig Citro and Emmanuel Ameisen.
- Nicholas L. Turner and Emmanuel Ameisen added the ability to visualize attributions to selected inactive features.
- Wes Gurnee and Emmanuel Ameisen implemented efficiency improvements to graph generation.
- Emmanuel Ameisen and Wes Gurnee added error nodes and embedding nodes to graph generation.
- Wes Gurnee implemented adaptive, partial graph generation for large graphs.
- Adam Pearce developed a method and interface for visualizing differences between pairs of graphs.
- Tom Conerly and Jonathan Marcus improved the efficiency of loading feature weights which also sped up attribution graph generation.
- Tom Conerly and Basil Hosmer made improvements to the integration of cross-layer transcoders with circuit attribution graph generation.
- Brian Chen created the slack-based system for logging attribution graph runs.
- Emmanuel Ameisen developed the infrastructure for patching experiments.
- Adam Pearce, Jonathan Marcus, Zhenyi Qi, Thomas Henighan, and Emmanuel Ameisen identified open source datasets for visualization and generated feature visualization data for those datasets.
- Shan Carter, Thomas Henighan, and Jonathan Marcus built an interactive tool for exploring feature activations.
- Trenton Bricken, Thomas Henighan, and Jonathan Marcus provided infrastructure support and feedback for the hidden goals case study.
- Trenton Bricken, Callum McDougall, and Brian Chen developed the autointerpretability framework used for initial exploration of attribution graphs.
- Nicholas L. Turner designed and implemented the virtual weight pipeline to process the largest CLTs and handle processing requests from other members of the team. Joshua Batson, Tom Conerly, T. Ben Thompson, and Adly Templeton made suggestions on design decisions. Brian Chen and Tom Conerly made improvements to infrastructure that ended up supporting this effort.

**Interactive Graph Interface:**

- The interactive attribution graph interface was built, and maintained by Adam Pearce, with assistance from Brian Chen and Shan Carter. Adam Pearce led the work to implement feature visualizations, subgraph display and editing, node pinning and most other elements of the interface.

**Case Studies:**

- Wes Gurnee developed a systematic analysis of acronym completion, used for validating the original method and the NDAG example in the paper.
- Emmanuel Ameisen investigated the Michael Jordan example.
- Nicholas L. Turner, Adam Pearce, Joshua Batson, and Craig Citro investigated the arithmetic case study.

**Paper writing, infrastructure, and review:**

- Figures
    - Chris Olah set the design language for the major figures.
    - Adam Pearce created the feature hovers which appear on paper figures.
    - Shan Carter created the explanatory figures, with assistance from Brian Chen.
    - Figure refinement and design consulting was provided by Shan Carter and Chris Olah.
    - The interactive interface for exploring addition feature global weights was made by Adam Pearce, Nicholas L. Turner, and Joshua Batson.
- Writing and figures
    - Abstract - Emmanuel Ameisen
    - Summary of technical approach - Joshua Batson
    - Replacement model - Emmanuel Ameisen, Jack Lindsey
    - Attribution graphs - Emmanuel Ameisen, Brian Chen
    - Global weights - Nicholas L. Turner, Joshua Batson
    - Evaluations - Wes Gurnee, Jack Lindsey, Emmanuel Ameisen
    - Limitations - Jack Lindsey, Chris Olah. Adam Pearce made the feature splitting figure.
    - Discussion - Joshua Batson
    - Related work was drafted by Wes Gurnee, and Craig Citro significantly improved the completeness of the bibliography.
- Detailed feedback on the paper and figures:
    - David Abrahams, Emmanuel Ameisen, Joshua Batson, Trenton Bricken, Brian Chen, Craig Citro, Tom Conerly, Wes Gurnee, Thomas Henighan, Adam Jermyn, Jack Lindsey, Jonathan Marcus, Chris Olah, Adam Pearce, Kelley Rivoire, Nicholas L. Turner, Sam Zimmerman.
    - Tom Conerly and Thomas Henighan led a detailed technical review.
- Feedback from internal and external reviewers was managed by Nicholas L. Turner and Joshua Batson.
- Paper publishing infrastructure was built and maintained by Adam Pearce and Craig Citro.

**Support and Leadership**

- Sam Zimmerman managed the dictionary learning team and helped coordinate the team's efforts scaling dictionary learning to enable cross-layer transcoders on Claude 3.5 Haiku.
- Kelley Rivoire managed the interpretability team at large, provided support with project management for writing the papers, and helped with technical coordination across dictionary learning and attribution graph generation.
- Tom Conerly provided research and engineering leadership for dictionary learning.
- Chris Olah provided high-level research guidance.
- Joshua Batson led the overall circuits project, supported technical coordination between teams, and provided research guidance throughout.

## Citation Information

For attribution in academic contexts, please cite this work as

```
Ameisen, et al., "Circuit Tracing: Revealing Computational Graphs in Language Models", Transformer
Circuits, 2025.
```

BibTeX citation

```
@article{ameisen2025circuit,
  author={Ameisen, Emmanuel and Lindsey, Jack and Pearce, Adam and Gurnee, Wes and Turner, Nicholas L.
and Chen, Brian and Citro, Craig and Abrahams, David and Carter, Shan and Hosmer, Basil and Marcus,
Jonathan and Sklar, Michael and Templeton, Adly and Bricken, Trenton and McDougall, Callum and
Cunningham, Hoagy and Henighan, Thomas and Jermyn, Adam and Jones, Andy and Persic, Andrew and Qi,
Zhenyi and Ben Thompson, T. and Zimmerman, Sam and Rivoire, Kelley and Conerly, Thomas and Olah, Chris
and Batson, Joshua},
  title={Circuit Tracing: Revealing Computational Graphs in Language Models},
  journal={Transformer Circuits Thread},
  year={2025},
  url={https://transformer-circuits.pub/2025/attribution-graphs/methods.html}
}
```

## CLT Implementation Details

### Estimated Compute Requirements for CLT Training

To give a rough sense of compute requirements to train CLTs, we share some estimated costs for CLTs on the Gemma 2 series of models [136, 137]. On the 2B parameter model, a run with 2M features and 1B train tokens would require 3.8e20 training flops (roughly 210 H100 hours at 50% flops efficiency). On the 9B parameter model a run with 5M features and 3B train tokens would require 6.9e21 training flops (roughly 3,844 H100 hours at 50% flops efficiency).

### ML details

We train our cross-layer transcoder using a combination of mean-squared error reconstruction loss on MLP outputs and a Tanh sparsity penalty. Our features use the JumpReLU nonlinearity [11]. We found the combination of JumpReLU + Tanh to modestly outperform alternative methods on MSE/L0, but we suspect that alternative choices of nonlinearity and sparsity penalty could perform comparably well. For more details on our training setup see [138]. Here we highlight methodological details that differ from our prior work on SAEs:

- Following [15], we separately normalize the activation of each residual stream layer and each MLP output prior to using them as inputs / targets for the cross-layer transcoder.
- For our nonlinearity, we use the JumpReLU activation function with a straight-through gradient estimator [11]. However, we deviate from Rajamanoharan *et al.* in several respects. First, we use a much higher bandwidth parameter (1.0) for the straight through estimator and a much higher initial JumpReLU threshold (0.03). Second, we allow the gradient to flow through the JumpReLU nonlinearity to all model parameters, rather than only the threshold parameter. We found these decisions improved the MSE/L0 frontier in our setup.
- We schedule our sparsity penalty to linearly ramp up from 0 to its final value throughout the entire duration of training. In previous work, the ramp-up phase took place over a small initial fraction of training steps.
- We introduce a new "pre-activation loss" of $\sum_f \mathrm{ReLU}(-h_f)$ where h_f is the pre-nonlinearity activation of feature f. We found this loss helps prevent dead features on large CLT runs. We only used this loss for our experiments on Haiku 3.5, with a coefficient of $3\times10^{-6}$.

We chose our number of training steps to scale slightly sublinearly with the number of features, and our learning rate to scale approximately as one over the square root of the number of FLOPs. These are rough best-guess estimates based on prior experiments, and not precise scaling laws. In our largest 18L run, we used ~3B training tokens. In our largest Haiku run, we used ~16B training tokens.

In 18L, we used a constant sparsity penalty across CLT sizes. In Haiku, we increased the penalty with the CLT size (we found that otherwise, the L0 of the runs increased with size). In general we targeted L0 values in the low hundreds, based on preliminary investigation of what produced interpretable results in our graph interface.

Our 18L model is a pretraining-only model, and thus we used only pretraining data when training CLTs for it. For Haiku, by contrast, we trained the CLT using a mix of pretraining and finetuning data.

Encoder parameters are initialized by sampling from $U\left(\frac{-1}{\sqrt{\mathrm{n\_features}}}, \frac{1}{\sqrt{\mathrm{n\_features}}}\right)$ and decoder parameters are initialized by sampling from $U\left(\frac{-1}{\sqrt{\mathrm{n\_layers}\cdot\mathrm{d\_model}}}, \frac{1}{\sqrt{\mathrm{n\_layers}\cdot\mathrm{d\_model}}}\right)$.

We shuffle our training data at the token level. We've found that not shuffling leads to significantly worse performance. We suspect a variety of partial shuffles, such as the one used in [137], could perform just as well as a full shuffle.

### Engineering & Infrastructure Considerations

Here we share some engineering considerations relevant to training CLTs at scale – in particular, how it differs from training standard transcoders (“per-layer transcoders”, PLTs).

In our training implementation, the key difference between PLTs and CLTs is that, *per accelerator (GPU, TPU, etc)*, they use the same number of FLOPS, but the CLT uses n_layers times more network bandwidth. Thus when training CLTs more time should be spent profiling and optimizing network operations. This isn’t obvious, so we work through the details below.

In our training setup, features are sharded across accelerators. For each batch:

1. Each accelerator receives input and target activations.
2. Each accelerator computes partial predictions based on the features it stores.
3. An all-reduce operation is performed across all accelerators to combine these partial predictions.
4. Each accelerator then runs a backward pass and performs a gradient step using the combined predictions. A constant fraction of each accelerator's high bandwidth memory (HBM) is dedicated to parameters, with the remainder used for intermediate computations.

PLTs and CLTs have the same number of parameters per accelerator because the same fraction of HBM is dedicated to parameters. The number of FLOPS scales with batch size multiplied by the number of parameters so PLTs and CLTs use the same FLOPS *per accelerator*. If a PLT and a CLT have the same number of features then the CLT would have more total parameters, use more accelerators, and use more total FLOPS, but FLOPS per accelerator would be the same.

Some optimizations to consider:

- Have each accelerator load a different training batch. When it’s time to train on a batch, the accelerator that loaded it broadcasts it to all other accelerators using fast accelerator-to-accelerator comms.
- Broadcast the batch for the next training step during the backwards pass of the previous training step.
- Fetch future training batches in the background while training.
- Use lower precision (e.g. bfloat16) to reduce the amount of data sent.

We store activations to a distributed file system, then load them during training. Both CLTs and PLTs on every layer require activations from every layer of the underlying model. Thus we’ve optimized our code that collects activations for that use case. Collecting activations from all layers requires the same FLOPs as collecting from a single layer, but n_layers times more network bandwidth.

A possible optimization we didn’t implement is sparse kernels from [10]. The decoder forward, decoder backward, and encoder backward passes only operate on active features, so with the proper sparse kernels, FLOPS would be significantly reduced. CLTs have the same number of encoder parameters and FLOPS as PLTs, but n_layers/2 times more decoder parameters and FLOPS. Thus this optimization is more important for CLTs. This optimization makes it more likely that network operations will be the training bottleneck. Note that JumpReLU’s straight-through estimator has nonzero gradients on inactive features. We think it’s plausible that that isn’t important or we could switch to a different nonlinearity, but we haven’t tried it.

Another possible optimization we didn’t implement is to change how the decoder is sharded to remove the all-reduce. We could shard the encoder over features and shard the decoder over the output dimension. The all-reduce is removed because each decoder shard computes the MSE over a slice of the output dimension. Each decoder shard needs access to all active features, so an all-to-all is needed to share all active features with all shards. Another all-to-all is required on the backward pass. Given feature sparsity, these network operations are much smaller than the all-reduce.

**Efficiency relative to alternatives**

Compared to alternatives like per-layer transcoders (PLTs), CLTs use more parameters – the same number of encoder parameters, but approximately n_layers/2 times as many decoder parameters.  Thus, even if CLTs provide value beyond PLTs with the same number of *features*, it is reasonable to ask whether they perform better at a fixed training *cost*. Empirically, we find that CLTs perform better (according to some metrics) or comparably (according to others) than PLTs at a fixed cost, even without the use of sparse kernels (which, as noted above, advantage CLTs more than PLTs). Broadly, this is because the number of *total features* (across all layers) required to achieve a given level of performance is much less for CLTs than SLTs, which compensates for their additional per-feature cost. In more detail:

- Note that in our experiments on the 18L model, we ran CLTs and sets-of-PLTs with 300K, 1M, 3M, and 10M total features. Note that we also scaled training steps along with n_features. Coincidentally, this meant that the number of FLOPs used by our PLT runs were comparable to the number of FLOPS used by the “next size smaller” CLT run (i.e. 10M PLT ≈ 3M CLT, etc.).
- Comparing approximately FLOPs-matched PLT and CLT runs on our various evaluations, we notice that:
    - In terms of (MSE, L0) and (replacement model KL divergence, L0), the 1M feature CLT strictly outperforms the 10M feature set-of-PLTs, using 5–10× less compute.
    - What about feature interpretability scores?
        - If we compare the 3M feature CLT to the 10M set-of-PLTs (roughly FLOPs equivalent), the CLT attains somewhat lower MSE and KL divergence and roughly equal Sort Eval scores.
        - The picture for the Contrastive Eval is less clear.  The 3M feature CLT attains lower MSE and KL divergence than the 10M feature PLTs, but higher worse Contrastive Eval scores. Thus we cannot declare a Pareto-winner here (we would need to run a sweep over the sparsity penalty to make the judgment).
- We have found that cross-layer dictionaries require a comparable order of magnitude of training tokens to single-layer dictionaries (a rough estimate based on our 18L experiments is that they benefit from ~2 times more tokens).

Thus, it seems that overall, CLTs are at least as cost-effective as PLTs (and presumably per-layer SAEs, which have the same parameter-count as PLTs) to reach a given level of circuit understanding, and likely moreso. However, if another group finds PLTs to be easier to implement or more cost-effective, we expect it’s possible to find interesting results using PLTs as well. We expect that even making attribution graphs with raw neurons will yield plenty of interesting insights.

We also note that optimizations could make CLTs even more cost-effective – as discussed above, implementing sparse kernels and communications could in principle reduce the number of FLOPs performed by CLTs by a factor of up to n_layers/2.

We suspect there are potential algorithmic optimizations to be made as well. For instance, it is possible that most of the benefits of CLTs can be captured by learning layer-to-layer linear transformations that are common to *all* features, eliminating the need for each feature to have its own independent decoder parameters for every downstream model layer.

## Attribution Graph Computation

We give complete definitions of nodes and edges in the attribution graph, continuing the discussion in the main text. Associated to each node type are two (sets of) vectors: input vectors, which affect the edges for which the node is a target, and output vectors, which affect the edges for which the node is a source.

- The output node corresponding to a vocabulary token $\mathrm{tok}$ has candidate output tokens has input vector $v_{in} = \nabla(\mathbf{logit}_{\mathrm{tok}} - \overline{\mathbf{logit}})$, the gradient of the pre-softmax logit for the target token minus the mean logit.
- A feature node corresponding to feature $s$ at context position $c_s$ has a set of output vectors $v^{\ell}_{out} = W^{\ell_s \to \ell}_{\text{dec, }s}$, for $\ell_s \leq \ell$. It has input vector $v^{\ell}_{in} = W^{\ell_s}_{\text{enc, }s}$.
- An embedding node corresponding to input token $\mathrm{tok}$ has output vector $v_{out} = \mathrm{Emb}_{\mathrm{tok}}$.
- An error node at context position $c$ and layer $\ell$ has output vector $v_{out} = \mathrm{MLP}_{\ell}(x_{c,\ell}) - \mathrm{CLT}_{\ell}(x_c)$ where $x$ represents the full residual stream vector at all context positions and layers from a forward pass of the underlying model, $\mathrm{MLP}_{c,\ell}$ is the output of the layer $\ell$ MLP, and $\mathrm{CLT}_{\ell}$ is the output of the CLT to layer $\ell$.

For edges from embedding or error nodes to feature or output nodes, the edge weight is:

$$A_{s\to t} = v^T_{out,s} J^{\blacktriangledown}_{c_s,\ell_s\to c_t,\ell_t} v_{in,t}.$$

For edges from feature nodes to feature or output nodes, the edge weight is

$$A_{s\to t} = a_s \sum_{\ell_s \leq \ell < \ell_t} (v^{\ell}_{out})^T J^{\blacktriangledown}_{c_s,\ell\to c_t,\ell_t} v_{in,t}.$$

The Jacobian $J^{\blacktriangledown}_{c_s,\ell_s\to c_t,\ell_t}$ can be expanded into a sum over all paths of length $\ell_t - \ell_s$ in the underlying model through attention heads and residual connections. Consider a path $p$ starting at position $c_s$ and layer $\ell_s$ and ending at position $c_t$ and layer $\ell_t$ consisting of $i = 1, 2, \ldots, (\ell_t - \ell_s)$ steps, each of which is either a residual stream step going from layer $\ell + i - 1$ to $\ell + i$ or an attention head step going from position $c_i$ to $c_{i+1}$ via attention head $h_i$ with attention weight $a^{h_i}_{c_i \to c_{i+1}}$ and applying transformation $OV_{h_i}$. Each step is associated to a linear transformation $\pi_i$ which is the identity for a residual stream step and $a^{h_i}_{c_i \to c_{i+1}} OV_{h_i}$ for an attention step. The whole path $p$ is thus associated to a linear transformation $\pi_p = \prod_i \pi_i$. Then we can write

$$J^{\blacktriangledown}_{c_s,\ell_s\to c_t,\ell_t} = \sum_{p \in \mathcal{P}(c_s,c_t,\ell_s,\ell_t)} \pi_p$$

where $\mathcal{P}(c_s, c_t, \ell_s, \ell_t)$ is the set of all paths starting at position $c_s$ and layer $\ell_s$ and ending at position $c_t$ and layer $\ell_t$.

