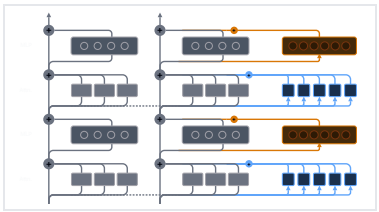
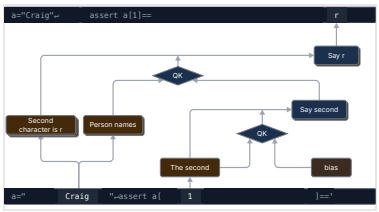
← OpenMOSS Interpretability Posts

Bridging the Attention Gap: Complete Replacement Models for Complete Circuit Tracing

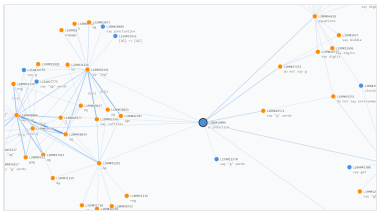
We introduce Complete Replacement Models (CRMs), which combine transcoders for MLPs with Low-Rank Sparse Attention (Lorsa) modules that decompose attention into interpretable features. This enables us to build attribution graphs that reveal interpretable sparse computational paths in both attention and MLP computation. Architectural completeness also unlocks accurate and efficient global weight analysis, letting us construct global circuits that reveal input-independent versions of canonical circuits like induction at the feature level.



The Complete Replacement Model



Attribution Graphs



Global Circuits

OpenMOSS Team, Shanghai Innovation Institute & Fudan University

Wentao Shu [†] Xuyang Ge [†,‡] Guancheng Zhou [†] Junxuan Wang Rui Lin Zhaoxuan Song Jiaxing Wu Zhengfu He Xipeng Qiu [*]

[†] Core contributor. [‡] Core infrastructure contributor. [*] Correspondence to xpqiu@fudan.edu.cn

Introduction

Understanding the underlying mechanisms of large language models is critical to mechanistic interpretability research. One of the main challenges is that most neurons and attention heads cannot be independently understood. The field has identified this problem as *superposition*[1,2]. This hypothesis suggests that there might exist *true features* that the model is using to represent information, but these features are often smeared across multiple neurons or attention heads. An active direction in addressing this problem is 1) to find these true features and 2) understand how they interact with each other via model parameters (i.e. circuits).

Sparse dictionary learning methods, particularly sparse autoencoders (SAEs) [3,4], have emerged as a promising approach to extract monosemantic features from language models. A variant called transcoders[5,6,7] makes circuit identification easier by approximating MLP outputs with sparse features. Recent work has used transcoders to build attribution graphs (visual representations of how features influence model outputs) for specific prompts[6,8].

However, these approaches share a critical limitation: missing attention circuits. Existing methods decompose MLP computation but treat attention patterns as given, leaving attention superposition unresolved. We address this by introducing complete replacement models (CRMs), which combine transcoders for MLPs with Low-Rank Sparse Attention (Lorsa)[9] modules for attention layers. This gives us interpretable features for every computational block[1].

Complete replacement models. In §3 From Transcoder-Only to Complete Replacement Models, we introduce CRMs, which replace each attention and MLP layer with a sparse replacement layer trained to approximate the original layer's output. Each Lorsa feature has its own QK circuit (determining where to attend) and rank-1 OV circuit (determining what to read and write), making attention computation fully decomposable. By freezing attention patterns and

layernorm denominators during attribution, we convert the model into a linear computation graph where all feature interactions flow through interpretable virtual weights. This architectural completeness is key to both efficient attribution and global weight analysis.

Why attention matters. Training SAEs on attention outputs can extract monosemantic features[10,11], but these features may still be collectively computed by multiple heads. This makes it difficult to trace how attention patterns form, a process called QK tracing[2]. Lorsa solves this by decomposing attention into sparse, independently interpretable features, each with its own QK and OV circuits.

Complete attribution graphs. In §4 Building Complete Attribution Graphs, we show how CRMs convert all attention-mediated paths into combinations of interpretable feature interactions. This eliminates the exponential growth of possible paths between features and enables complete attribution graphs that trace both MLP and attention circuits.

Revisiting the biology of language models. In §5 Revisiting the Biology of Language Models, we apply CRMs to study how Qwen3-1.7B implements specific behaviors. In string indexing tasks, we trace how the model selectively retrieves characters at different positions through shared attention feature families. For induction, we identify distinct subcircuits operating at different depths: lower layers perform relation-keyed binding while higher layers execute generic retrieval. These case studies reveal attention-mediated feature interactions that were previously hidden in transcoder-only analyses.

Global circuits. In §6 Global Circuits, we show how architectural completeness unlocks efficient global weight analysis. Because all feature interactions flow through static virtual weights rather than input-dependent attention patterns, we can compute expected attributions over distributions and construct global circuit atlases. These reveal canonical computational patterns such as induction circuits at the feature level for the first time.

Evaluation. In §7 Evaluation, we evaluate our CRMs trained on Qwen3-1.7B in

three dimensions: interpretability (feature quality and reconstruction fidelity), sufficiency (how well attribution graphs capture model behavior), and mechanistic faithfulness (whether interventions match predictions). While Lorsa introduces additional reconstruction error, the resulting attribution graphs achieve comparable faithfulness to transcoder-only models while providing complete circuit coverage.

Replacement Layers

Replacement layers are sparse, interpretable modules trained to approximate the computation of individual attention or MLP modules in the underlying model. Each replacement layer learns a dictionary of features and uses sparse coding to reconstruct the original layer's output: sparse codes are computed from the layer inputs, then decoded through the dictionary to approximate the original output. Conceptually, the original layer is "sandwiched" between the encoder (which computes sparse features from inputs) and the decoder (which reconstructs outputs from these features), allowing us to decompose the computation into interpretable, monosemantic components. By combining replacement layers for all underlying layers, we obtain a complete replacement model that rewrites the model's computation in a more interpretable form.

Transcoders

Transcoders[6,7,8] are replacement layers that approximate the computation of an MLP layer. In this work, we use the per-layer version instead of cross-layer

transcoders[8] for both conceptual and engineering simplicity.[3] A Per-layer Transcoder (PLT) has the same architecture as Sparse Autoencoders (SAEs) [3,4], but learns to approximate downstream activations instead of its original input.

Concretely, given an MLP input activation, denoted as $\mathbf{x} \in \mathbb{R}^d$, the transcoder computes the feature activation as:

$$\mathbf{a} = \mathrm{TopK}(\mathbf{W}_{\text{enc}}\mathbf{x}),$$

where $\mathbf{W}_{\text{enc}} \in \mathbb{R}^{F \times d}$ is the encoder weight matrix. The input is encoded into a feature activation space with $F \gg d$ dimensions, followed by a Top-K sparsity constraint to select the K strongest features and set others to zero[12]. Alternative sparsity constraints such as BatchTopK[13] and JumpReLU[14] should have similar effect[15].