To compute the graph edges in practice, we iterate over target nodes in the graph (output or feature nodes). For each node we:

- Inject the input vector for the node into the residual stream. For CLT features, we inject the encoder of the target CLT feature in the residual stream at the layer and token position where it reads. For logit nodes, we instead inject the gradient of the logit (minus the mean logit) at the final layer residual stream.
- Do a backwards pass in the underlying model with
    - stop-gradient operators on the MLP outputs, and
    - normalization denominators and attention patterns set to values recorded from the underlying model's forward pass on the prompt.
- For a CLT source node (feature, token position), we take the sum of the dot products of its decoder vector in each layer with the gradient in that layer, times the activation of that feature.
- For an embedding or error source node, we simply take the product of its vector with the gradient.

The cost to compute the graph is linear in the number of active features in the prompt, and is dominated by the cost of the underlying model backwards pass. To economize, we sometimes compute the graph adaptively: starting from the output nodes for the logits, then maintaining a queue of the feature nodes with the greatest influence on the logit, and computing the input edges for nodes based on their order in the queue. This allows us to compute the most important parts of the graph first.

## Graph Pruning

To increase the signal to noise ratio of our manual interpretation, we rely heavily on a graph pruning step to reduce the number of nodes and edges in the graph. We employ a two-step algorithm which first prunes the nodes and then prunes the edges of the remaining nodes. The details are as follows:

- We take the adjacency matrix of the graph and replace all the edge weights with their absolute values to obtain an unsigned adjacency matrix. Then we normalize the input edges to each node so that they sum to 1. Let A refer to this normalized, unsigned adjacency matrix.
- We compute the *indirect influence matrix* $B = A + A^2 + A^3 + \cdots$ which can be efficiently computed as $B = (I - A)^{-1} - I$. The entries of $B$ indicate the sum of the strengths of all paths between a given pair of nodes, where the strength of any given path is given by the product of the values of its constituent edges in A.
- We take the rows of $B$ corresponding to logit nodes, and take their weighted average, weighting according to the model's output probabilities for each token. This results in a vector of *logit influence scores* for each node in the graph.
- We sort the logit influence scores of non-logit nodes in descending order and choose the minimum cutoff index such that the sum of the influence scores prior to the cutoff divided by the total sum exceeds a given threshold. We use a threshold of 0.8 unless otherwise noted. We prune away all non-logit nodes after the cutoff.
- With the pruned graph, we re-compute a normalized unsigned adjacency matrix and logit influence scores for each node using the same method as above. Then we assign a logit influence score to each *edge* by multiplying the logit influence score of the edge's output node by the normalized edge weight. We prune edges according to the same strategy as for nodes, but using a higher cutoff 0.98 unless otherwise noted.
- Logit nodes are pruned separately. We keep the logit nodes corresponding to the top $K$ most likely token outputs, where $K$ is chosen so that the total probability of the logit nodes is greater than 0.95. If this would result in $K > 10$, we clamp $K$ to $10$.
- Embedding nodes and error nodes are not pruned.

We include pseudocode below. Our pruning thresholds can, very roughly speaking, be interpreted as determining the percent of "total logit influence" we lose from pruning. That is, we choose the subset of nodes that are responsible for ~80% of the influence on the logits, and the subset of their input edges responsible for ~98% of the remaining influence. Our choice of thresholds is arbitrary and was chosen manually to balance preservation of important paths with the need to prune graphs to a manageable, interpretable size.

For longer prompts, we can also employ an adaptive algorithm, to greedily construct a graph out of the most influential nodes, rather than generating the full graph only to prune most of it away. To do so, we maintain a set of *explored* nodes (i.e., nodes from which we have computed backward attributions) and an estimate of the most influential nodes, where *unexplored* nodes count as errors. At each step, we compute the top $k$ most influential nodes using our influence scores, and attribute back from these. We terminate when a target number of nodes have been explored.

With our default parameters, pruned graphs are substantially smaller than the original raw graphs; the number of nodes typically decreases by ~10× and the number of edges typically decreases by ~500× (the exact numbers are sensitive to prompt length).

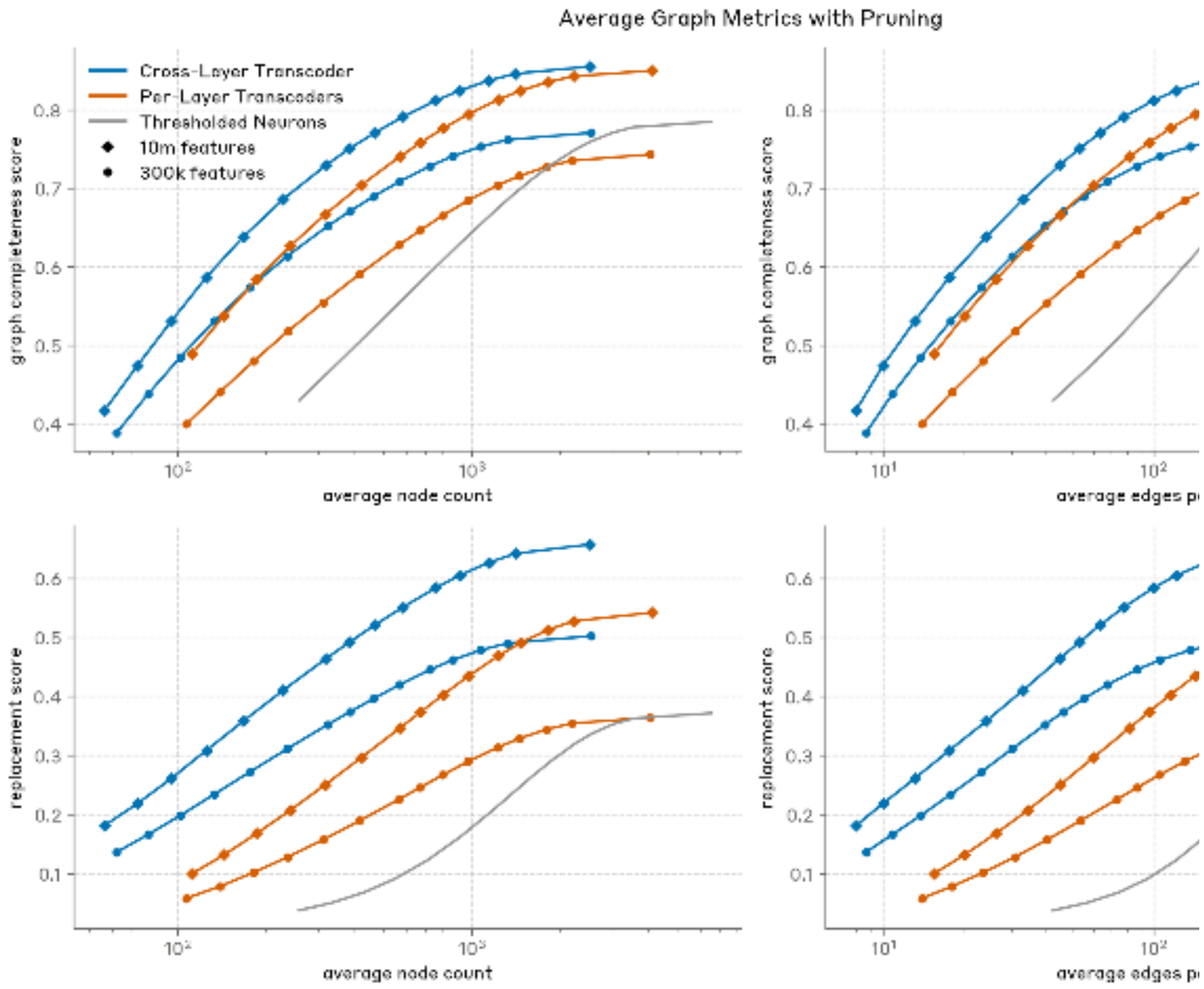


**Figure 32**: Comparison of graph completeness and replacement score versus per-token L0 for graphs generated with different underlying dictionaries. Higher scores corre
lower L0 suggests more interpretable graphs.

Pseudocode for pruning algorithm:

```
function compute_normalized_adjacency_matrix(graph):
    # Convert graph to adjacency matrix A
    # A[j, i] = weight from i to j (note the transposition)
    A = convert_graph_to_adjacency_matrix(graph)
    A = absolute_value(A)

    # Normalize each row to sum to 1
    row_sums = sum(A, axis=1)
    row_sums = maximum(row_sums, 1e-8)  # Avoid division by zero
    A = diagonal_matrix(1/row_sums) @ A

    return A

function prune_nodes_by_indirect_influence(graph, threshold):
    A = compute_normalized_adjacency_matrix(graph)

    # Calculate the indirect influence matrix: B = (I - A)^-1 - I
    # This is a more efficient way to compute A + A^2 + A^3 …
    B = inverse(identity_matrix(size=A.shape[0]) - A) - identity_matrix(size=A.shape[0])

    # Get weights for logit nodes.
    # This is 0 if a node is a non-logit node and equal to the probability for logit nodes
    logit_weights = get_logit_weights(graph)

    # Calculate influence on logit nodes for each node
    influence_on_logits = matrix_multiply(B, logit_weights)

    # Sort nodes by influence
    sorted_node_indices = argsort(influence_on_logits, descending=True)

    # Calculate cumulative influence
    cumulative_influence = cumulative_sum(
        influence_on_logits[sorted_node_indices]) / sum(influence_on_logits)

    # Keep nodes with cumulative influence up to threshold
    nodes_to_keep = cumulative_influence <= threshold
    # Create new graph with only kept nodes and their edges
    return create_subgraph(graph, nodes_to_keep)

# Edge pruning by thresholded influence
function prune_edges_by_thresholded_influence(graph, threshold):
    # Get normalized adjacency matrix
    A = compute_normalized_adjacency_matrix(graph)

    # Calculate influence matrix (as before)
    B = estimate_indirect_influence(A)

    # Get logit node weights (as before)
    logit_weights = get_logit_weights(graph)

    # Calculate node scores (influence on logits)
    node_score = matrix_multiply(B, logit_weights)

    # Edge score is weighted by the logit influence of the target node
    edge_score = A * node_score[:, None]

    # Calculate edges to keep based on thresholded cumulative score
    sorted_edges = sort(edge_score.flatten(), descending=True)
    cumulative_score = cumulative_sum(sorted_edges) / sum(sorted_edges)
    threshold_index = index_where(cumulative_score >= threshold)
    edge_mask = edge_score >= sorted_edges[threshold_index]

    # Create new graph with pruned adjacency matrix
    pruned_adjacency = A * edge_mask
    return create_subgraph_from_adjacency(graph, pruned_adjacency)
```

## Validating the Replacement Model

### Validating Node-to-Logit Influence

We rely on indirect influence statistics for pruning, adaptive generation, and our graph statistics. In this section, we validate that indirect influence is a better proxy for “importance” than other basic baselines like activation or logit attribution (with stop grads).

Specifically, on a sample of 20 prompts, we compute the effect of ablating every feature and measure the KL divergence between the model with the ablation and the clean forward pass. We perform this intervention using “constrained” patching, so we also sweep over different ranges of layers to apply the intervention. In the plot below, we report the average log-log Pearson correlation between ablation KL and initial activation, direct logit edge weight, and indirect influence on the logit. We see that node influence is most predictive of ablation effect, followed by direct edge weights. Both outperform a simple activation baseline.

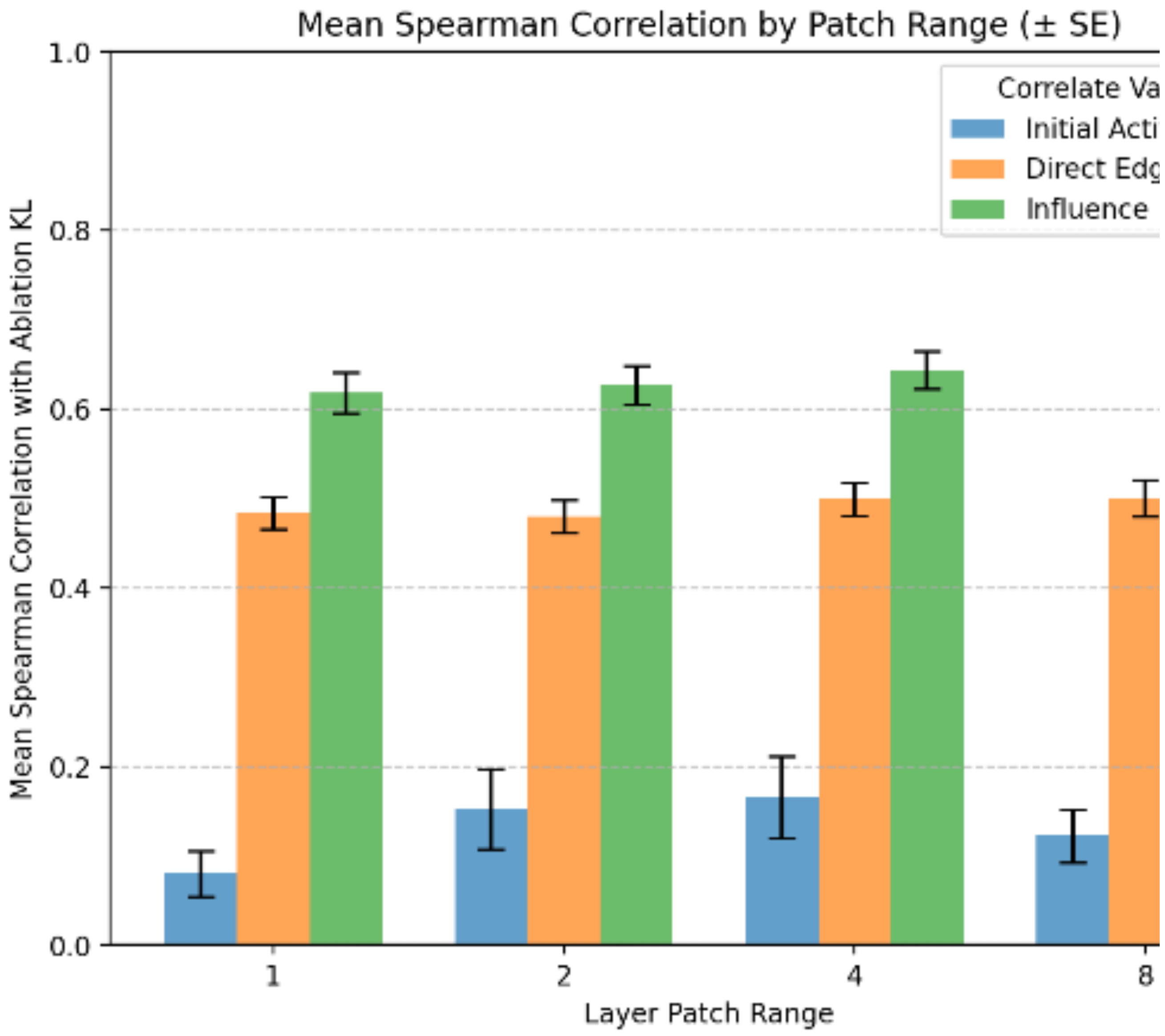


**Figure 33**: Spearman correlation between a node's ablation KL divergence (the KL of a clean forward pass and a forward pa feature ablated) and several natural node importance baselines: graph-based influence, the direct edge weight to the logit, original activation. We sweep over ranges to apply our "constrained" patching, and find the range makes little difference.

#### Validating Feature-to-feature Influence

We can use edges in the attribution graphs to estimate the influence features should have on each other. However, if the replacement model is unfaithful or incomplete, a perturbation experiment might not yield results which are consistent with an estimate based on graph influence. For example, initially inactive components could engage in self-correction and dampen the effect of the perturbation. To estimate the extent to which this happens in our graphs, we measure how often perturbations have the expected effect by ablating every feature in the graph, and recording the activation of features downstream of it. Our ablations are done using constrained patching in the range $[i, i+2]$ where $i$ is the encoder layer of the feature being perturbed (note this choice of layer is arbitrary, see further discussion here). When averaged over a dataset of 20 prompts, we find a Spearman correlation of 0.72 between the influence of a source feature on a target feature as described by the graph, and the effect of ablating said feature on the target feature's activation (where we normalize by the original activation and take the absolute value since influence is unsigned). We include scatter plots from three examples from the paper:

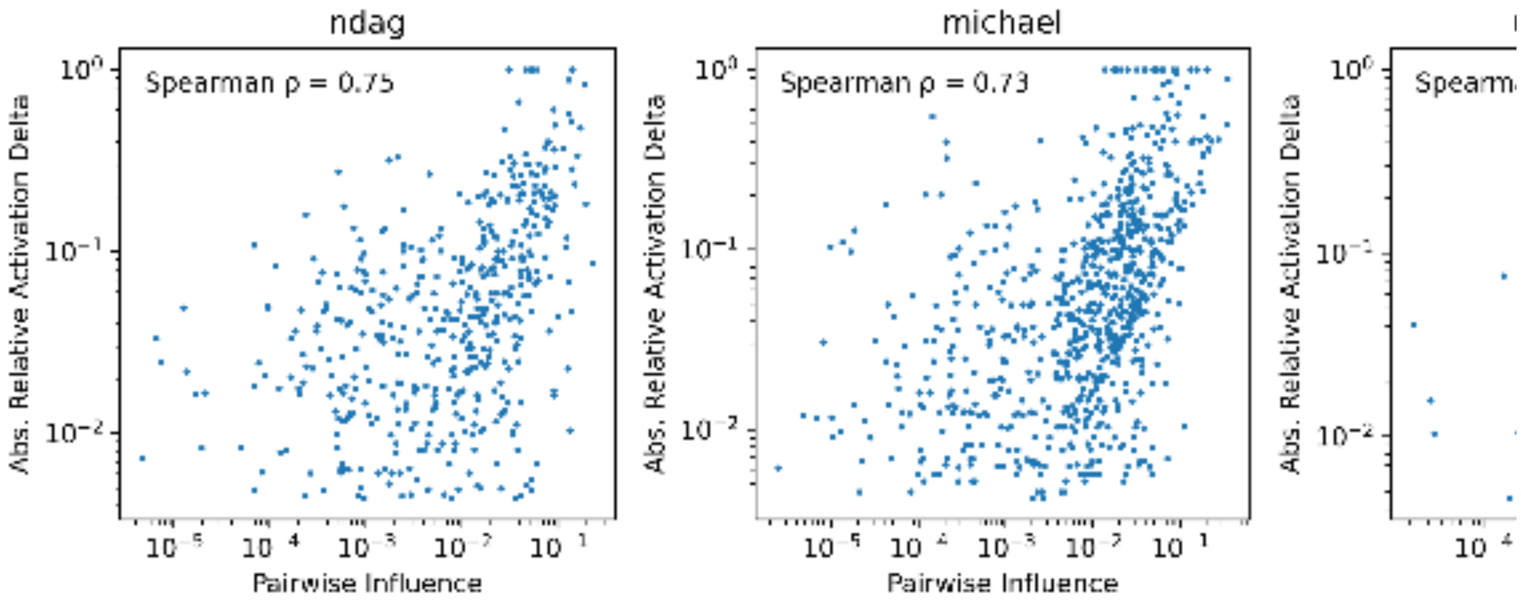


**Figure 34**: Relationship between (x) pairwise influence of all nodes in the pruned graph (y) the absolute relative effect on a target node's activation after ablating the source n
observe the graph-based pairwise influence is fairly predictive of the ablation effect.