The sparse feature activations are then decoded to the output space as:

$$\mathbf{y}' = \mathbf{W}_{\text{dec}}\mathbf{a} + \mathbf{b}_{\text{dec}},$$

where $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times F}$ is the decoder weight matrix, and $\mathbf{b}_{\text{dec}} \in \mathbb{R}^d$ is the decoder bias.

The transcoder is trained to simply minimize the mean squared error between the decoded output and the original output $\mathbf{y} \in \mathbb{R}^d$:

$$\mathcal{L} = \mathbb{E}_{X \sim \mathcal{D}} \left[\|\mathbf{y} - \mathbf{y}'\|_2^2 \right]$$

Low-Rank Sparse Attention (Lorsa)

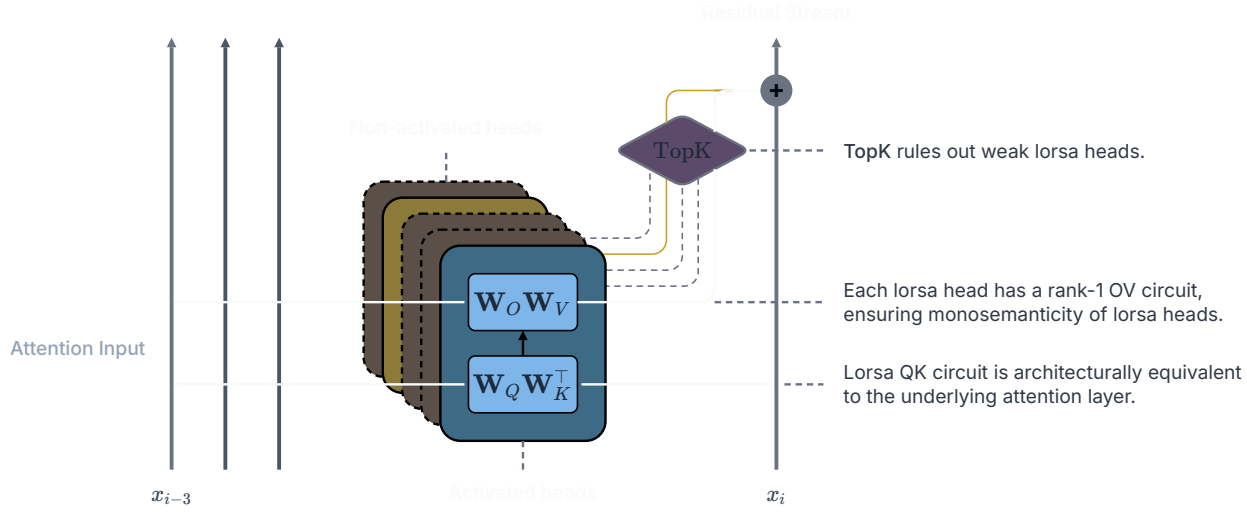
Lorsa[9] is designed as a sparse replacement model for attention layers. As the attention counterpart of transcoders, it is also trained to find a sparse representation of the attention layer output, while also accounting for how these sparse features are computed from attention input.

Attentional Features and Attention Superposition

These two terms were first introduced in Jermyn et al. (2023)[16] and Jermyn et al. (2024)[17]. We further extended investigation of this topic in our paper *Towards Understanding the Nature of Attention with Low-Rank Sparse Decomposition*[9]. These works assumed and provided evidence that we can extract monosemantic attentional features from superposition in the attention layer.

Our current understanding of attentional features is that they each have:

- **A QK circuit**[4] reflecting how they choose to attend from one token to others;
- **A rank-1 OV circuit**[5] reading a specific set of features from key-side residual streams and writing to the query side.
- **Overcompleteness and Sparsity:** The underlying attention layer learned to represent *orders of magnitude more attentional features* than it has heads, if and only if they are *sparsely activated* at any given position. This is analogous to the definition of superposition in activation spaces[2].



Architecture of Low-Rank Sparse Attention (Lorsa)

Following the definition of attentional features, we design Lorsa to have an overcomplete set of sparsely activating attention heads, each of which has a rank-1 OV circuit and a QK circuit with the same dimension as the underlying attention layer. In practice, we adopt weight sharing across groups of heads (see Lorsa Feature Families) to maintain scalability. However, for conceptual

simplicity, we illustrate the architecture with independent QK circuits.

Lorsa QK Circuits

Let $\mathbf{X} \in \mathbb{R}^{l \times d}$ denote the input of the attention layer, where l is the sequence length and d is the dimension of the input. The QK circuit of a Lorsa head with head dimension d_h is computed as follows (we omit head index for simplicity):

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q; \mathbf{K} = \mathbf{X}\mathbf{W}_K,$$

where $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d \times d_h}$ are the QK weight matrices. Attention pattern is then computed as:

$$\mathbf{P} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^T),$$

where $\mathbf{P} \in \mathbb{R}^{l \times l}$ is the attention pattern. **Lorsa QK circuit is architecturally equivalent to the underlying attention layer.**

Since attention architecture of the underlying model varies across models, we adopt an architectural compatibility principle to apply causal mask, attention scale[18], rotary embedding[19], grouped query attention[20] and QK-layernorm [21,22].

Lorsa OV Circuits

The OV circuit of a Lorsa head is computed as:

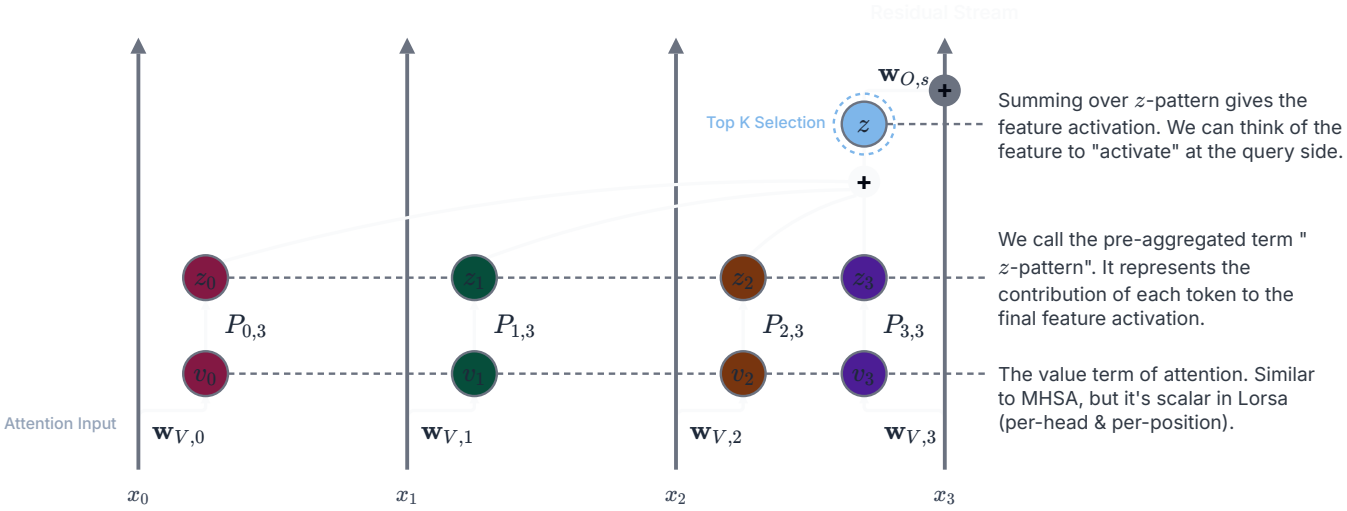
$$\mathbf{v} = \mathbf{X}\mathbf{w}_V; \mathbf{z} = \mathbf{P}\mathbf{v},$$