These figures show that there are relatively few points that (1) have large influence and no ablation effect (lower right) and (2) have low influence but high ablation effect (upper left). This suggests that indirect influence through the graph is a fairly good proxy for real effects in the model.

#### Evaluating Faithfulness of the Local Replacement Model

In this work, we use (local) replacement models as a window into the mechanisms of the original model. This approach is problematic if the (local) replacement model uses very different mechanisms than the original model. In the main text, we performed perturbations of features to confirm specific mechanistic theories and measured how well the replacement model's outputs matched the underlying model's outputs. Here, we focus on how well their inner states match in response to a broader set of perturbations, including off-distribution perturbations, as an (imperfect) gauge of the replacement model's quality.

We use the local replacement model (i.e. with reconstruction errors added back in as constant factors, and attention patterns frozen[43] ), ensuring agreement of the replacement and original models under zero-size perturbations. Like "iterative patching," the replacement model may respond to perturbations by activating features which were not previously active at baseline.

We perturb the models at single token positions using 3 types of perturbations: adding a feature's encoder-vector to the residual stream, adding random directions to the residual stream, and perturbing an upstream feature (to be defined precisely below).

Broadly, averaging across choices of intervention layers, we find that:

- Perturbation results are reasonably similar between the replacement model and the underlying model when measured one layer after the intervention.
- Perturbation discrepancies compound significantly over layers.
- Compounding errors have a gradually detrimental effect on the faithfulness of the *direction* of perturbation effects, which are largely consistent across CLT sizes, with signs of faithfulness worsening slightly as dictionary size increases.
- Compounding errors can have a catastrophically detrimental effect on the *magnitude* of perturbations. This effect is worse for larger dictionaries.

**To measure faithfulness for perturbations in encoder directions**:

- We first choose a residual-stream layer and select a random active feature with an encoder-vector in that layer.
- We take the encoder-vector of the chosen feature, and add a multiple of it (see next bullet) to the residual stream of both the replacement model and frozen underlying model.
- This perturbation to the residual is scaled to increase the chosen feature's activation to 0.1 above its original value.
- After modifying the residual stream, we perform the forward pass of both perturbed models.

**To measure faithfulness for perturbations in random directions**:

- We follow the above process to determine the perturbation's magnitude, but prior to its addition to both residual streams, the perturbation-vector is rotated to a random direction on the unit sphere.

**To measure faithfulness for perturbations to upstream features**:

- We select an active feature to perturb and an *intervention layer* after the feature's encoder layer.
- We increase the feature's activation by 0.1 and run the local replacement model with this perturbation applied, up until the intervention layer.
- At the intervention layer, we compare two forward passes:
    - In one, the residual-stream state of this forward pass is copied-over to the underlying model at the intervention layer, and the underlying model is run forward from there.
    - In the other, we continue the forward pass of the local replacement model.

At each layer downstream of the perturbation, we can determine a net perturbation for both perturbed models by taking the activations from the (perturbed) forward pass and subtracting clean (un-perturbed) baseline activations. The net perturbations for each model are then compared to yield measures of faithfulness, by computing cosine-similarity and mean-squared-error between them.

We believe that perturbing upstream features produces the most "on-distribution" perturbations of these approaches, since random directions may fall into relatively inactive dimensions and encoder-directions are the most sensitive dimensions of the replacement model. Below are results for upstream-feature perturbations to features in layer 5 of the model, averaged over choice of the intervention layer.

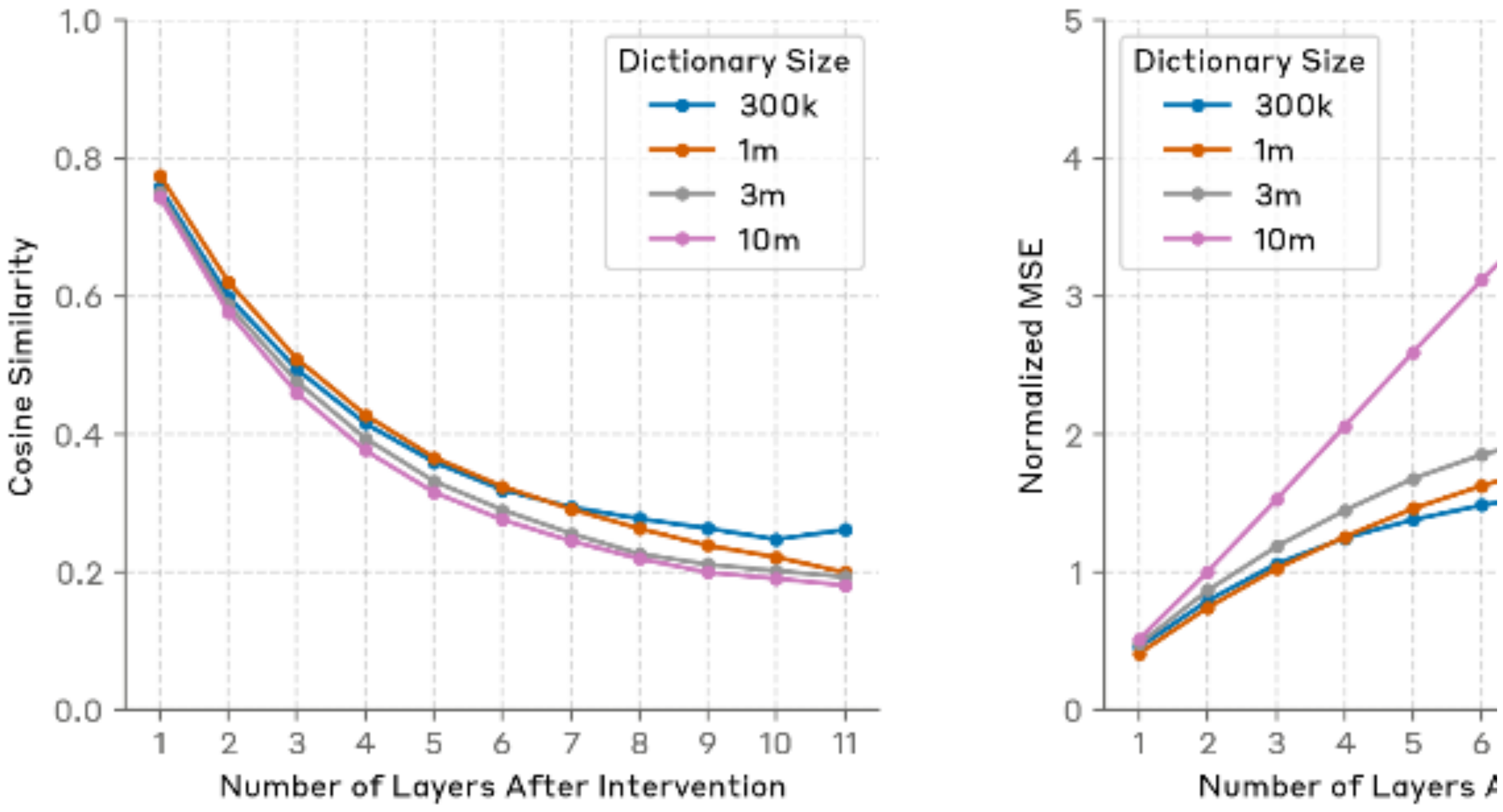


**Figure 35**: Faithfulness of perturbations from upstream features in layer 5.

Below are cosine-similarity faithfulness metrics for the 10m dictionary for other choices of upstream layers and other perturbation strategies.

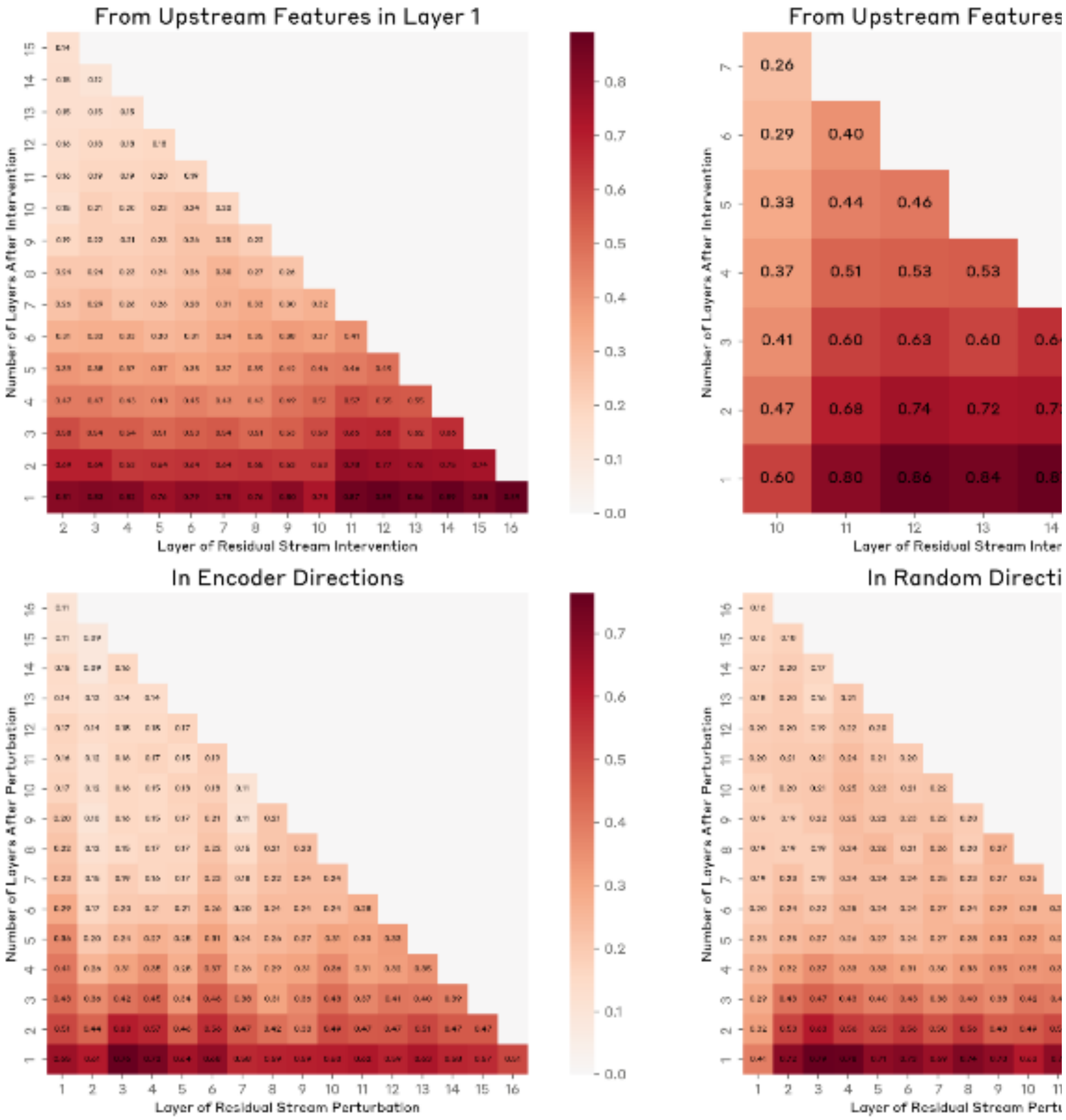


**Figure 36**: Cosine similarity of perturbations.

As can be seen from the bottom rows of the graphic above, cosine-faithfulness of the 10m dictionary for 18L in the layer following the intervention or perturbation layer is around 60–80%. Computing the average of this bottom-row for different dictionary sizes, in the figure below, the largest dictionaries show signs of mildly diminished faithfulness for perturbations from upstream features.

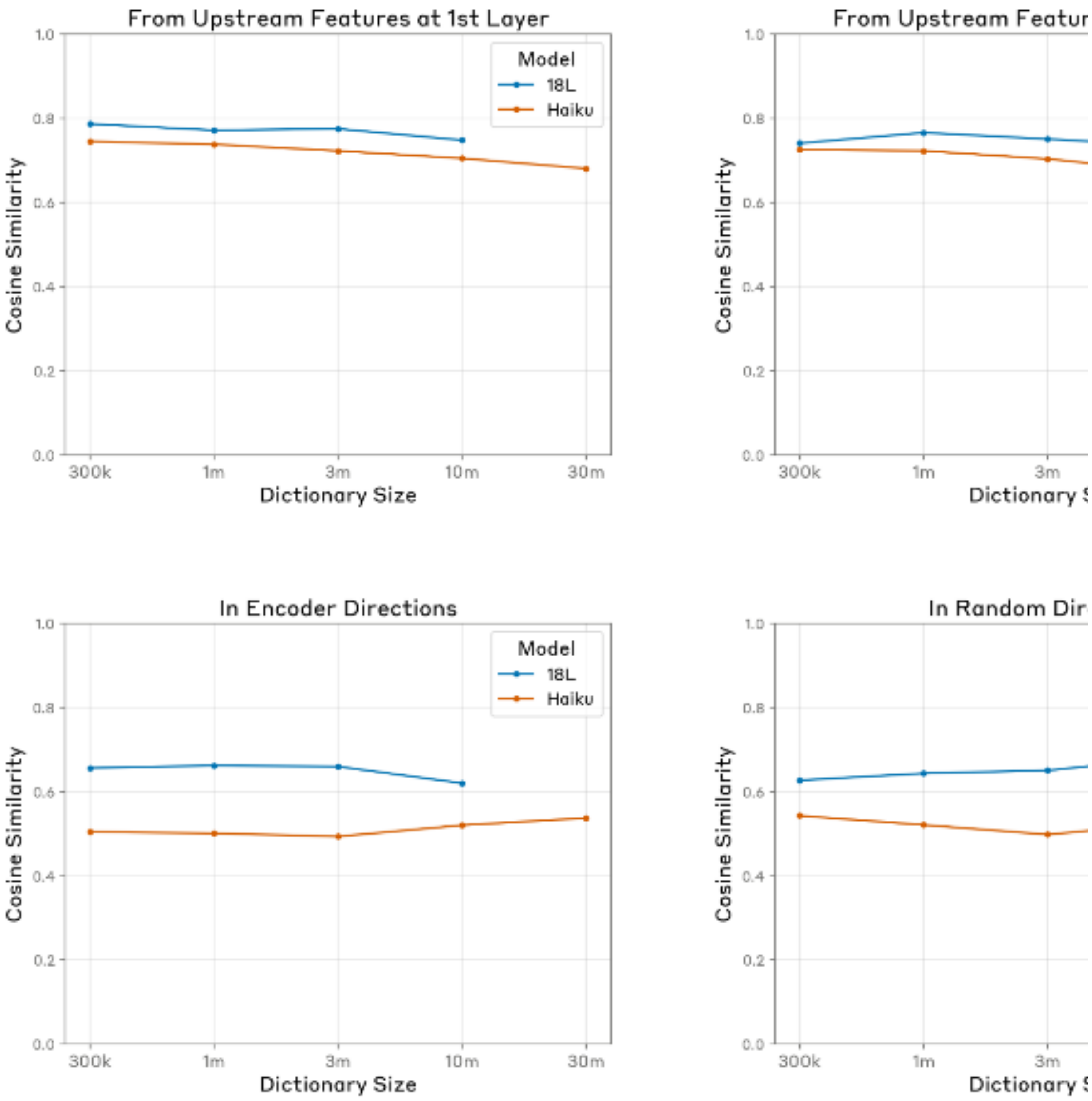


**Figure 37**: Average cosine similarity 1 layer after an intervention/perturbation.

The decay of faithfulness-metrics over multiple layers, at first glance, raises questions about the reliability of our interpretability graphs, analyses, and steering techniques. Indeed, these results seem at odds with the qualitatively successful results of many of the perturbation experiments we use to validate attribution graphs (both in this paper and its companion). We suspect this is due to a combination of factors:

- The compounding unfaithfulness problem affects perturbation *magnitudes* more severely than *directions.* We compensate for this issue in practice by trying a range of perturbation strengths in ad-hoc, empirical fashion.
- Aggregating features into supernodes may help "denoise" faithfulness errors.
- The perturbation experiments we perform in the context of graph validation are highly nonrandom – we are typically perturbing and measuring "important" or "cruxy" features, and doing so in the layers where their output is strong. A proportion of the effect we observe can also be attributed to "guaranteed effects", as we describe in § Nuances of Steering with Cross-Layer Features.

How do these results compare for per-layer transcoder (PLT) dictionaries? Cosine-faithfulness metrics for PLTs were broadly similar to those for CLTs. The PLT replacement model achieves lower normalized MSE at one layer after the intervention, but compounding errors in the largest PLT replacement models accumulate slightly more rapidly after several layers, as seen in the figure below. This may reflect an advantage of the CLT skip connections, which allow for shorter paths through the replacement model.