where $\mathbf{w}_V \in \mathbb{R}^d$ is the value weight projection. This maps the input to $\mathbf{v}$, which is **a scalar at each position**. Then the attention pattern *moves* these values to the query side as feature activation $\mathbf{z} \in \mathbb{R}^l$ at each position. Similar to transcoders, we apply a Top-K sparsity constraint (along the head dimension) to select the K strongest features and set others to zero[12]. For each position, we have a set of selected features denoted as $\mathbf{S}_c, c = 1, 2, \cdots, l$.

The output of this head at a given position c is simply:

$$\mathbf{y}'_c = \sum_{s \in \mathbf{S}_c} \mathbf{z}_s \mathbf{w}_{O,s},$$

where $\mathbf{w}_{O,s} \in \mathbb{R}^d$ is the output weight of feature s. Training a Lorsa layer is similarly minimizing the MSE loss between Lorsa output $\mathbf{y}'_c$ and attention output $\mathbf{y}_c$ at each position.



Notion of feature activation in Lorsa.

Visualizing Features

Our understanding of sparse features are based on their mathematical and statistical properties, typically shown by their direct logit attribution (DLA) and their top activation contexts. Below we show an example Transcoder feature from layer 18 of Qwen3-1.7B, which activates on tokens starting with "g" or similar patterns in the latinization of other languages:

G-initial	qwen3-1.7b-plt-8x-topk128-layer18 #105
POSITIVE	' spy _rid _rise ↵_____ ↵ ven rails imiento {o sword
NEGATIVE	ghan gable gers ame gom getFile gren aker gies gram
ギ 13.75	リオは、主に次の2つの要素で構成されています。↵↵1.ギャラリーからの CloudPassage の追加 ↵2. Azure ADシングル
→G 13.63	//-- ↵ //→Greenplum Database↵//→Copyright (C)2011 EMC Corp.↵//↵//
egl 13.06	sondern dass sich der Artist ständig neuentdeckt, was Marilyn Manson hier wirklich grandios geglückt ist. Zur Platte selbst möchte ich sagen, dass ich der Aussage man cher hier
.get 12.63	this.corkedRequestsFree = new CorkedRequest(this);↵ }↵ W ritableState.prototype.getBuffer = function getBuffer() {↵ var current = this.bufferedRequest;↵ var out =

Lorsa features can be inspected in similar way. In the visualization of Lorsa features, we further provide the z-pattern, which can be observed by hovering activated tokens. Below is an example Lorsa feature at layer 22 of Qwen3-1.7B, which attends to "g" tokens and predicting next tokens to end with "g":

say "ending with G" qwen3-1.7b-lorsa-8x-topk128-layer22 #14037

Hover on top activations to see z-pattern

POSITIVE	_PG FG CG _SG PG _CG RG SG _MG MG
NEGATIVE	_given given Given 给 _Given 給 给你 _dado _giving _give
0 **40.00**	\n" \↵ "h -104\n" \↵ "g -103\n" \↵ "f -102\n" \↵ "d -
9 **33.50**	, et al.,↵1983), concept maps (Novak & Gowin,1984), and thematic organizers (Alvarez,↵1980,198
《 **33.00**	但要拍摄也更困难。摄影/秦红宇↵晋代的郭缘生在《述征记》中说:“太行山首始于河内,北至幽州
3 **33.00**	↵const double C4 =261.63;↵const double G4 =392.00;↵const double A4 =440.00;↵const
《 **33.00**	音乐，制新曲曰《凉州》，开元中列上献之。”郭茂倩《乐府诗集》载：“《凉州》，宫调曲。开元中，西凉

Training Setup

We train replacement layers on Qwen3-1.7B with $K = 64, 128, 256$ and dictionary sizes of 16,384 (8×) and 65,536 (32×). Detailed training configurations, including dataset, learning rate, and initialization, are provided in the appendices.

From Transcoder-Only to Complete Replacement Models

In this section, we start from a specific attribution graph obtained by a transcoder-only replacement model, and show four fundamental problems of it. Then we introduce how a complete replacement model (i.e. a model with both transcoder and lorsa replacement layers) can solve these problems.

Transcoder-Only Replacement Models

The idea of building attribution graphs with MLP replacement layers was first proposed in our previous work *Automatically Identifying Local and Global Circuits with Linear Computation Graphs*[6], but not investigated deeply enough.