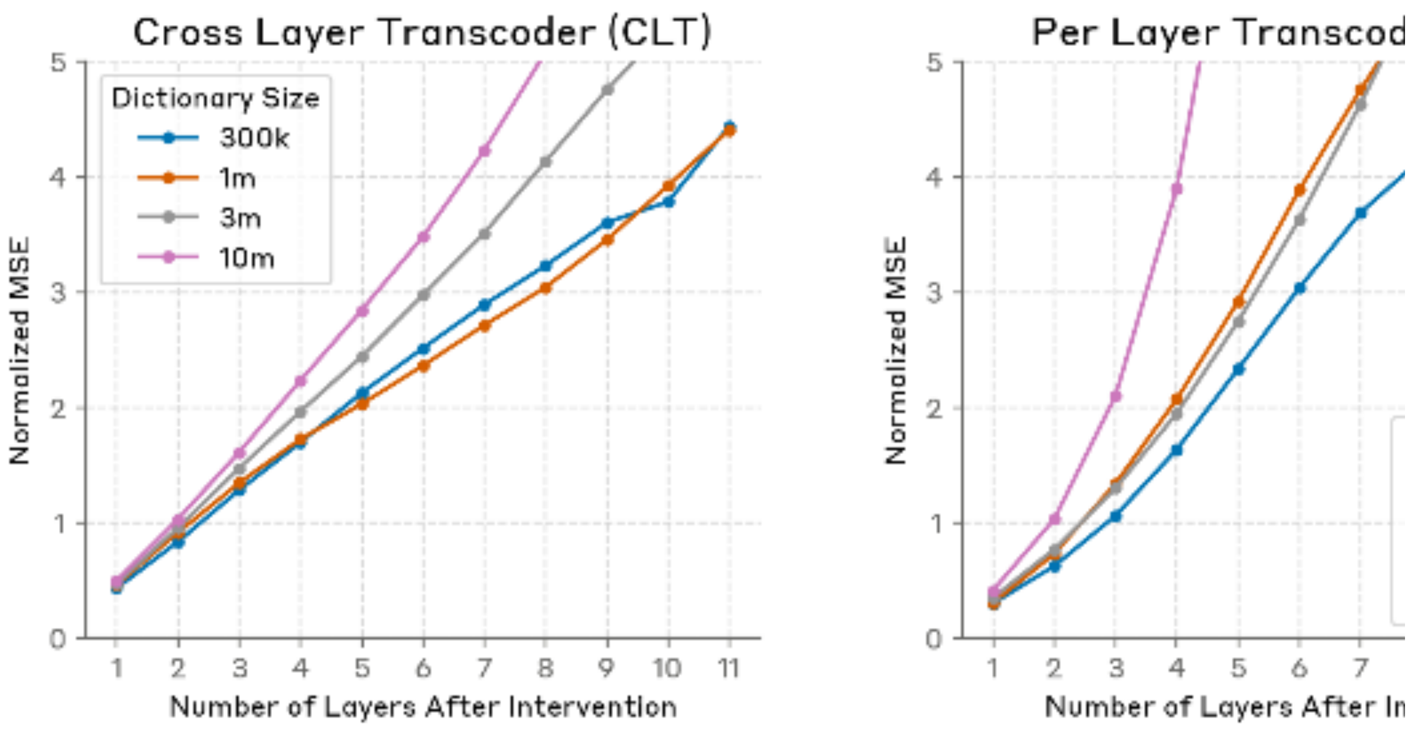


**Figure 38**: Normalized MSE of perturbations from upstream features in layer 5.

Normalized MSE results are sensitive to the scale of the perturbation. To make scales comparable across replacement model types for the figure above, the perturbation to the initial upstream feature in Layer 5 is scaled differently for each sample so that Layer 6 is perturbed by a constant proportion of its magnitude.

## Nuances of Steering with Cross-Layer Features

Our perturbation methodology (constrained patching and iterative patching) is somewhat non-obvious at first glance. Why don't we simply add to the residual stream along a feature's decoder vector at each layer as we run a forward pass of the model? The reason for this is that it risks double-counting the effect of the feature. For instance, consider a feature in layer 1, with decoders writing to layers 2, 3, and onward. It may be that the activations being reconstructed in layer 2 were, in the original model, causally upstream of the activations being reconstructed in layer 3. Thus, injecting a perturbation into layers 2 and 3 would "double up" on the feature's effect. Across more layers, this effect could compound to be quite significant.

Our constrained patching approach avoids this problem by computing perturbed MLP outputs once at the outset, and then clamping MLP outputs to these precomputed values (rather than adding to the MLP outputs). While this avoids the double-counting problem, it requires us to choose an intervention layer – we apply the perturbations in all layers up to the intervention layer, but not after (otherwise we would be clamping all of the MLP outputs of the model, and thus we wouldn't be testing any hypotheses about how the model responds to perturbations). This makes it challenging to measure "the entire effect" of a feature (see § Unexplained Variance & Choice of Steering Factors).

It also makes the interpretation of perturbation experiment effects somewhat awkward. Suppose a source feature is in layer 1, and our perturbation intervention layer is 5, i.e. we apply the perturbation in layers 1 through 5. This perturbation will have an impact on, say, features in layer 3. However, the "knock-on" effects of this impact will be at least partially (and perhaps completely) overwritten, since the MLP outputs at layers 4 and 5 are clamped to values computed at the outset. Note that our alternative approach to patching, "iterative patching," does capture such knock-on effects. However, because of this, it is not as suitable as a validation of mechanisms in an attribution graph, since the perturbation is less precise.

Another subtle aspect of our perturbation experiments is that *direct* feature-feature interactions described by an attribution graph are nearly forced to be confirmed in steering experiments (when we freeze attention patterns), since they measure linear effects of source feature decoders on target feature encoders (via fixed linear weights, conditioned on frozen attention patterns) – and guaranteed when choosing the intervention layer to be the same as the target feature's encoder layer. Thus, the nontrivial thing we are testing with perturbation experiments is the validity of the attribution graph's claims about "knock-on effects." That is, suppose our attribution graph tells us that feature A excites feature B, which excites feature C, which upweights token X. When we perform a perturbation experiment by inhibiting feature A, we will get an inhibitory effect on feature B "for free," and we are interested in validating whether the subsequent effects on features C and the output token are as expected.

## Unexplained Variance and Choice of Steering Factors

Our choice of steering factor scales for intervention experiments is somewhat ad-hoc and empirically driven. For instance, in inhibition experiments, to meaningfully change the output predictions we often must clamp features to *negative* multiples of their original value, rather than simply to 0. In patching experiments where we add in a feature that was not originally active on a prompt, we often do so using activations significantly greater than that feature's typical activations.

Why do we need to "overcompensate" in this fashion? We suspect that this is because even when our features play the mechanistic roles that we expect based on attribution graphs, our perturbation experiments capture these mechanisms incompletely:

- Unexplained variance – our feature dictionaries are not infinitely large, and our CLTs do not reconstruct activations fully. Some mechanisms may simply be missing, or the features we extract may be projections of "fuller" ground-truth features that partially reside in this unexplained variance.
- Inexhaustive feature selection – on any given prompt, there are typically groups of active features with related meanings that have similar edges in the attribution graph. Thus, perturbing any single feature is insufficient – we need to perturb the full group to see its full mechanistic effect. However, identifying "the full group" is not precisely defined, and also cumbersome (it would require inspecting every active feature on every token position for each prompt). Thus, we end up perturbing groups of features that are most likely incomplete, requiring us to choose extra strong perturbation factors to compensate.
- Incomplete capturing of cross-layer effects – due to the way we perform steering with cross-layer features, our steering results measure the effect of a feature *in a particular layer,* rather than the effect of the feature globally (see § Nuances of Steering with Cross-Layer Features). Thus, our perturbations systematically undervalue features' whole effects.

## Similar Features and Supernodes

We often find that there are many features in a given graph which seem to have similar roles. There are a few candidate theories for explaining this phenomenon (which we discuss below). Whatever the underlying reason, this suggests that individual features are often best understood as *partial* contributors to a component of the mechanism used by the model. To capture these components more completely, we group related features into "supernodes". While this process is inherently somewhat subjective, we find it is important to clarify key mechanisms.

Grouping features produces a simplified "supernode graph," where we compute the edges between supernodes according to the formula below:

$$\text{supernode\_adjacency}(t, s) = \frac{\sum_{n_t \in N_t} \text{frac\_external}(n_t) \cdot \sum_{n_s \in N_s} A'_{n_t, n_s}}{\sum_{n_t \in N_t} \text{frac\_external}(n_t)}$$

Where:

- $N_t$ and $N_s$ are the set of nodes in supernodes $t$ and $s$, respectively.
- $n_t$ and $n_s$ refer to a node in the corresponding supernode.
- $\text{frac\_external}(n_t)$ is the fraction of the absolute valued input weights to $n_t$ that originate from nodes outside of the supernode.
- $A'$ refers to an input normalized graph adjacency matrix, where the weights of edges targeting each node are divided by a constant to sum to 1. $A'_{n_t, n_s}$ is the normalized edge weight from node $n_t$ to node $n_s$.

Hypotheses for why there so many related features include:

- These features represent subtly different concepts that are hard to discern from our visualization tools. Indeed, in some cases the differences are noticeable – for instance, we may choose to group together a "Texas cities" feature with a feature representing "the state of Texas" if both play a similar role in the circuit. See our § Limitations section on feature splitting for more on this issue. A related issue that could be involved is that of *feature manifolds* [139, 140, 141, 142] – concepts which are represented in truly multidimensional fashion by the model. Our CLTs may represent these as collections of hard-to-distinguish features.
- Features in different layers may represent the same concept, but based on different inputs. For instance, a layer 5 "Michael Jordan" feature may be capable of detecting more indirect references to Michael Jordan, while a layer 1 "Michael Jordan" feature can only detect when the name is written out explicitly.
- The output response of an MLP after increasing a given input feature is generically nonlinear, even when monotonic, and may require a few CLT features to approximate. (Note that a single-layer MLP with one input dimension and one output dimension can be highly nonlinear as you vary the input, and require multiple features to approximate; composing MLP layers could exacerbate this.) One experiment that could help evaluate this hypothesis would be to perturb an upstream feature using a sweep of strengths, and measure its effect on the activity of a downstream feature; if the Dallas→Austin interaction were capable of being mediated by a single Texas feature, then that dependency would have to be threshold-linear. If it's not, we'd need a group of features to capture the mechanism.

## Iterative Patching

Instead of constrained patching, we may wish to measure the effect of a feature intervention within the patching range in addition to its effect outside of the range. To do so, we can iteratively recompute the values of features which read from layers in the patching range, letting them take on new values due to our intervention. We call this iterative patching. While iterative patching may seem advantageous since it accounts for all of the effects of a feature intervention (at least according to the replacement model), it can often lead to cascading errors, as we need to repeatedly encode and decode features. In addition, the main goal of interventions is to validate edges between nodes and supernodes in attribution graphs. Constrained patching provides a simpler way to do this by limiting indirect effects.

The illustration below shows cascading effects. We patch a feature at layer 1, and impact two features computed at layer 2, and so are affected by our patch. At the end of our patch, we inject the computed state of the residual stream back into the model.

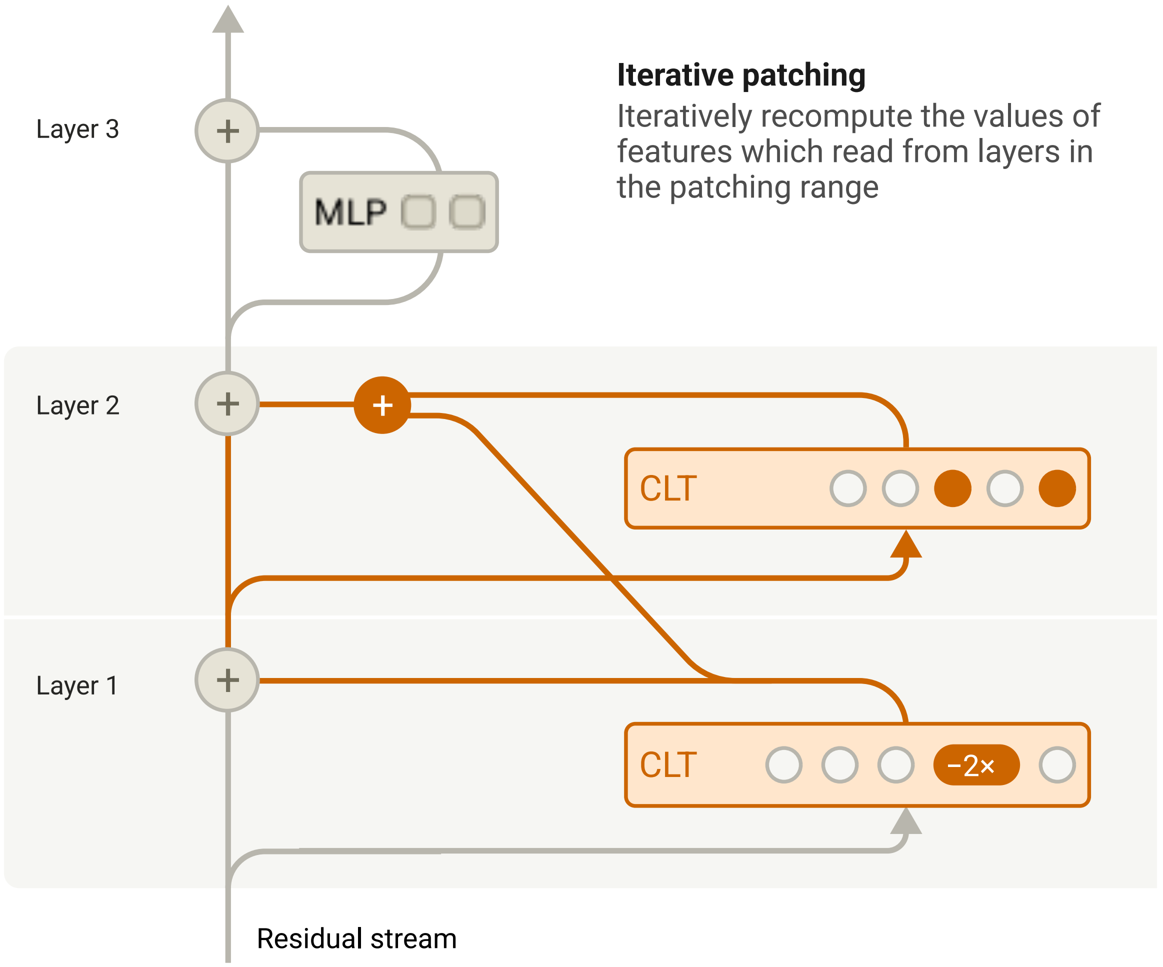

Layer 3
MLP
Iterative patching
Iteratively recompute the values of features which read from layers in the patching range
Layer 2
CLT
Layer 1
CLT
−2×
Residual stream

**Figure 39**: A schematic of iterative patching.

## Details of Interventions

In this section, we show full intervention results for some of the case studies we discussed above.

Below, we show the effect of inhibiting every supernode in the acronym case study with an inhibition factor of 1.

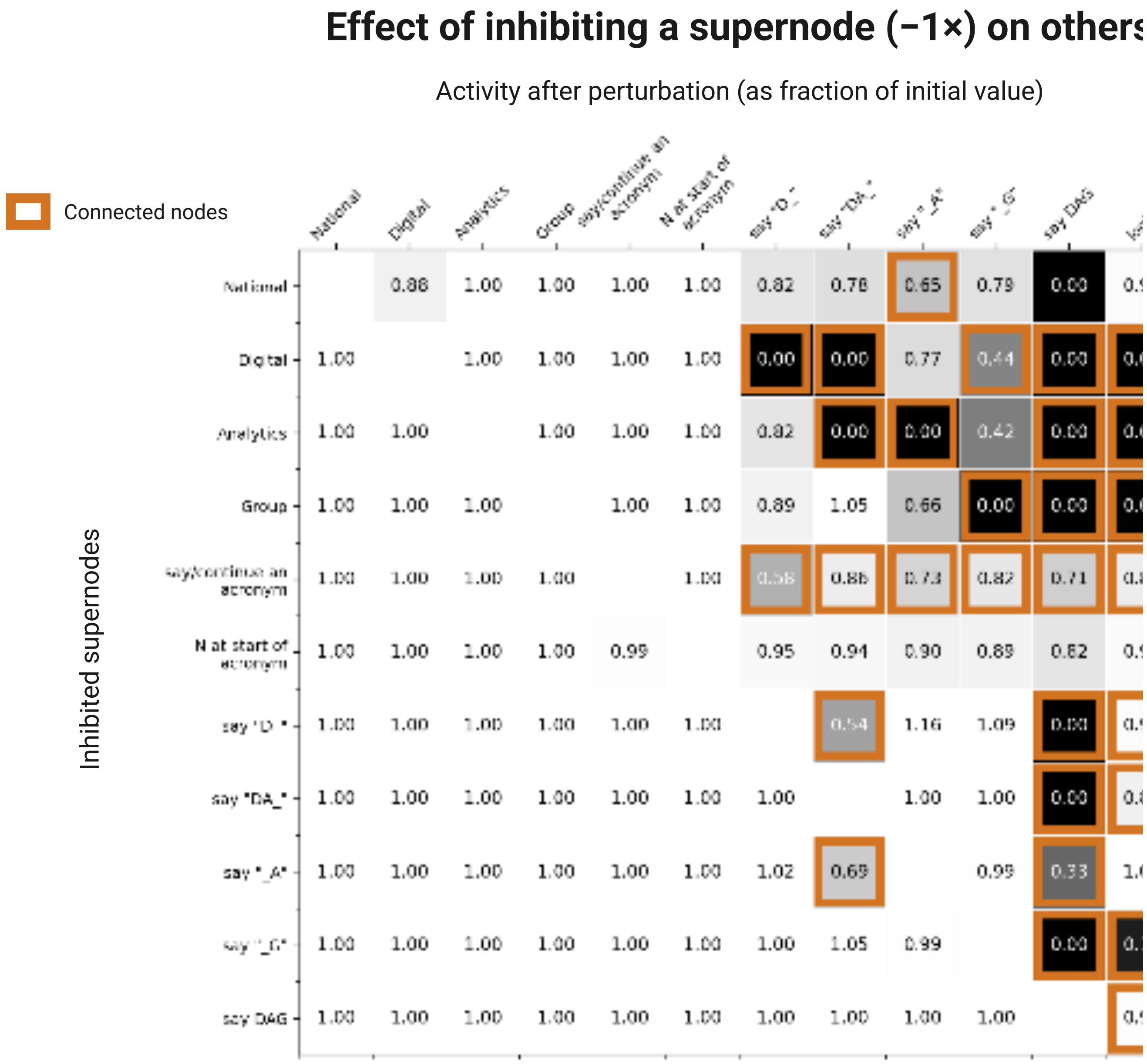
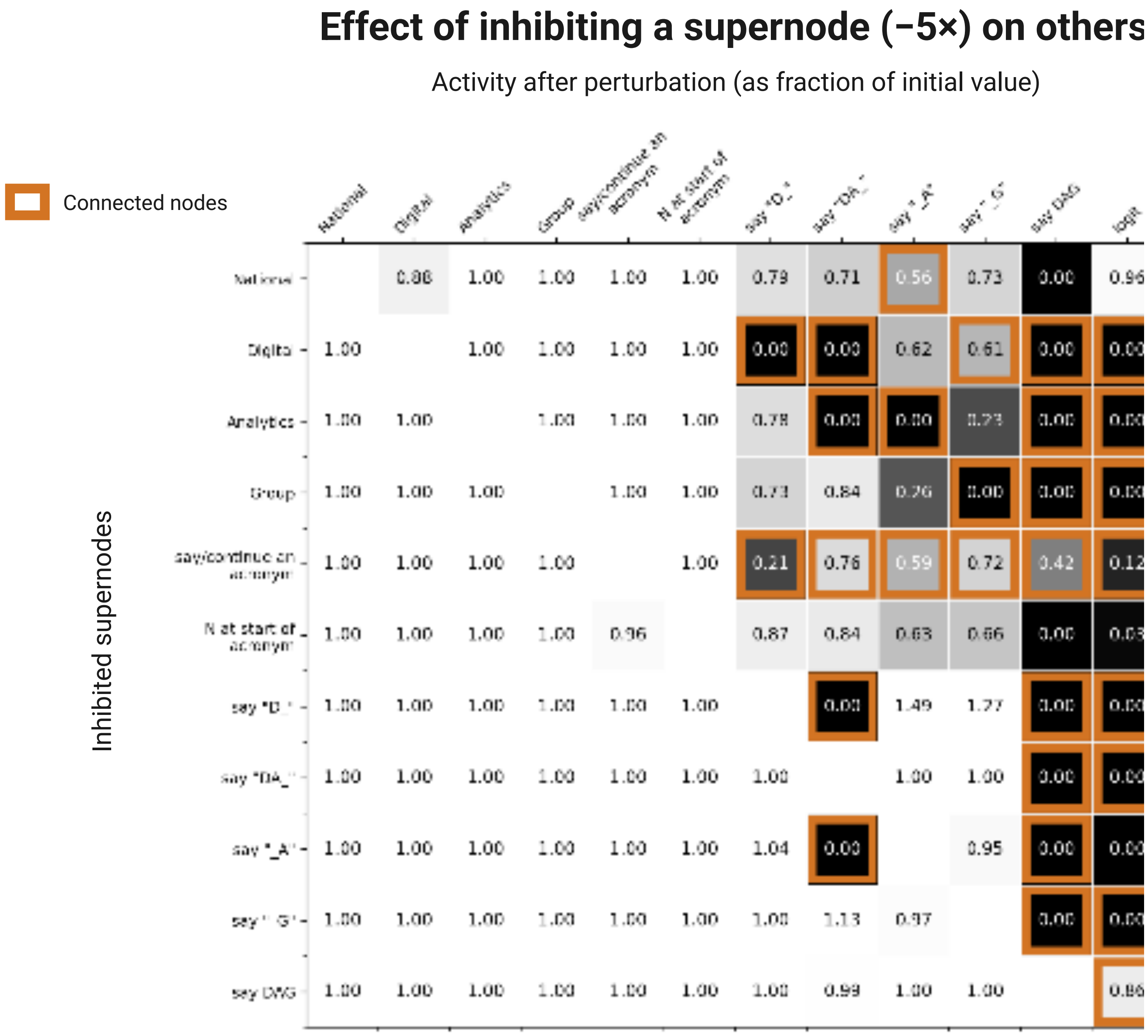