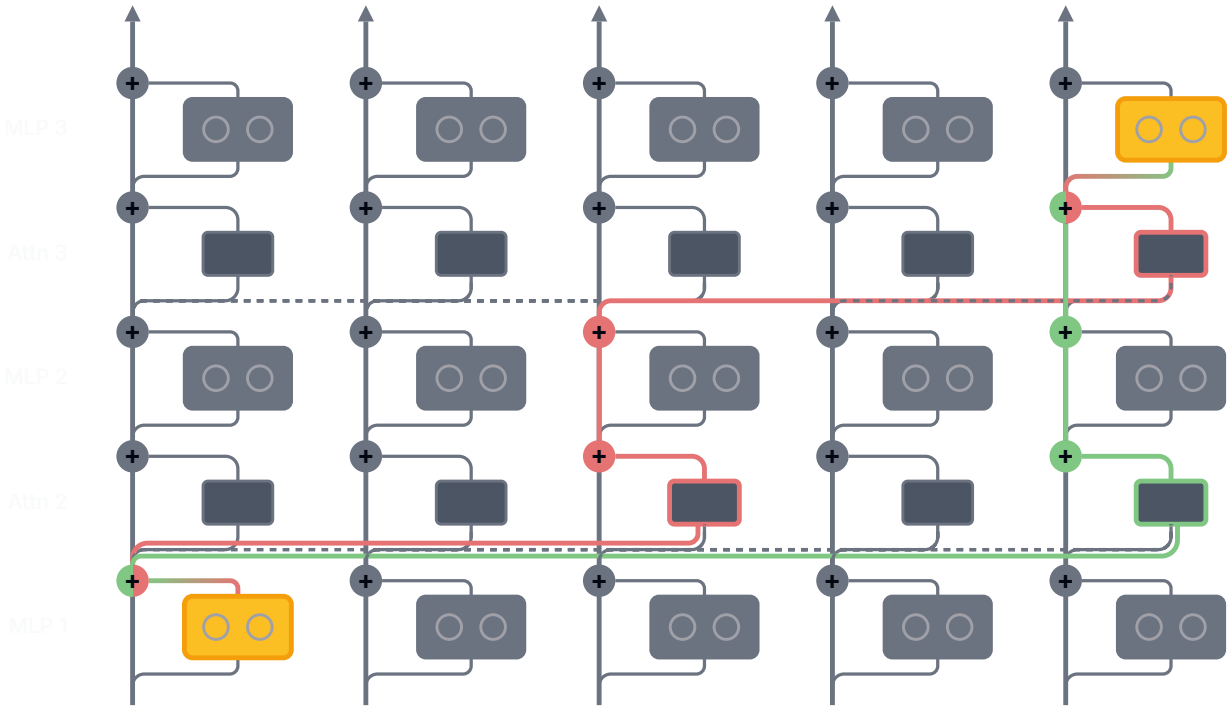
Transcoder-only replacement models use transcoders for direct feature-feature interactions computed by MLP blocks, with attention modules left as is. By "freezing" attention patterns and layernorm denominators, the computation of the underlying model can be described as a linear computation graph given a specific input[23]. This conditional linearity makes attribution well-defined as a graph problem, where each feature's contribution can be traced along direct, composable pathways through the network. We can further prune the graph to isolate a subgraph of interest for easier digestion.

This methodology was greatly improved by Ameisen et al.[8] in many aspects including attribution and pruning algorithm, visualization interface, evaluation and global weight analysis. Importantly, they proposed to replace all MLP modules with a Cross-Layer Transcoder (CLT) rather than a set of Per-Layer Transcoders (PLTs) to combine features that might exist across multiple MLP layers[24] for simpler attribution graphs.

The Problem of Missing Attention Circuits

This attribution gives us a partial picture of the underlying model behavior. However, it fails to explain which attention head(s) are responsible for edges between two features from different positions. Concretely, this breaks down to four fundamental problems:

1. **Attention Superposition:** Despite substantial existing work on identifying independently interpretable attention heads[25,26,27,28], most heads in a transformer language model cannot be independently understood[29].
2. **Exponential Growth of Attention-Mediated Paths:** Consider two transcoder features separated by P tokens and L layers. The edge connecting these two features has $\sum_{k=1}^{L} \binom{L}{k} \binom{P+k-1}{k-1}$ possible paths (under the assumption that each layer has only one attention head).



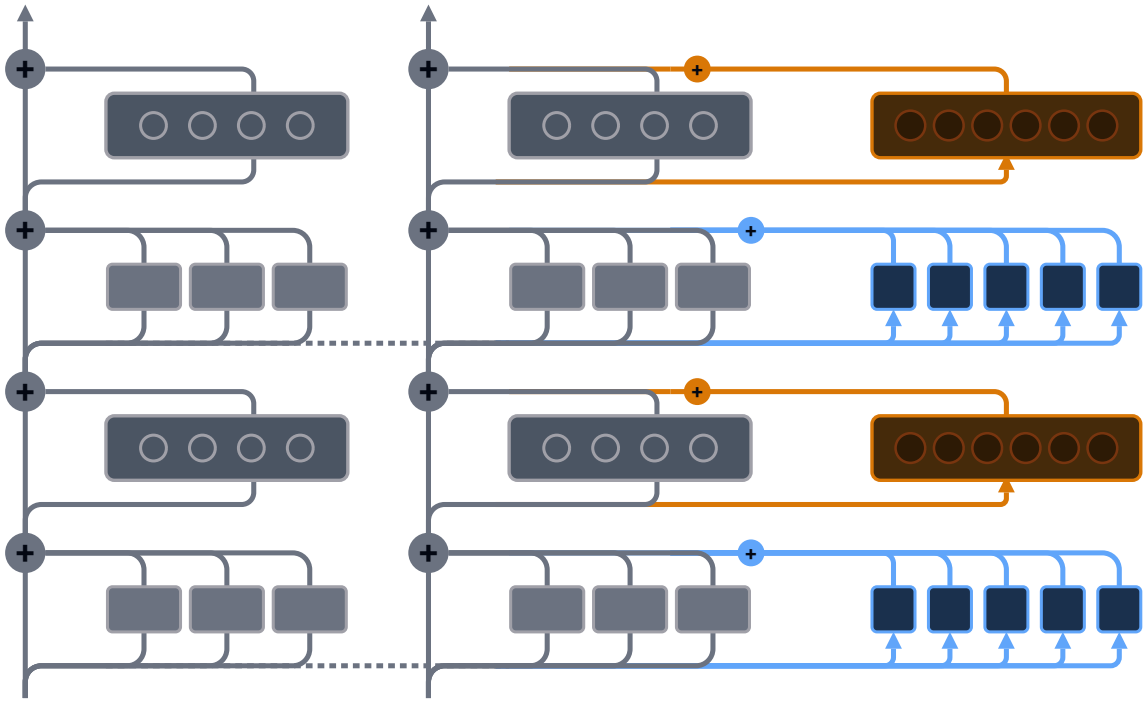
*A direct edge in transcoder-only replacement model can have **possible attention-mediated paths exponentially growing with the number of layers**. For the example shown with two features separated by 4 tokens and 2 layers, there are 7 possible paths in total, with 5 of them passing **two attention heads** (red line), and 2 of them going through **one attention head and one residual connection** (green line).*

3. **Head Loading[30] in One Attention Layer:** Even in cases where a single path

almost exclusively contributes to an edge, we still need to quantify the amount that each attention head is responsible for mediating its corresponding attention layer.

4. **Attention-Mediated Interaction in one Residual Stream:** Since attention layers can attend from one token to itself, attribution inside a residual stream should also take into account the attention-mediated interaction. We show how this affects global weight analysis in §6 Global Circuits.

Complete (Local) Replacement Models



A complete replacement model with transcoder and lorsa replacement layers.

With Lorsas, we now have a sparse replacement layer for each original computational block. Combining these replacement layers yields a **Complete Replacement Model (CRM)**, which adopts a distinct yet related parameter set to the underlying model. The CRM and underlying model share the same overall architecture, but the CRM's neurons (i.e., transcoder features) and attention heads (i.e., lorsa heads) show substantially greater monosemanticity.

The forward pass of any prompt is taken over by the CRM and computes as

follows:

1. Tokens pass through the same embedding layer, which is by itself an interpretable transformation of the input tokens.
2. Upon reaching an attention layer, we use its corresponding Lorsa module to compute its output at each token position. The same applies to transcoders for MLP layers.
3. Lorsa attention patterns and layernorm denominators are *frozen* after they are computed. This means we treat them as a constant in the replacement model, instead of being computed from their inputs.[6]
4. Following Marks et al. and Ameisen et al., we add an error term to the output of each replacement layer's output to exactly make up the reconstruction error so that later layers will receive the same input as the original model.
5. Finally, we unembed the output to get the same final logits as the original model.

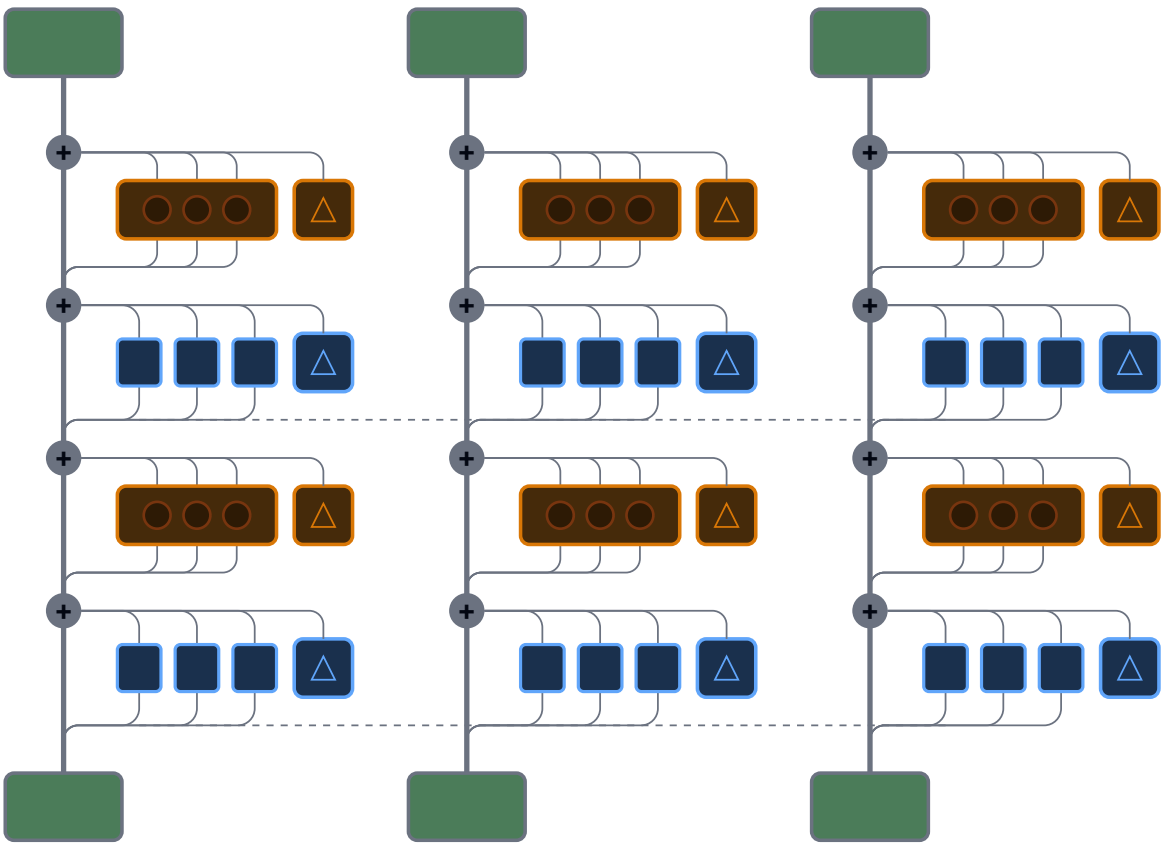
Compared to a transcoder-only replacement model, a complete replacement model offers us a more complete picture of the underlying model behavior. It also has many good properties for attribution, which we will discuss later on. However, these come at the cost of additional error terms from Lorsas. We will measure the impact of these error terms on attribution in §7 Evaluation.

Including attention replacement layers in a replacement model makes it *complete* in the sense that it is designed to attack superposition happening in both attention and MLP layers. However, it is still *local* because of 1) frozen lorsa attention patterns and layernorm denominators and 2) uninterpretable error terms. Both are dependent on specific model inputs but we choose not to explain them in this phase. We will later show how we further explain lorsa attention patterns in QK Tracing.

Building Complete Attribution Graphs

The algorithm we use for attribution is generally similar to that in transcoder-only models[6,7,8]. That is, for any two nodes in a local replacement model, the attribution between an upstream (source) node and a downstream (target) node is defined as $A_{s \to t} := a_s w_{s \to t}$ where a_s is the activation of the source node, and $w_{s \to t}$ is the sum over all possible paths from the source to the target.[7]

A good property in CRMs is that direct interaction between any two nodes is mediated by no more than one path.[8] So the term "edge" in graph contexts and "path" in Transformer contexts are equivalent. We will be using these terms interchangeably in the rest of this work.



Since the residual stream mediates all inter-node interactions, attribution

between any two nodes depends only on how the source node *writes to* the residual stream and how the target node *reads from* the residual stream. We abstract *generic encoder and decoder vectors* to unify the measuring of these read/write abilities of all node kinds. The generic encoder vector $\mathbf{w}_{\text{enc},t}$ is given by

$$\mathbf{w}_{\text{enc},t} = \begin{cases} \mathbf{W}_{\text{enc},t} & \text{if } t \text{ is a Transcoder feature} \\ \mathbf{w}_{\text{V},t} & \text{if } t \text{ is a Lorsa feature} \\ \mathbf{w}_{\text{unembed},t} & \text{if } t \text{ is a logit} \end{cases}.$$

Similarly, the generic decoder vector $\mathbf{w}_{\text{dec},s}$ for a source node s is given by

$$\mathbf{w}_{\text{dec},s} = \begin{cases} \mathbf{W}_{\text{dec},s} & \text{if } s \text{ is a Transcoder feature} \\ \mathbf{w}_{\text{O},s} & \text{if } s \text{ is a Lorsa feature} \\ \mathbf{w}_{\text{embed},s} & \text{if } s \text{ is a token embedding} \\ \epsilon_s & \text{if } s \text{ is an error node} \end{cases}.$$

Consider any source node s from position i and a target node t at position j s.t. $i \leq j$, the graph weight is given by

$$A_{s \to t} = a_s P^t_{i,j} \mathbf{w}^\top_{\text{dec},s} \mathbf{w}_{\text{enc},t} = a_s P^t_{i,j} \Omega_{s \to t},$$

where $P^t_{i,j}$ is the attention pattern between positions i and j computed by the target node t. Transcoder features and logits do not have attention pattern, in which case we set an identity attention pattern $P^t = I$ without losing generality. $\Omega_{s \to t} := \mathbf{w}^\top_{\text{dec},s} \mathbf{w}_{\text{enc},t}$ is defined as the residual-direct virtual weight[23] between two features.