**Figure 40**: Intervention effects on feature activations for acronyms with a steering strength of 1

Supernodes require different steering strengths to show an effect because the strength of their outgoing edges varies. This strength depends both on the number of features in the supernode and their individual edge strengths. The steering plot above shows that inhibiting "say _A" or "say DA" has only a minor effect on the logit at a strength of −1. This is partially due to "say _A" containing only one feature, while supernodes like "say D_" contain six. "say _A"'s influence on the logit is also indirect and mediated by the "say DA" supernode. If we increase steering strength to −5, we do observe an effect from inhibiting "say _A".



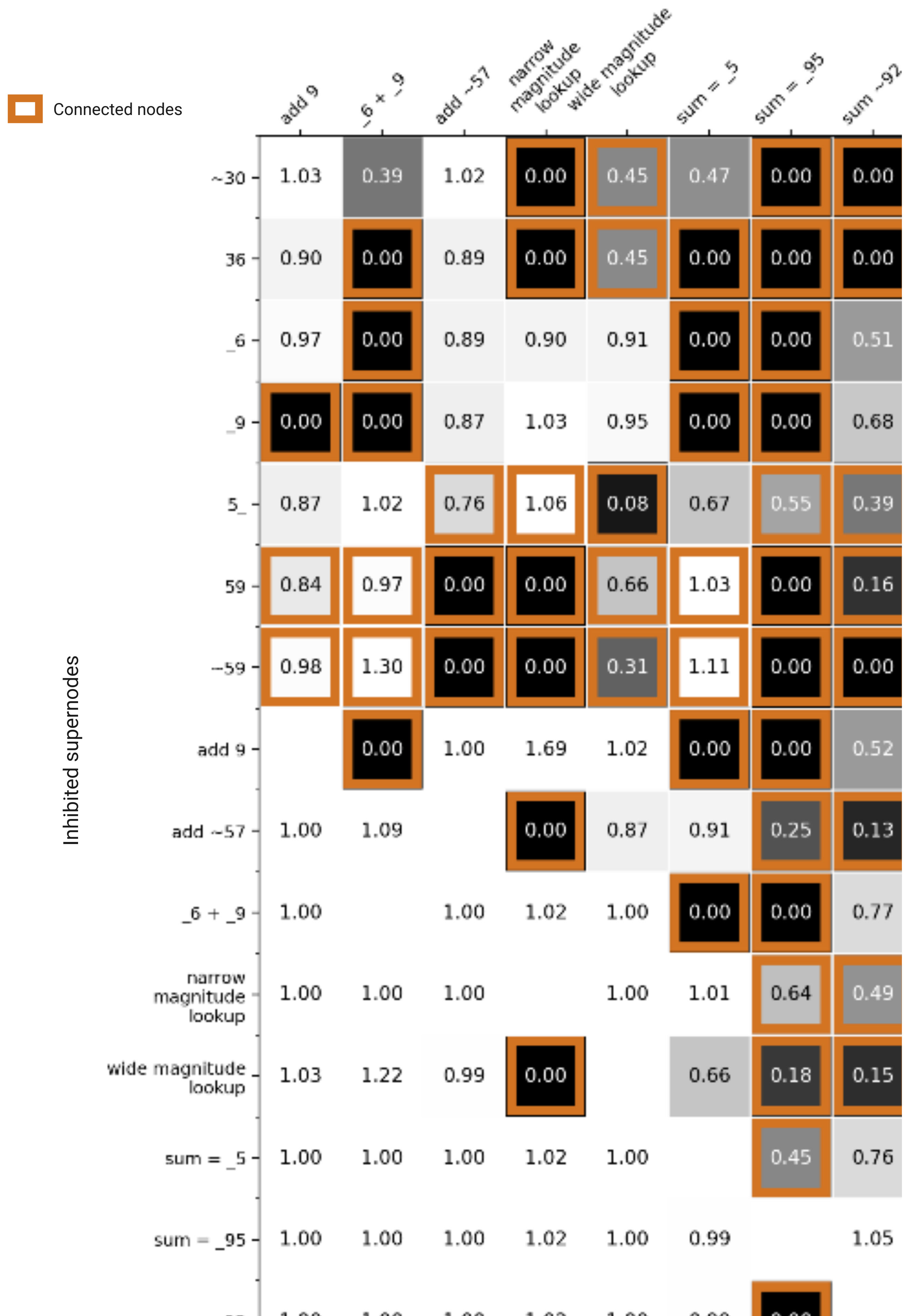
| | National | Digital | Analytics | Group | say/continue an acronym | N at start of acronym | say "D_" | say "DA_" | say "_A" | say "_G" | say DAG | logit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| National | | 0.88 | 1.00 | 1.00 | 1.00 | 1.00 | 0.79 | 0.71 | 0.56 | 0.73 | 0.00 | 0.96 |
| Digital | 1.00 | | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 | 0.00 | 0.62 | 0.61 | 0.00 | 0.00 |
| Analytics | 1.00 | 1.00 | | 1.00 | 1.00 | 1.00 | 0.78 | 0.00 | 0.00 | 0.23 | 0.00 | 0.00 |
| Group | 1.00 | 1.00 | 1.00 | | 1.00 | 1.00 | 0.73 | 0.84 | 0.26 | 0.00 | 0.00 | 0.00 |
| say/continue an acronym | 1.00 | 1.00 | 1.00 | 1.00 | | 1.00 | 0.21 | 0.76 | 0.59 | 0.72 | 0.42 | 0.12 |
| N at start of acronym | 1.00 | 1.00 | 1.00 | 1.00 | 0.96 | | 0.87 | 0.84 | 0.63 | 0.66 | 0.00 | 0.03 |
| say "D_" | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | | 0.00 | 1.49 | 1.77 | 0.00 | 0.00 |
| say "DA_" | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | | 1.00 | 1.00 | 0.00 | 0.00 |
| say "_A" | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.04 | 0.00 | | 0.95 | 0.00 | 0.00 |
| say "_G" | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.13 | 0.97 | | 0.00 | 0.00 |
| say DAG | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 1.00 | | 0.86 |

**Figure 41**: Intervention effects on feature activations for acronyms with a steering strength of 5

For reference, we also show the effect of inhibiting more of the supernodes in the 36+59 prompt.

**Effect of inhibiting a supernode (−2×) on others**

Activity after perturbation (as fraction of initial value)



| | add 9 | _6 + _9 | add ~57 | narrow magnitude lookup | wide magnitude lookup | sum = _5 | sum = _95 | sum ~92 |
|---|---|---|---|---|---|---|---|---|
| ~30 | 1.03 | 0.39 | 1.02 | 0.00 | 0.45 | 0.47 | 0.00 | 0.00 |
| 36 | 0.90 | 0.00 | 0.89 | 0.00 | 0.45 | 0.00 | 0.00 | 0.00 |
| _6 | 0.97 | 0.00 | 0.89 | 0.90 | 0.91 | 0.00 | 0.00 | 0.51 |
| _9 | 0.00 | 0.00 | 0.87 | 1.03 | 0.95 | 0.00 | 0.00 | 0.68 |
| 5_ | 0.87 | 1.02 | 0.76 | 1.06 | 0.08 | 0.67 | 0.55 | 0.39 |
| 59 | 0.84 | 0.97 | 0.00 | 0.00 | 0.66 | 1.03 | 0.00 | 0.16 |
| ~59 | 0.98 | 1.30 | 0.00 | 0.00 | 0.31 | 1.11 | 0.00 | 0.00 |
| add 9 | | 0.00 | 1.00 | 1.69 | 1.02 | 0.00 | 0.00 | 0.52 |
| add ~57 | 1.00 | 1.09 | | 0.00 | 0.87 | 0.91 | 0.25 | 0.13 |
| _6 + _9 | 1.00 | | 1.00 | 1.02 | 1.00 | 0.00 | 0.00 | 0.77 |
| narrow magnitude lookup | 1.00 | 1.00 | 1.00 | | 1.00 | 1.01 | 0.64 | 0.49 |
| wide magnitude lookup | 1.03 | 1.22 | 0.99 | 0.00 | | 0.66 | 0.18 | 0.15 |
| sum = _5 | 1.00 | 1.00 | 1.00 | 1.02 | 1.00 | | 0.45 | 0.76 |
| sum = _95 | 1.00 | 1.00 | 1.00 | 1.02 | 1.00 | 0.99 | | 1.05 |

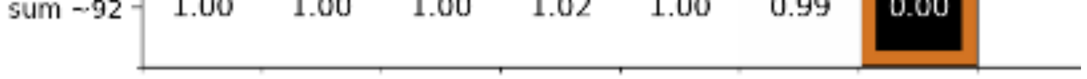


**Figure 42**: More intervention effects on feature activations for “calc: 36+59=”

## Notes on the Interface

We share an MIT licensed version of the interactive attribution graph interface used in this paper on github.

This interface has been simplified from the full version used internally for rapid exploration in favor of a publicly sharable version useful for inspecting labeled graphs. Analysis tools which were trimmed, and may be useful for practitioners to reimplement, include:

- Feature scatterplots comparing arbitrary scalars attached to features during the graph generation pipeline.
- In situ feature and supernode labeling.
- Feature examples and additional detail shown on hover to skim through features faster.
- A diff view comparing two attribution graphs, highlighting the difference and intersections in feature activations and edge weights.
- A list of all the active features, their labels and edge weight to the active node.

A rearrangeable grid containing these diverse “widgets” made it easier to experiment with new ideas and rescale the interface to show dozens or thousands of features.

## Residual Stream Norms Growth

In many models, the norm of the residual stream grows exponentially over the forward pass. Heimersheim and Turner [143] observed this norm growth in the GPT-2, OPT, and Pythia families, and conjectured it serves the functional purpose of making space for new information. Rather than deleting old information by explicitly writing its negative, a layer may simply write new information at a larger magnitude, drowning out the influence of the older information by increasing the denominator of the normalization denominators. Thus to keep information around and legible, the model must actively amplify in many layers. A cross-layer transcoder can absorb all of that into a single feature, while per-layer transcoders or neurons will have features (neurons) in many layers performing the same function, and all will appear in an attribution graph. This does occur in practice.

We observe the same phenomenon in our research model, 18L, which also has normalization denominators.

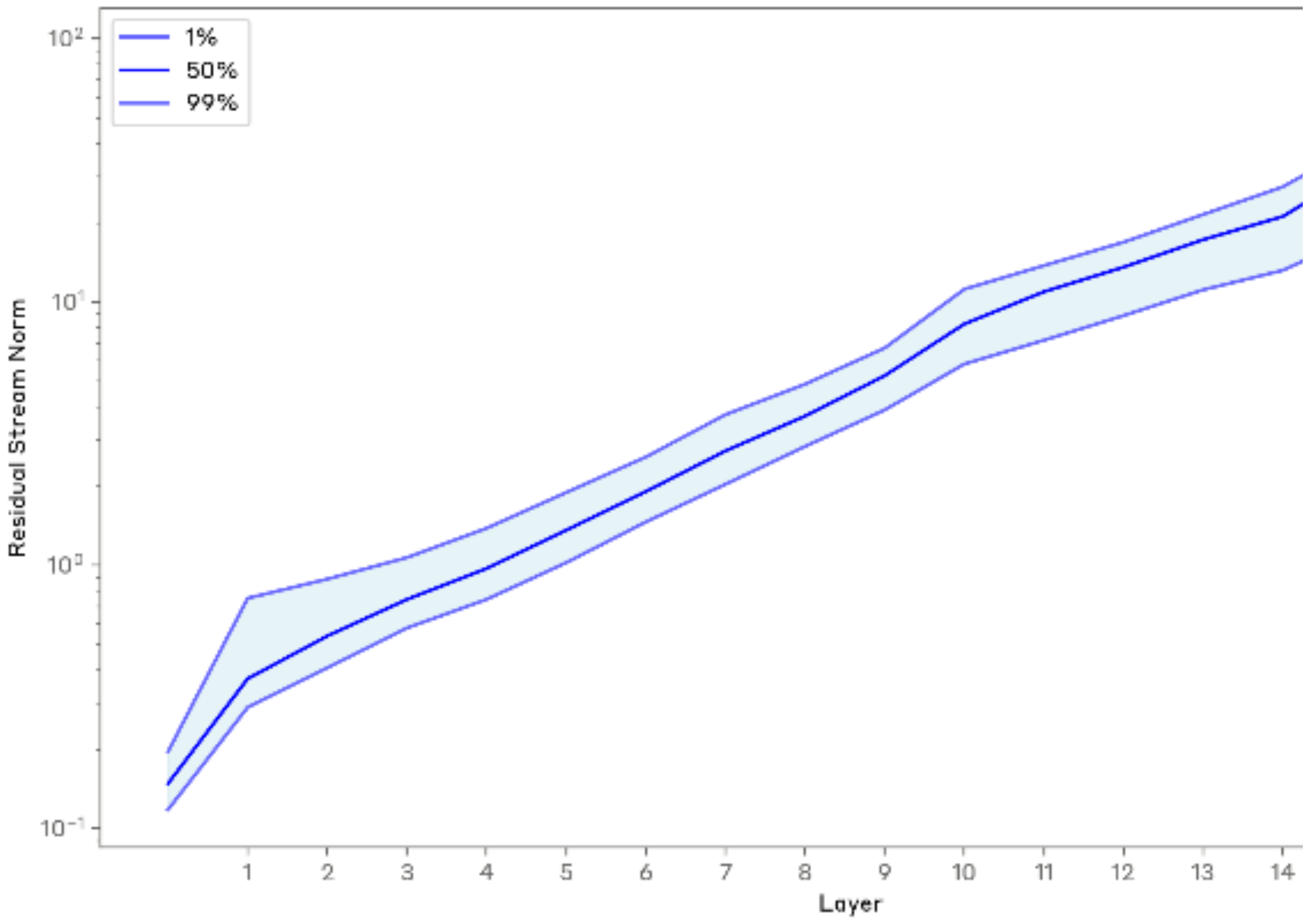


**Figure 43**: L2 Norm of residual stream activations by layer in 18L.

## Interference Weights over More Features

In the main text, we showed a single feature's virtual weights, and found that many large connections had weak coactivation statistics. Here, we run a similar analysis on a larger sample of features to better understand if this pattern generalizes.

We begin with the set of features that activate on our addition prompts and appear in our pruned attribution graphs. From these, we sample 1000 features with equal representation across layers. We then compute the virtual weights and coactivation statistics for all input and output connections of these features using a dataset of ~150M tokens of the CLT's training data.

Below, we show the virtual weight distribution across all of these connections, and compare it to the distribution of weights between pairs that ever coactivate on the dataset. The percentage of coactive weights within each bin appears as text labels in the figure.

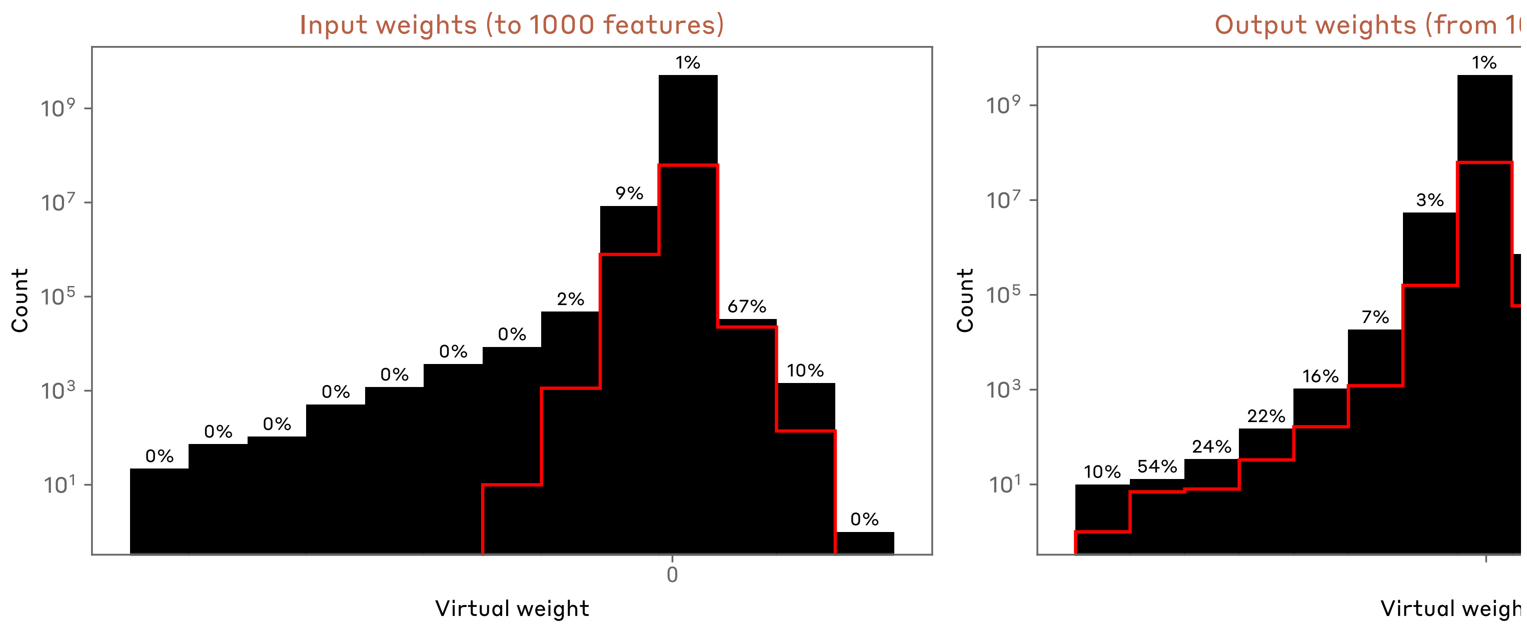


**Figure 44**: Virtual weights for 1000 features, and their weights with coactive partners.

We consider weights between features with no coactivations to be "unused" in our dataset; a source feature that never coactivates with a target feature likely never causally affects the target's activation value.[44]

In general, a large mass of small weights dominates the distribution (as expected). The smallest weights are very often unused with many bins having single-digit "use percentages." Large negative weights also stand out as a difference between the raw and coactive distributions, yet this result isn't surprising (even without interference); if an upstream feature is active, such a strong negative weight might guarantee that its downstream target never activates.

More significantly, we also observe that even among extremely large positive weights (beyond the 99.9999th percentile), a significant proportion are unused. This is not conclusive proof of interference, because we may still miss crucial tokens in our sampling where these edges are used. Still, we take this analysis as suggestive of interference in 18L.

## Number Output Weights for Select Features

We show the full output weights (over tokens 0, 1, ..., 999) for three features whose output patterns are neither periodic nor approximate magnitudes. The first promotes "simple numbers", promoting both smaller numbers and rounder numbers. The second promotes numbers starting with 9, and the latter promotes numbers starting (or, to a lesser degree, ending) with 95. (In the color scheme, upweighted tokens are red and downweighted tokens are blue.)

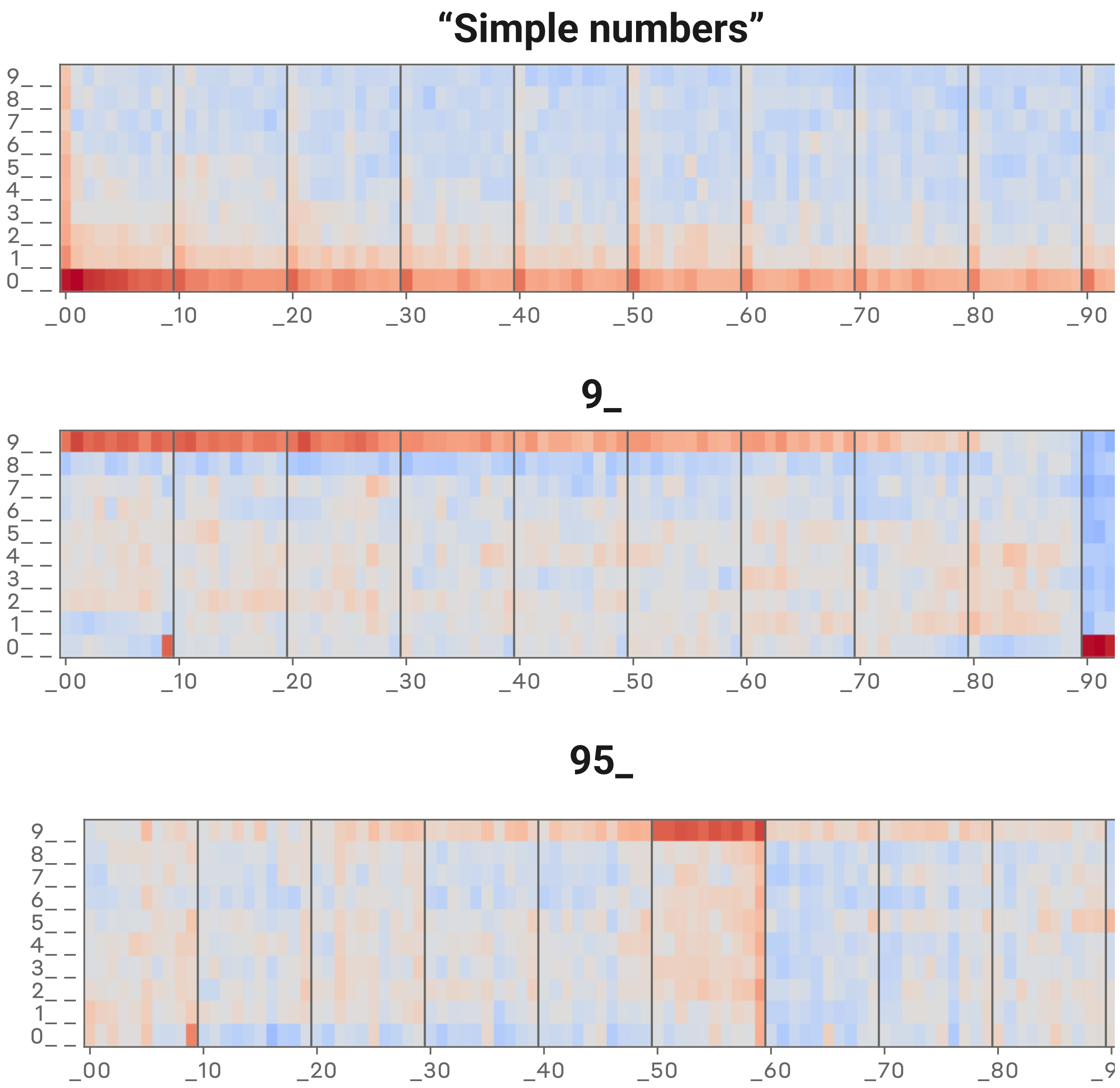


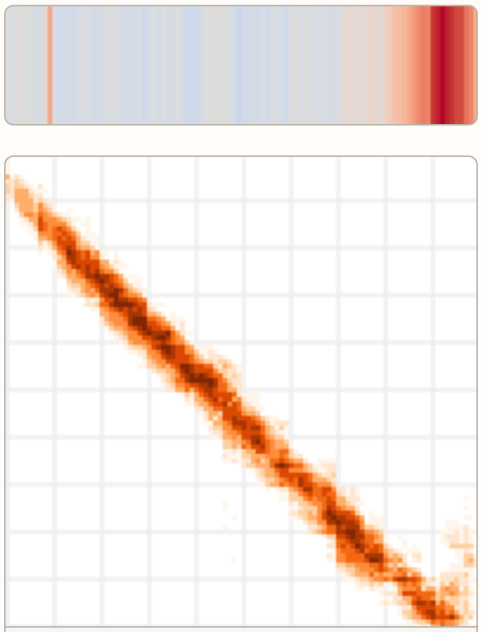
**Figure 45**: Output weight plots for CLT features that aren't as easily described by Fourier components.

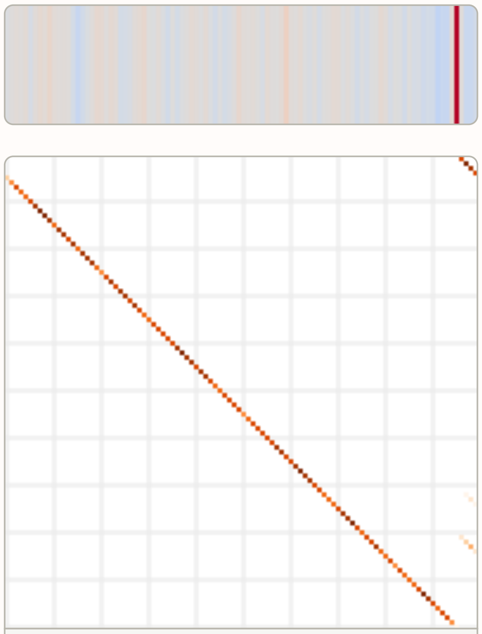
## Comparison of Addition Features Between Small and Large Models

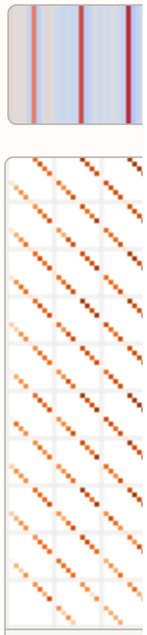
We find similar families of features in Haiku and 18L for performing addition, but note that the ones for 18L are less precise. 18L performs slightly worse on these prompts (98.6% top-1 accuracy vs. 100% for Haiku 3.5), which possibly reflects this imprecision.

## SUM FEATURES

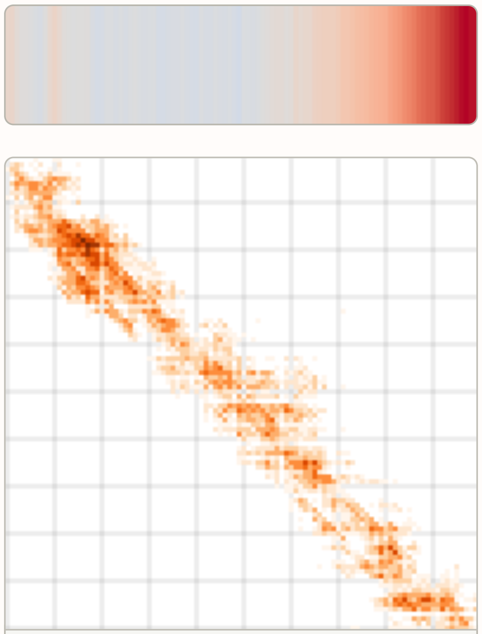
**Haiku**

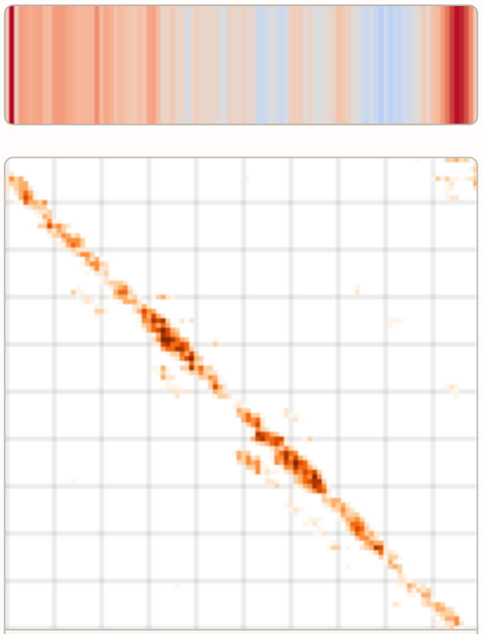

sum ~92

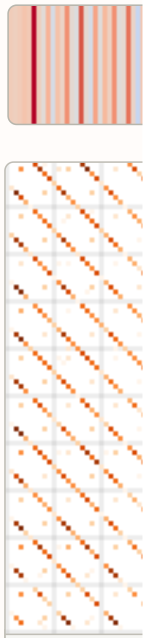

sum = _95



**18L**

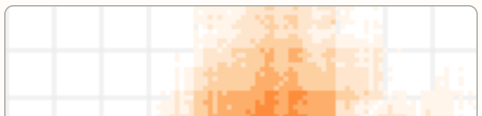

sum ~92

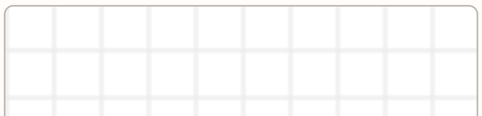

sum ~_95

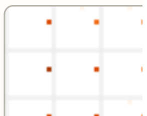


## LOOKUP TABLE FEATURES

**Haiku**







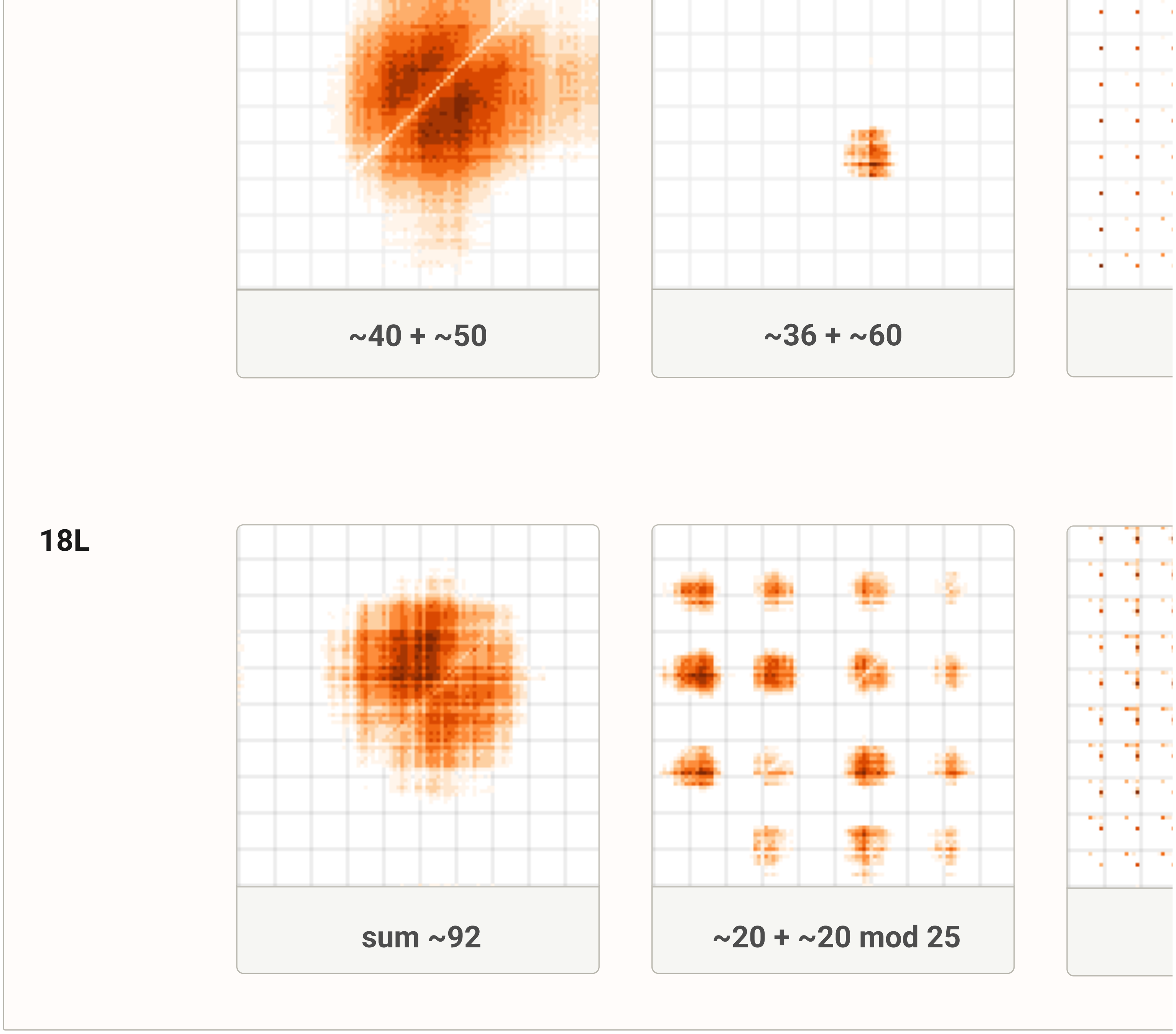


**Figure 46**: A comparison of similar features' operand plots between Haiku and 18L.

## Additional Evaluation Details

To evaluate our replacement model and attribution graphs, we curated a dataset of random but nontrivial pretraining tokens. Specifically, we considered short sequences from the Pile (minus books) dataset [27] and target tokens where

- The underlying model made the correct top-1 prediction;
- The token has a dataset density less than `1e-4` to filter out stop tokens and common tokens like articles, punctuations, and newlines;
- The token did not appear elsewhere in the context (in these cases we often have high graph scores but that the graph is largely mediated by attention which we do not explain);
- The token has loss >0.2 to filter out other trivial predictions coming from completing multi-token words, fuzzy induction, memorized text, etc.

Due to the expense of graph generation, we used a dataset size of n=260 for graph evaluation.