For a given prompt, we first run a forward pass to get all necessary node activations and Lorsa attention patterns. Then we can compute the connections between nodes using source node activations, attention patterns, and virtual weights. This will give us a complete linear attribution graph for the given prompt.

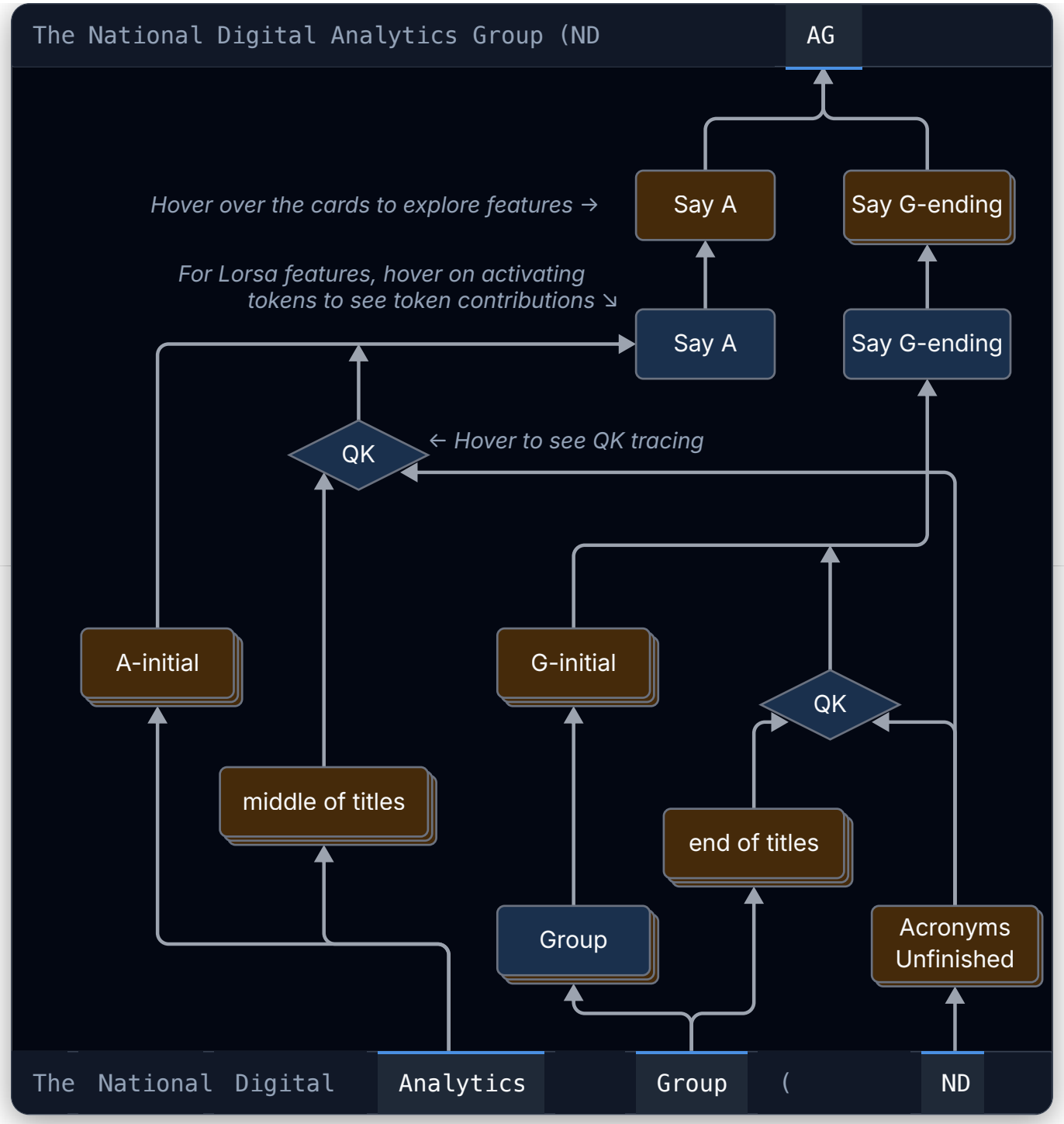
In practice, we apply a gradient-based implementation[6,8] on CRMs to compute

the attribution by stopping gradient propagation at the Lorsa attention patterns and layernorm denominators, where the computation is equivalent to the forward perspective we introduced above. We refer readers to §G Circuit Tracing for the full algorithmic details. The main reason to adopt this approach is efficiency and backward compatibility.

To ease comprehension, we prune the resulting attribution graph following the procedure of Ameisen et al.[8]. See §H Graph Pruning for the procedure.

An Example Complete Attribution Graph

After pruning and human distillation, we can describe mechanistic hypotheses about the model's behavior for any given prompt. We study a similar prompt to Ameisen et al. for more straightforward comparison. The model takes "The National Digital Analytics Group (ND" as input and is able to complete the acronym by predicting "AG" as the next token.



The overall structure of the attribution graph is similar to the established conclusions from transcoder-only models: The model learns a group of "First Letter" features at early layers of the model (e.g., `A-initial` and `G-initial` features in the graph above). A group of "First Letter Mover" features then moves them to the last token position to enhance probability of token "AG" being predicted.

Compared to transcoder-only models, a complete attribution graph identifies important attention heads responsible for the model's behavior (moving first letters in this example). We can then learn from this graph that the `Say A` Lorsa features are reading a set of `A-initial` features from "Analytics" position and writing to the residual stream to encourage "Saying A". This is the OV perspective of describing an attentional feature.

We also want to understand why this features attend to the specific residual stream position. By looking at features contributing to the attention pattern between the last token and "Analytics" position, we can see that the query side (last token) has a set of `Acronyms Unfinished` features searching for the next words in the acronym. The key side ("Analytics") has a group of `middle of titles` features indicating that this word is likely to be a middle word of a title. Similarly, the `Say G-ending` Lorsa attention pattern is mainly contributed by a group of `end of titles` features on the key side and `Acronyms Unfinished` features on the query side. We call this process QK tracing.

QK Tracing

The idea of QK tracing in the context of sparse feature circuits was first introduced in He et al.[10], and further illustrated in more detail in Ge et al.[6] and Kamath et al.[30]. We can leverage the bilinear nature of attention score computation. Since attention scores are computed as dot products between query and key vectors—which are linear transformations of the residual stream—we can decompose each attention score into a sum of interpretable terms.