## Prompt and Graph Lists

The below table contains a random sample of prompts and graphs for our Pile minus books evaluation set. It includes graphs for both our 10m crosslayer transcoder (CLT) and 10m per-layer transcoder (PLT) dictionaries on our 18L model.

| Links | Prompt | Target |
|---|---|---|
| clt plt | `arrangement, the optical fiber 104 is optically` | `coupled` |
| clt plt | `" "I already called the police." "They\'re almost here." "You got to go." "Goodbye." "Good` | `luck` |
| clt plt | `types, such as a front projection type and a rear projection type, depending on how images are` | `projected` |
| clt plt | `for grants at the moment to pay a few students a paltry sum to stay and work in her lab over the` | `summer` |
| clt plt | `invention. In a separate aspect, the invention provides a method of potentiating the actions of other CNS active compounds. This method comprises administering an` | `effective` |

We also include a sample of the basic curated prompts we used in the development of our techniques and their corresponding CLT and PLT graphs. These were designed to exercise basic capabilities such as factual recall, analogical reasoning, memorization, arithmetic, multilinguality, and in-context learning.

| Links | Prompt | Target |
|---|---|---|
| clt plt | `Fact: Michael Jordan plays the sport of` | `basketball` |
| clt plt | `Fait: Michael Jordan joue au` | `basket` |
| clt plt | `At first, Sally hated school. But over time she changed her mind. Now she is` | `happy` |
| clt plt | `The International Advanced Security Group (IA` | `SG` |
| clt plt | `The National Digital Analytics Group (N` | `DAG` |
| clt plt | `7 14 21 28 35` | `42` |
| clt plt | `grass: green sky: blue corn: yellow carrot: orange strawberry:` | `red` |
| clt plt | `The opposite of "hot" is "` | `cold` |
| clt plt | `The opposite of "small" is "` | `big` |
| clt plt | `5 + 3 =` | `8` |
| clt plt | `Examiner interviews are available via` | `telephone` |
| clt plt | `Mexico:peso :: Europe:` | `euro` |
| clt plt | `Zagreb:Croatia :: Copenhagen:` | `Denmark` |
| clt plt | `def customer_spending(transaction_df): for customer_id, customer_df in transaction_df.` | `group` |
| clt plt | `La saison après le printemps s'appelle l'` | `été` |
| clt plt | `a = "Craig" assert a[0] == "` | `C` |

## Footnotes

1. An alternative approach is to study the roles of gross model components such as entire MLP blocks and attention heads [1, 2, 3]. This approach has identified interesting roles these components play in specific behaviors, but large components play a multitude of unrelated roles across the data distribution, so we seek a more granular decomposition. [↩]

2. We use 'feature' following the tradition of 'feature detectors' in neuroscience and 'feature learning' in machine learning. Some recent literature uses the term 'latent,' which refers to specific vectors in the model's latent space. We find 'feature' better captures the computational role these elements play, making it more appropriate for describing transcoder neurons than SAE decoder vectors. [↩]

3. Direct feature-feature interactions are linear because transcoder features “bridge over” the MLP nonlinearities, replacing their computation, and because we've frozen the remaining nonlinearities: attention patterns and normalization denominators. It's worth noting that strictly we mean that the pre-activation of a feature is linear with respect to the activations of earlier features.
Freezing attention patterns is a standard approach which divides understanding transformers into two steps: understanding behavior given attention patterns, and understanding why the model attends to those positions. This approach was explored in depth for attention models in *A Mathematical Framework* [18], which also discussed a generalization to MLP layers which is essentially the approach used in this paper. Note that factoring out understanding attention patterns in this way leads to the issues with attention noted in § 7.1 Limitations: Missing Attention Circuits. However, we can then also take the same solution *Framework* takes of studying QK circuits. [↩]

4. Other architectures besides CLTs can be used for circuit analysis, but we’ve empirically found this approach to work well. [↩]

5. 18L has no MLP in layer 0, so our CLT has 17 layers. [↩]

6. We use a randomly sampled set of prompts and target tokens, restricting to those which the model predicts correctly, but with a confidence lower than 80% (to filter out “boring” tokens). [↩]

7. This is similar in spirit to a Taylor approximation of a function $f$ at a point $a$; both agree locally in a neighborhood of $a$ but diverge in behavior as you move away. [↩]

8. Due to the “Caps Lock” token, the actual target token is “dag”. We write the token in uppercase here and in the rest of the text for ease of reading. [↩]

9. We chose this threshold arbitrarily. Empirically, fewer than three logits are required to capture 95% of the probability in the cases we study. [↩]

10. Alternatively, w_{s \rightarrow t} is the derivative of the preactivation of $t$ with respect to the source feature activation, with stop-gradients on all non-linearities in the local replacement model. [↩]

11. That is, our model ignores the "QK-circuits" but captures the "OV-circuits". [18] [↩]

12. We sometimes used labels from our automated interpretability pipeline as a starting point, but generally found human labels to be more reliable. [↩]

13. They can be indirectly affected since they sit downstream of affected nodes. [↩]

14. For a discussion of why we steer negatively instead of ablating the feature, see § I Unexplained Variance and Choice of Steering Factors. [↩]

15. For each patched supernode, we choose the end-layer range which causes the largest suppression effect on the logit. [↩]

16. Note that we use a large steering factor in this experiment. For a discussion of this, see § I Unexplained Variance and Choice of Steering Factors. [↩]

17. We use the prefix "calc:" because the 18L performs much better on the problem with it. This prefix is not necessary for Haiku 3.5, but we include it nevertheless for the purposes of direct comparison in later sections. [↩]

18. During analysis we visualize effects on [0, 999]. This is important to understand effects beyond the first 100 number tokens (e.g., the feature predicting 95 mod 100), but we only show [0, 99] for simplicity. [↩]

19. With hover you can see the variability in the precision of the low precision lookup table features and the moderate precision sum features. [↩]

20. As in other sections of this work, we apply these labels manually. [↩]

21. Write $f_{pre}$ for the preactivation of $x$. Then $f_{pre}(x+y) + f_{pre}(z+w) = f_{pre}(x+w) + f_{pre}(y+z)$. If both terms on the LHS are positive, at least one on the right must be. [↩]

22. Concretely, the intermediate info is the ones digit produced from adding the ones digit of the operands and the approximate magnitude of the sum of two numbers in the 20s. [↩]

23. The attention-direct terms can also be written in terms of virtual weights given by multiplying various decoder vectors by a series of attention

head OVs, and then by an encoder, but these get scaled on a prompt by both the source feature activation and the attention patterns, which make their analysis more complex. [↩]

24. We also see interference weights when looking at a larger sample of features in the § O Appendix: Interference Weights over More FeaturesAppendix. [↩]

25. It only represents the residual-direct component and does not include the attention-direct one. [↩]

26. Note that these weights cannot be 0 as this would also send TWERA to 0. [↩]

27. There's no guarantee that the CLT features are the most parsimonious way to split up the computation, and it's possible some of our less important, roughly-periodic features which are harder to interpret are artifacts of the periodic aspects of the representation. [↩]

28. In § P Appendix: Number Output Weights over More Features, we show output weight plots for the 9_ and 95_ features on all number predictions from [0,999]. We also show a miscellaneous feature that promotes "simple numbers": small numbers, multiples of 100 and a few standouts like 360. [↩]

29. One route is to consider *families* of prompts, and to exclude from considerations features that are present across all prompts within a family (but see [26]). [↩]

30. We scale the number of training steps with the number of features, so improvements reflect a combination of both forms of scaling – see § D Appendix: CLT Implementation Details for details. [↩]

31. Specifically, we sweep over a range of scalar thresholds, and for each value we clamp all neurons with activation less than the threshold to 0. For our metrics and graphs, we then only consider neurons above the threshold. [↩]

32. We choose the threshold at approximately the point where the neurons achieve similar automated interpretability scores as our smallest dictionaries, see figures above. [↩]

33. If there are multiple logit nodes, we compute an average of the rows weighted by logit probability. [↩]

34. We normalize influence scores by the total influence of embeddings in the unpruned graph. This normalization factor is exactly equal to the replacement score, which we define in the next section. [↩]

35. Compounding errors have a gradually detrimental effect on the faithfulness of the direction of perturbation effects, which are largely consistent across CLT sizes, with signs of faithfulness worsening slightly as dictionary size increases. Compounding errors can have a catastrophically detrimental effect on the magnitude of perturbations, with worse effects for larger dictionaries. We suspect the lack of normalization denominators in the local replacement model may be why its perturbation effect magnitudes deviate so significantly from the underlying model, even when the perturbation effect directions are significantly correlated. [↩]

36. Other explanations are possible, particularly for the direct paths from `"B"` that do not go through `tokens following "B"` – one alternative is that the model may use a "binding ID vector" [31] to group the `"B"` tokens with `"1945"` and nearby tokens and use this to attend directly back to the `"B"` token from the final token position – see Feng & Steinhardt [31] for more details on this type of mechanism. [↩]

37. For example, the fundamental reason we need features to be independently interpretable is to avoid needing to think about all of them at once (see discussion here). [↩]

38. See Sharkey *et al.* [32] for a detailed description of the reverse engineering philosophy. [↩]

39. Credit attribution in a non-linear setting is a hard problem. The core challenge can't simply be washed away, and it's worth asking where we implicitly have pushed the complexity. There are three places it may have gone. First – and this is the best option – is that the non-linear interactions have become multi-step paths in our attribution graph, which can then be reasoned about in interpretable ways. Next, a significant amount of it must have gone into the frozen components, factored into separate questions we haven't tried to address here, but at least know remain. But there is also a bad option: some of it may have been simplified by our CLT taking a non-mechanistically faithful shortcut, which approximates MLP computation with a linear approximation that is often correct. [↩]

40. Our setup allows for some similar principled notions of linear interaction in the global context, but we now have to think of different weights for interactions along different paths – for example, what is the interaction of feature A and feature B mediated by attention head H? This is discussed in § 4 Global Weights. The *Framework* paper also discussed these general conceptual issues in the appendix. [↩]

41. Analyzing the global weights mediated by a head may be interesting here. For example, an induction head might systematically move "I'm X" features to "say X" features. A successor head might systematically map "X" features to "X+1" features. [↩]

42. Of course, not all heads are individually interesting in the way induction heads or successor heads often are. We might suspect there are many more "attentional features" like induction and succession hiding in *superposition over attention heads*. If we could reveal these, there might be a much richer story. [↩]

43. We also freeze attention patterns in the underlying model. [↩]

44. This interpretation is not strictly true with negative weights, as a strong negative weight might actually prevent coactivation. [↩]

## References

1. **Locating and editing factual knowledge in gpt** [link]
Meng, K., Bau, D., Andonian, A. and Belinkov, Y., 2022. arXiv preprint arXiv:2202.05262.
2. **Interpretability in the wild: a circuit for indirect object identification in gpt-2 small** [link]
Wang, K., Variengien, A., Conmy, A., Shlegeris, B. and Steinhardt, J., 2022. arXiv preprint arXiv:2211.00593.
3. **Towards automated circuit discovery for mechanistic interpretability** [PDF]
Conmy, A., Mavor-Parker, A., Lynch, A., Heimersheim, S. and Garriga-Alonso, A., 2023. Advances in Neural Information Processing Systems, Vol 36, pp. 16318--16352.
4. **Thread: Circuits** [link]
Cammarata, N., Carter, S., Goh, G., Olah, C., Petrov, M., Schubert, L., Voss, C., Egan, B. and Lim, S.K., 2020. Distill.
5. **Linear algebraic structure of word senses, with applications to polysemy** [PDF]
Arora, S., Li, Y., Liang, Y., Ma, T. and Risteski, A., 2018. Transactions of the Association for Computational Linguistics, Vol 6, pp. 483--495. MIT Press.
6. **Decoding The Thought Vector** [link]
Goh, G., 2016.
7. **Toy Models of Superposition** [HTML]
Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M. and Olah, C., 2022. Transformer Circuits Thread.
8. **Towards Monosemanticity: Decomposing Language Models With Dictionary Learning** [HTML]
Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J.E., Hume, T., Carter, S., Henighan, T. and Olah, C., 2023. Transformer Circuits Thread.
9. **Sparse Autoencoders Find Highly Interpretable Model Directions** [link]
Cunningham, H., Ewart, A., Smith, L., Huben, R. and Sharkey, L., 2023. arXiv preprint arXiv:2309.08600.
10. **Scaling and evaluating sparse autoencoders** [link]
Gao, L., la Tour, T.D., Tillman, H., Goh, G., Troll, R., Radford, A., Sutskever, I., Leike, J. and Wu, J., 2024. arXiv preprint arXiv:2406.04093.
11. **Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders** [link]
Rajamanoharan, S., Lieberum, T., Sonnerat, N., Conmy, A., Varma, V., Kramar, J. and Nanda, N., 2024. arXiv preprint arXiv:2407.14435.
12. **Transcoders find interpretable LLM feature circuits** [PDF]
Dunefsky, J., Chlenski, P. and Nanda, N., 2025. Advances in Neural Information Processing Systems, Vol 37, pp. 24375--24410.
13. **dictionary_learning Github Repository** [link]
Marks, S., Karvonen, A. and Mueller, A., 2024. Github.
14. **Predicting Future Activations** [link]
Templeton, A., Batson, J., Jermyn, A. and Olah, C., 2024.
15. **Sparse Crosscoders for Cross-Layer Features and Model Diffing** [HTML]
Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J. and Olah, C., 2024.
16. **Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models** [link]
Marks, S., Rager, C., Michaud, E.J., Belinkov, Y., Bau, D. and Mueller, A., 2024. arXiv preprint arXiv:2403.19647.
17. **Automatically identifying local and global circuits with linear computation graphs** [link]
Ge, X., Zhu, F., Shu, W., Wang, J., He, Z. and Qiu, X., 2024. arXiv preprint arXiv:2405.13868.
18. **A Mathematical Framework for Transformer Circuits** [HTML]
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., 2021. Transformer Circuits Thread.
19. **Fact Finding: Attempting to Reverse-Engineer Factual Recall on the Neuron Level** [link]

19. Fact Finding: Attempting to Reverse-Engineer Factual Recall on the Neuron Level [link]
Nanda, N., Rajamanoharan, S. and Shah, R., 2023.

20. Using Features For Easy Circuit Identification [link]
Batson, J., Chen, B. and Jones, A., 2024.

21. Arithmetic Without Algorithms: Language Models Solve Math With a Bag of Heuristics [link]
Nikankin, Y., Reusch, A., Mueller, A. and Belinkov, Y., 2024.

22. Language Models Use Trigonometry to Do Addition [link]
Kantamneni, S. and Tegmark, M., 2025.

23. Pre-trained large language models use fourier features to compute addition [link]
Zhou, T., Fu, D., Sharan, V. and Jia, R., 2024. arXiv preprint arXiv:2406.03445.

24. Sparse Autoencoders Work on Attention Layer Outputs [link]
Kissane, C., robertzk,, Conmy, A. and Nanda, N., 2024.

25. Superposition, Memorization, and Double Descent [HTML]
Henighan, T., Carter, S., Hume, T., Elhage, N., Lasenby, R., Fort, S., Schiefer, N. and Olah, C., 2023. Transformer Circuits Thread.

26. Transformer circuit faithfulness metrics are not robust [link]
Miller, J., Chughtai, B. and Saunders, W., 2024. arXiv preprint arXiv:2407.08734.

27. The Pile: An 800GB Dataset of Diverse Text for Language Modeling [link]
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., 2020.

28. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset [link]
Zheng, L., Chiang, W., Sheng, Y., Li, T., Zhuang, S., Wu, Z., Zhuang, Y., Li, Z., Lin, Z., Xing, E.P., Gonzalez, J.E., Stoica, I. and Zhang, H., 2023.

29. Interpretability Evals for Dictionary Learning [link]
Lindsey, J., Cunningham, H., Conerly, T. and Templeton, A., 2024.

30. In-context Learning and Induction Heads [HTML]
Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S. and Olah, C., 2022. Transformer Circuits Thread.

31. How do language models bind entities in context? [link]
Feng, J. and Steinhardt, J., 2023. arXiv preprint arXiv:2310.17191.

32. Open Problems in Mechanistic Interpretability [link]
Sharkey, L., Chughtai, B., Batson, J., Lindsey, J., Wu, J., Bushnaq, L., Goldowsky-Dill, N., Heimersheim, S., Ortega, A., Bloom, J. and others,, 2025. arXiv preprint arXiv:2501.16496.

33. Are Sparse Autoencoders Useful? A Case Study in Sparse Probing [link]
Kantamneni, S., Engels, J., Rajamanoharan, S., Tegmark, M. and Nanda, N., 2025. arXiv preprint arXiv:2502.16681.

34. AXBENCH: Steering LLMs? Even Simple Baselines Outperform Sparse Autoencoders [link]
Wu, Z., Arora, A., Geiger, A., Wang, Z., Huang, J., Jurafsky, D., Manning, C.D. and Potts, C., 2025. arXiv preprint arXiv:2501.17148.

35. A is for absorption: Studying feature splitting and absorption in sparse autoencoders [link]
Chanin, D., Wilken-Smith, J., Dulka, T., Bhatnagar, H. and Bloom, J., 2024. arXiv preprint arXiv:2409.14507.

36. Do sparse autoencoders find "true features"? [link]
Till, D., 2024.

37. Measuring feature sensitivity using dataset filtering [link]
Turner, N.L., Jermyn, A. and Batson, J., 2024.

38. Matryoshka Sparse Autoencoders [link]
Nabeshima, N., 2024.

39. Learning Multi-Level Features with Matryoshka SAEs [link]
Bussmann, B., Leask, P. and Nanda, N., 2024.

40. Showing SAE Latents Are Not Atomic Using Meta-SAEs [link]
Bussmann, B., Pearce, M., Leask, P., Bloom, J., Sharkey, L. and Nanda, N., 2024.

41. **Monitor: An AI-Driven Observability Interface** [link]
Meng, K., Huang, V., Chowdhury, N., Choi, D., Steinhardt, J. and Schwettmann, S., 2024.

42. **Axiomatic attribution for deep networks** [link]
Sundararajan, M., Taly, A. and Yan, Q., 2017. arXiv preprint arXiv:1703.01365.

43. **A unified approach to interpreting model predictions** [PDF]
Lundberg, S.M. and Lee, S., 2017. Advances in neural information processing systems, Vol 30.

44. **Successor Heads: Recurring, Interpretable Attention Heads In The Wild** [link]
Gould, R., Ong, E., Ogden, G. and Conmy, A., 2023.

45. **Toward transparent ai: A survey on interpreting the inner structures of deep neural networks** [link]
Räuker, T., Ho, A., Casper, S. and Hadfield-Menell, D., 2023. 2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), pp. 464--483.

46. **Mechanistic Interpretability for AI Safety--A Review** [link]
Bereska, L. and Gavves, E., 2024. arXiv preprint arXiv:2404.14082.

47. **A Primer on the Inner Workings of Transformer-based Language Models** [link]
Ferrando, J., Sarti, G., Bisazza, A. and Costa-jussa, M.R., 2024. arXiv preprint arXiv:2405.00208.

48. **The quest for the right mediator: A history, survey, and theoretical grounding of causal interpretability** [link]
Mueller, A., Brinkmann, J., Li, M., Marks, S., Pal, K., Prakash, N., Rager, C., Sankaranarayanan, A., Sharma, A.S., Sun, J. and others,, 2024. arXiv preprint arXiv:2408.01416.

49. **Efficient estimation of word representations in vector space** [link]
Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. arXiv preprint arXiv:1301.3781.

50. **Visualizing and understanding recurrent networks** [link]
Karpathy, A., Johnson, J. and Fei-Fei, L., 2015. arXiv preprint arXiv:1506.02078.