Specifically, by expanding the residual stream at query and key positions as sums of feature activations, bias terms, and residual errors, the bilinear form naturally decomposes into feature-feature interactions (inner products between query-side and key-side features), bias interactions, and error terms. This allows us to explain why an attention head attends to particular positions in terms of which features on the query side interact with which features on the key side.

For a detailed mathematical treatment, we refer readers to Kamath et al..

However, attention scores are normalized by softmax to form a probability distribution, which means we cannot simply examine how attention scores form at one position in isolation. Suppressing other positions will affect the attention pattern at the target position. This same issue arises in logit attribution as well. We refer readers to §9.5 "Uninterested" Circuits for further discussions.

Navigating an Attribution Graph

Now we demonstrate how we visualize and navigate an attribution graph to form mechanistic hypotheses for the example above. The interactive visualization below shows a heavily pruned attribution graph. Nodes in the graph represent active features at different tokens and different layers with influence strength above a certain threshold. These include token embeddings, predicted logits, **Transcoder** features and **Lorsa** features. **Error** nodes are for uncaptured residuals in either Transcoders or Lorsas.

You can click on any node to inspect the detailed information of the feature, along with its incoming and outgoing connections to any other nodes. By navigating through the primary connections, we can trace how Lorsa OV and transcoder paths influence the model's behavior.

For any Lorsa feature, we can click on it and select the "QK Tracing" tab to see the top marginal and pairwise feature contributions to the attention scores between the target (query) position and the position with largest z-pattern contribution. These nodes are not displayed in the graph, but we can click on them in the QK tracing tab to see their detailed information.

sa (16.875) A22#14037@6

OV Inputs/Outputs QK Tra

INPUT NODES

M20	G- initial	+7.969 9.1
M18	G- initial	+2.938 5.25
M19	G-initial citations	+2.
M16	ig	+0.961 4.28
M17	给	+0.844 3.13
M19	say G words	−1.63 3.5

OUTPUT NODES

M22	say "two-letter abbreviation	
M27	say g endings	+4.6 12
Logit@6: AG (100.0%)		+3.98

NEGATIVE LOGITS		POSITIVE LOGITS	
_(	-26.750	_	57.750
g	-22.750	F	57.500
G	-21.625	C	57.500
丝	-21.375	_	56.500
_(	-21.250	P	56.000
絲	-19.500	_(	55.500
丝	-19.000	R	55.250
_(	-18.375	S	55.000
_(	-18.375	_	54.750
_(	-18.375	M	54.500

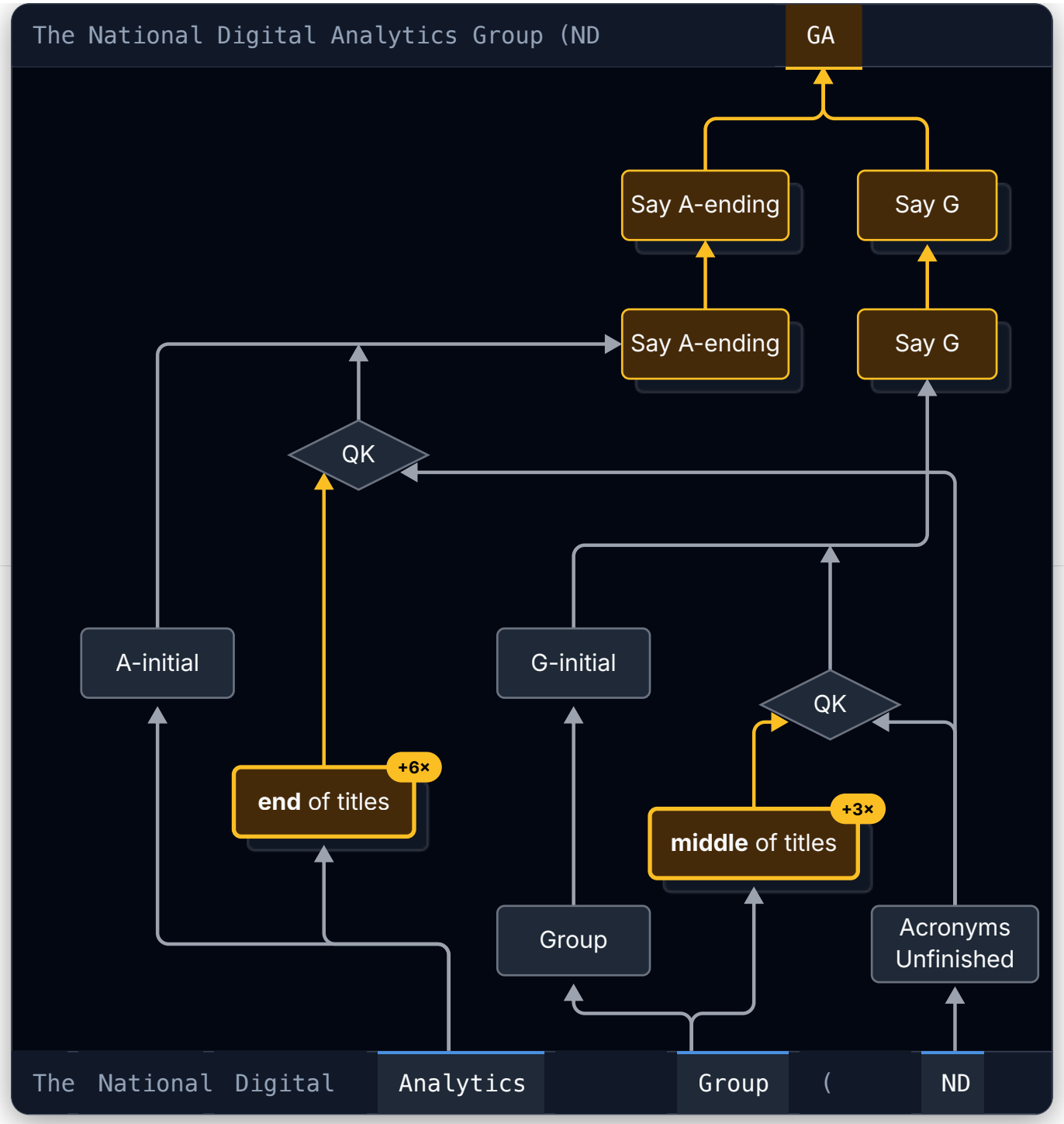
ACTIVATION TIMES 1.412%

TOP ACTIVATIONS Stacked

0 40.00	4\n" \↵ "g −103\n" \↵ "f −
9 33.50	maps (Novak & Gowin, 1984), and thematic organizers (Alv
《 33.00	红宇↵晋代的郭缘生在《述征记》中说:“