51. **Curve Detectors** [link]
Cammarata, N., Goh, G., Carter, S., Schubert, L., Petrov, M. and Olah, C., 2020. Distill.

52. **Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned** [link]
Voita, E., Talbot, D., Moiseev, F., Sennrich, R. and Titov, I., 2019. arXiv preprint arXiv:1905.09418.

53. **Tensor2tensor transformer visualization** [link]
Jones, L., 2017.

54. **A primer in bertology: What we know about how bert works** [link]
Rogers, A., Kovaleva, O. and Rumshisky, A., 2020. Transactions of the Association for Computational Linguistics, Vol 8, pp. 842--866. MIT Press. DOI: 10.1162/tacl_a_00349

55. **The Building Blocks of Interpretability** [link]
Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. and Mordvintsev, A., 2018. Distill. DOI: 10.23915/distill.00010

56. **A multiscale visualization of attention in the transformer model** [link]
Vig, J., 2019. arXiv preprint arXiv:1906.05714.

57. **Representation learning: A review and new perspectives** [link]
Bengio, Y., Courville, A. and Vincent, P., 2013. IEEE transactions on pattern analysis and machine intelligence, Vol 35(8), pp. 1798--1828. IEEE.

58. **Compressed sensing** [PDF]
Donoho, D.L., 2006. IEEE Transactions on information theory, Vol 52(4), pp. 1289--1306. IEEE.

59. **Compressed Sensing, Sparsity, and Dimensionality in Neuronal Information Processing and Data Analysis** [link]
Ganguli, S. and Sompolinsky, H., 2012. Annual Review of Neuroscience, Vol 35(1), pp. 485-508. DOI: 10.1146/annurev-neuro-062111-150410

60. **Sparse coding with an overcomplete basis set: A strategy employed by V1?** [link]
Olshausen, B.A. and Field, D.J., 1997. Vision research, Vol 37(23), pp. 3311--3325. Elsevier. DOI: 10.1016/S0042-6989(97)00169-7

61. **Sparse and redundant representations: from theory to applications in signal and image processing**
Elad, M., 2010. , Vol 2(1). Springer.

62. **Local vs. Distributed Coding** [link]
Thorpe, S.J., 1989. Intellectica, Vol 8, pp. 3--40.

63. **Unsupervised representation learning with deep convolutional generative adversarial networks** [link]
Radford, A., Metz, L. and Chintala, S., 2015. arXiv preprint arXiv:1511.06434.

64. **Activation Addition: Steering Language Models Without Optimization** [link]
Turner, A.M., Thiergart, L., Udell, D., Leech, G., Mini, U. and MacDiarmid, M., 2023.

65. **Zoom In: An Introduction to Circuits** [link]
Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M. and Carter, S., 2020. Distill. DOI: 10.23915/distill.00024.001

66. **Distributed Representations: Composition & Superposition** [HTML]
Olah, C., 2023.

67. **Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors** [link]
Yun, Z., Chen, Y., Olshausen, B.A. and LeCun, Y., 2021. arXiv preprint arXiv:2103.15949.

68. **Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet** [HTML]
Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N.L., McDougall, C., MacDiarmid, M., Freeman, C.D., Sumers, T.R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C. and Henighan, T., 2024. Transformer Circuits Thread.

69. **Sparse autoencoder** [PDF]
Ng, A. and others,, 2011. CS294A Lecture notes, Vol 72(2011), pp. 1--19.

70. **k-Sparse Autoencoders** [link]
Makhzani, A. and Frey, B.J., 2013. CoRR, Vol abs/1312.5663.

71. **Addressing Feature Suppression in SAEs** [link]
Wright, B. and Sharkey, L., 2024.

72. **Sparse Autoencoders Do Not Find Canonical Units of Analysis** [link]
Leask, P., Bussmann, B., Pearce, M., Bloom, J., Tigges, C., Moubayed, N.A., Sharkey, L. and Nanda, N., 2025. arXiv preprint arXiv:2502.04878.

73. **Saes are highly dataset dependent: A case study on the refusal direction** [link]
Kissane, C., Krzyzanowski, R., Nanda, N. and Conmy, A., 2024. Alignment Forum.

74. **Sparse Autoencoders Trained on the Same Data Learn Different Features** [link]
Paulo, G. and Belrose, N., 2025. arXiv preprint arXiv:2501.16615.

75. **Language models can explain neurons in language models** [HTML]
Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J. and Saunders, W., 2023.

76. **Automatically interpreting millions of features in large language models** [link]
Paulo, G., Mallen, A., Juang, C. and Belrose, N., 2024. arXiv preprint arXiv:2410.13928.

77. **Sparse Autoencoders Can Interpret Randomly Initialized Transformers** [link]
Heap, T., Lawson, T., Farnik, L. and Aitchison, L., 2025. arXiv preprint arXiv:2501.17727.

78. **Residual Stream Analysis with Multi-Layer SAEs** [link]
Lawson, T., Farnik, L., Houghton, C. and Aitchison, L., 2024. arXiv preprint arXiv:2409.04185.

79. **Transcoders Beat Sparse Autoencoders for Interpretability** [link]
Paulo, G., Shabalin, S. and Belrose, N., 2025. arXiv preprint arXiv:2501.18823.

80. **Features that Make a Difference: Leveraging Gradients for Improved Dictionary Learning** [link]
Olmo, J., Wilson, J., Forsey, M., Hepner, B., Howe, T.V. and Wingate, D., 2024. arXiv preprint arXiv:2411.10397.

81. **Efficient dictionary learning with switch sparse autoencoders** [link]
Mudide, A., Engels, J., Michaud, E.J., Tegmark, M. and de Witt, C.S., 2024. arXiv preprint arXiv:2410.08201.

82. **Identifying functionally important features with end-to-end sparse dictionary learning** [PDF]
Braun, D., Taylor, J., Goldowsky-Dill, N. and Sharkey, L., 2025. Advances in Neural Information Processing Systems, Vol 37, pp. 107286--107325.

83. **Improving Dictionary Learning with Gated Sparse Autoencoders** [link]
Rajamanoharan, S., Conmy, A., Smith, L., Lieberum, T., Varma, V., Kramar, J., Shah, R. and Nanda, N., 2024. arXiv preprint arXiv:2404.16014.

84. **Jacobian Sparse Autoencoders: Sparsify Computations, Not Just Activations** [link]
Farnik, L., Lawson, T., Houghton, C. and Aitchison, L., 2025. arXiv preprint arXiv:2502.18147.

85. **Towards principled evaluations of sparse autoencoders for interpretability and control** [link]
Makelov, A., Lange, G. and Nanda, N., 2024. arXiv preprint arXiv:2405.08366.

86. **Ravel: Evaluating interpretability methods on disentangling language model representations** [link]
Huang, J., Wu, Z., Potts, C., Geva, M. and Geiger, A., 2024. arXiv preprint arXiv:2402.17700.

87. **Measuring progress in dictionary learning for language model interpretability with board game models** [PDF]
Karvonen, A., Wright, B., Rager, C., Angell, R., Brinkmann, J., Smith, L., Mayrink Verdun, C., Bau, D. and Marks, S., 2025. Advances in Neural Information Processing Systems, Vol 37, pp. 83091--83118.

88. **Evaluating open-source sparse autoencoders on disentangling factual knowledge in gpt-2 small** [link]
Chaudhary, M. and Geiger, A., 2024. arXiv preprint arXiv:2409.04478.

89. **SAEBench: A comprehensive benchmark for sparse autoencoders, December 2024** [link]
Karvonen, A., Rager, C., Lin, J., Tigges, C., Bloom, J., Chanin, D., Lau, Y., Farrell, E., Conmy, A., Mc-Dougall, C. and others,. URL https://www. neuronpedia. org/sae-bench/info.

90. **Evaluating Sparse Autoencoders on Targeted Concept Erasure Tasks** [link]
Karvonen, A., Rager, C., Marks, S. and Nanda, N., 2024. arXiv preprint arXiv:2411.18895.

91. **The local interaction basis: Identifying computationally-relevant and sparsely interacting features in neural networks** [link]
Bushnaq, L., Heimersheim, S., Goldowsky-Dill, N., Braun, D., Mendel, J., Hanni, K., Griffin, A., Stohler, J., Wache, M. and Hobbhahn, M., 2024. arXiv preprint arXiv:2405.10928.

92. **Using degeneracy in the loss landscape for mechanistic interpretability** [link]
Bushnaq, L., Mendel, J., Heimersheim, S., Braun, D., Goldowsky-Dill, N., Hanni, K., Wu, C. and Hobbhahn, M., 2024. arXiv preprint arXiv:2405.10927.

93. **Interpretability in Parameter Space: Minimizing Mechanistic Description Length with Attribution-based Parameter Decomposition** [link]
Braun, D., Bushnaq, L., Heimersheim, S., Mendel, J. and Sharkey, L., 2025. arXiv preprint arXiv:2501.14926.

94. **Progress measures for grokking via mechanistic interpretability** [link]
Nanda, N., Chan, L., Lieberum, T., Smith, J. and Steinhardt, J., 2023. arXiv preprint arXiv:2301.05217.

95. **Investigating gender bias in language models using causal mediation analysis** [PDF]
Vig, J., Gehrmann, S., Belinkov, Y., Qian, S., Nevo, D., Singer, Y. and Shieber, S., 2020. Advances in neural information processing systems, Vol 33, pp. 12388--12401.

96. **Towards best practices of activation patching in language models: Metrics and methods** [link]
Zhang, F. and Nanda, N., 2023. arXiv preprint arXiv:2309.16042.

97. **How to use and interpret activation patching** [link]
Heimersheim, S. and Nanda, N., 2024. arXiv preprint arXiv:2404.15255.

98. **Localizing model behavior with path patching** [link]
Goldowsky-Dill, N., MacLeod, C., Sato, L. and Arora, A., 2023. arXiv preprint arXiv:2304.05969.

99. **Finding alignments between interpretable causal variables and distributed neural representations** [PDF]
Geiger, A., Wu, Z., Potts, C., Icard, T. and Goodman, N., 2024. Causal Learning and Reasoning, pp. 160--187.

100. **Interpretability at scale: Identifying causal mechanisms in alpaca** [PDF]
Wu, Z., Geiger, A., Icard, T., Potts, C. and Goodman, N., 2023. Advances in neural information processing systems, Vol 36, pp. 78205--78226.

101. **Attribution Patching: Activation Patching At Industrial Scale** [link]
Nanda, N., 2023.

102. **Attribution Patching Outperforms Automated Circuit Discovery** [link]
Syed, A., Rager, C. and Conmy, A., 2023. arXiv preprint arXiv:2310.10348.

103. **AtP*: An efficient and scalable method for localizing LLM behaviour to components** [link]
Kramár, J., Lieberum, T., Shah, R. and Nanda, N., 2024. arXiv preprint arXiv:2403.00745.

104. **Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms** [link]
Hanna, M., Pezzelle, S. and Belinkov, Y., 2024. arXiv preprint arXiv:2403.17806.

105. **EAP-GP: Mitigating Saturation Effect in Gradient-based Automated Circuit Identification** [link]
Zhang, L., Dong, W., Zhang, Z., Yang, S., Hu, L., Liu, N., Zhou, P. and Wang, D., 2025. arXiv preprint arXiv:2502.06852.

106. **Automatic discovery of visual circuits** [link]

Rajaram, A., Chowdhury, N., Torralba, A., Andreas, J. and Schwettmann, S., 2024. arXiv preprint arXiv:2404.14349.

107. Position-aware Automatic Circuit Discovery [link]
Haklay, T., Orgad, H., Bau, D., Mueller, A. and Belinkov, Y., 2025. arXiv preprint arXiv:2502.04577.

108. Low-complexity probing via finding subnetworks [link]
Cao, S., Sanh, V. and Rush, A.M., 2021. arXiv preprint arXiv:2104.03514.

109. Discovering variable binding circuitry with desiderata [link]
Davies, X., Nadeau, M., Prakash, N., Shaham, T.R. and Bau, D., 2023. arXiv preprint arXiv:2307.03637.

110. Finding transformer circuits with edge pruning [PDF]
Bhaskar, A., Wettig, A., Friedman, D. and Chen, D., 2025. Advances in Neural Information Processing Systems, Vol 37, pp. 18506--18534.

111. Uncovering intermediate variables in transformers using circuit probing [link]
Lepori, M.A., Serre, T. and Pavlick, E., 2023. arXiv preprint arXiv:2311.04354.

112. Sparse autoencoders enable scalable and reliable circuit identification in language models [link]
O'Neill, C. and Bui, T., 2024. arXiv preprint arXiv:2405.12522.

113. Information flow routes: Automatically interpreting language models at scale [link]
Ferrando, J. and Voita, E., 2024. arXiv preprint arXiv:2403.00824.

114. VISIT: Visualizing and interpreting the semantic information flow of transformers [link]
Katz, S. and Belinkov, Y., 2023. arXiv preprint arXiv:2305.13417.

115. Dictionary Learning Improves Patch-Free Circuit Discovery in Mechanistic Interpretability: A Case Study on Othello-GPT [link]
He, Z., Ge, X., Tang, Q., Sun, T., Cheng, Q. and Qiu, X., 2024. arXiv preprint arXiv:2402.12201.

116. Attention Output SAEs Improve Circuit Analysis [link]
Kissane, C., Krzyzanowski, R., Conmy, A. and Nanda, N., 2024.

117. Causal abstractions of neural networks [PDF]
Geiger, A., Lu, H., Icard, T. and Potts, C., 2021. Advances in Neural Information Processing Systems, Vol 34, pp. 9574--9586.

118. Causal abstraction: A theoretical foundation for mechanistic interpretability [link]
Geiger, A., Ibeling, D., Zur, A., Chaudhary, M., Chauhan, S., Huang, J., Arora, A., Wu, Z., Goodman, N., Potts, C. and others,, 2023. arXiv preprint arXiv:2301.04709.

119. Causal proxy models for concept-based model explanations [PDF]
Wu, Z., D'Oosterlinck, K., Geiger, A., Zur, A. and Potts, C., 2023. International conference on machine learning, pp. 37313--37334.

120. Decomposing and editing predictions by modeling model computation [link]
Shah, H., Ilyas, A. and Madry, A., 2024. arXiv preprint arXiv:2404.11534.

121. Causal scrubbing, a method for rigorously testing interpretability hypotheses [link]
Chan, L., Garriga-Alonso, A., Goldwosky-Dill, N., Greenblatt, R., Nitishinskaya, J., Radhakrishnan, A., Shlegeris, B. and Thomas, N., 2022. AI Alignment Forum.

122. Hypothesis testing the circuit hypothesis in LLMs [PDF]
Shi, C., Beltran Velez, N., Nazaret, A., Zheng, C., Garriga-Alonso, A., Jesson, A., Makar, M. and Blei, D., 2025. Advances in Neural Information Processing Systems, Vol 37, pp. 94539--94567.

123. The clock and the pizza: Two stories in mechanistic explanation of neural networks [PDF]
Zhong, Z., Liu, Z., Tegmark, M. and Andreas, J., 2023. Advances in neural information processing systems, Vol 36, pp. 27223--27250.

124. A toy model of universality: Reverse engineering how networks learn group operations [PDF]
Chughtai, B., Chan, L. and Nanda, N., 2023. International Conference on Machine Learning, pp. 6243--6267.

125. Grokking group multiplication with cosets [link]
Stander, D., Yu, Q., Fan, H. and Biderman, S., 2023. arXiv preprint arXiv:2312.06581.

126. Fourier circuits in neural networks and transformers: A case study of modular arithmetic with multiple inputs [link]
Li, C., Liang, Y., Shi, Z., Song, Z. and Zhou, T., 2024. arXiv preprint arXiv:2402.09469.

127. A circuit for Python docstrings in a 4-layer attention-only transformer [link]
Heimersheim, S. and Janiak, J., 2023. Alignment Forum.

128. **How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model** [PDF]
Hanna, M., Liu, O. and Variengien, A., 2023. Advances in Neural Information Processing Systems, Vol 36, pp. 76033--76060.

129. **Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla** [link]
Lieberum, T., Rahtz, M., Kramar, J., Nanda, N., Irving, G., Shah, R. and Mikulik, V., 2023. arXiv preprint arXiv:2307.09458.

130. **Identifying a preliminary circuit for predicting gendered pronouns in gpt-2 small** [link]
Mathwin, C., Corlouer, G., Kran, E., Barez, F. and Nanda, N., 2023. URL: https://itch.io/jam/mechint/rate/1889871.

131. **Identifying and adapting transformer-components responsible for gender bias in an English language model** [link]
Chintam, A., Beloch, R., Zuidema, W., Hanna, M. and Van Der Wal, O., 2023. arXiv preprint arXiv:2310.12611.

132. **Scaling Sparse Feature Circuits For Studying In-Context Learning** [link]
Kharlapenko, D., Shabalin, S., Barez, F., Nanda, N. and Conmy, A., 2025.

133. **Circuit component reuse across tasks in transformer language models** [link]
Merullo, J., Eickhoff, C. and Pavlick, E., 2023. arXiv preprint arXiv:2310.08744.

134. **Circuit Compositions: Exploring Modular Structures in Transformer-Based Language Models** [link]
Mondorf, P., Wold, S. and Plank, B., 2024. arXiv preprint arXiv:2410.01434.

135. **LLM circuit analyses are consistent across training and scale** [link]
Tigges, C., Hanna, M., Yu, Q. and Biderman, S., 2024. arXiv preprint arXiv:2407.10827.

136. **Gemma 2: Improving open language models at a practical size** [PDF]
Team, G., Riviere, M., Pathak, S., Sessa, P.G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Rame, A. and others,, 2024. arXiv preprint arXiv:2408.00118.

137. **Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2** [PDF]
Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A., Shah, R. and Nanda, N., 2024.

138. **Dictionary Learning Optimization Techniques** [link]
Conerly, T., Cunningham, H., Templeton, A., Lindsey, J., Hosmer, B. and Jermyn, A., 2024.

139. **Feature Manifold Toy Model** [link]
Olah, C. and Batson, J., 2023.

140. **Not all language model features are linear** [PDF]
Engels, J., Michaud, E.J., Liao, I., Gurnee, W. and Tegmark, M., 2024. arXiv preprint arXiv:2405.14860.

141. **What is a Linear Representation? What is a Multidimensional Feature?** [link]
Olah, C., 2024.

142. **Curve Detector Manifolds in InceptionV1** [link]
Gorton, O., 2024.

143. **Residual stream norms grow exponentially over the forward pass** [link]
Heimersheim, S. and Turner, A., 2023.