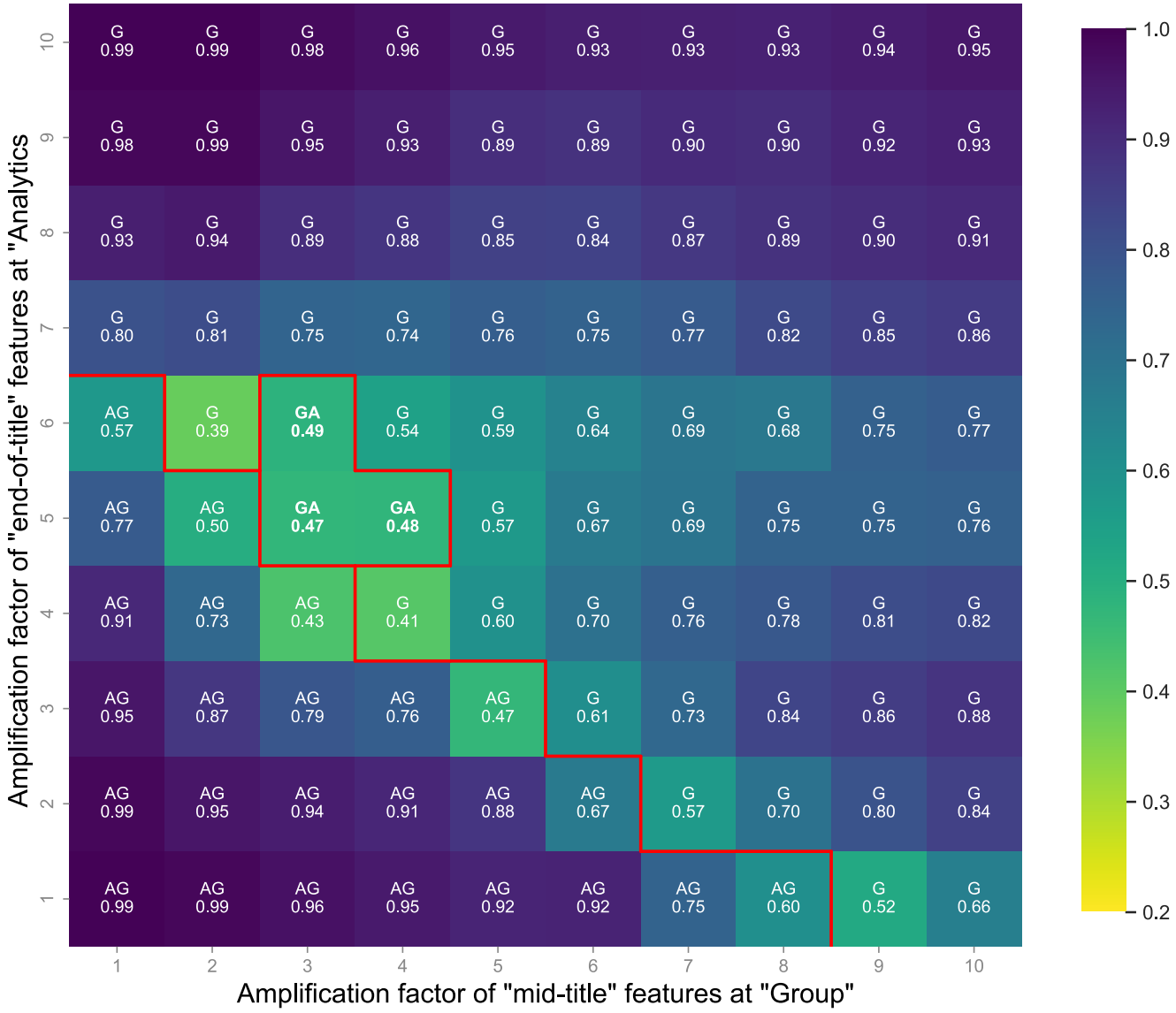
Causal Validation: Reversing Word Order

Feature steering offers a way to verify that the mechanisms captured by the attribution graph are causally real: if manipulating features produces changes consistent with the graph's predictions, the identified mechanism is likely genuine. The graph above suggests that the model's prediction of "AG" relies on "middle of titles" features activating on "Analytics" and "end of titles" features activating on "Group." To test this, we can swap the activation positions of these features and expect the model to output "GA" instead of "AG," reflecting the reversed positional signals.



Specifically, we swap the activations of the "middle of titles" and "end of titles" features between the "Analytics" and "Group" positions. To prevent downstream self-correction by the model, we further scale up the swapped activations — amplifying the "end of titles" features at "Analytics" and the "middle of titles" features at "Group" by multiplicative factors.

Next-Token Probabilities Under Title-Feature Swapping



Next-token prediction (top token and its probability) at the "ND" position under activation swapping, across varying amplification scales of swapped "end-of-title" features at "Analytics" (y-axis) and "middle-of-title" features at "Group" (x-axis).

Through activation swapping with varying amplification factors, we observe that the model's top prediction shifts from "AG" to "GA", with the logit of "G" significantly increasing at the position where "A" was previously predicted. This confirms that the model internally relies on these two groups of features to determine the ordering of abbreviation letters. The prevalence of "G" over "GA" at higher amplification factors likely reflects the dominance of the middle-of-title feature, and may be related to positional mechanisms such as RoPE that are not captured in the attribution graph.

Nevertheless, this steering experiment provides strong evidence that the

features and pathways identified in our attribution graph are not merely correlational, but represent genuinely causal mechanisms that determine model output. Future work will include comprehensive visualization of these steering interventions and their effects on the attribution graph structure.

From Attribution Graphs to Testable Hypotheses

Beyond explaining individual predictions, attribution graphs reveal systematic patterns in how models process information more generally. The graph above suggests the model uses a fuzzy position-indexing algorithm: features marking the start of acronyms activate at "ND", while the model searches for middle and end-of-title features to determine which letters to output next. This mechanism raises a testable hypothesis: if the model relies on approximate positional signals rather than precise counting, it should struggle with longer titles where positional ambiguity increases.

To test this, we generated synthetic acronym completion prompts with controlled title lengths. For five-word titles[9], the model achieves 60% accuracy (6/10 correct predictions). In contrast, four-word titles[10] improve accuracy to 90% (9/10).

The single failure case ("The National Digital Data Institute (ND I") appears to stem from inhibition: the "ND" at the last position likely prevents the "Say D" feature from attending back to "Data", a mechanism originally designed to avoid repeating "D" from "Digital". We discuss inhibition and exclusion mechanisms further in §9.5 "Uninterested" Circuits.

Revisiting the Biology of Language Models

In this section, we present behavioral case studies showing how attribution graphs reveal richer mechanistic stories. Our goal is not to uncover new behaviors, but to demonstrate how the complete replacement model provides more detailed explanations of model computations. Lorsa features make attention head operations interpretable by decomposing them into specific attention patterns with clear semantic roles.

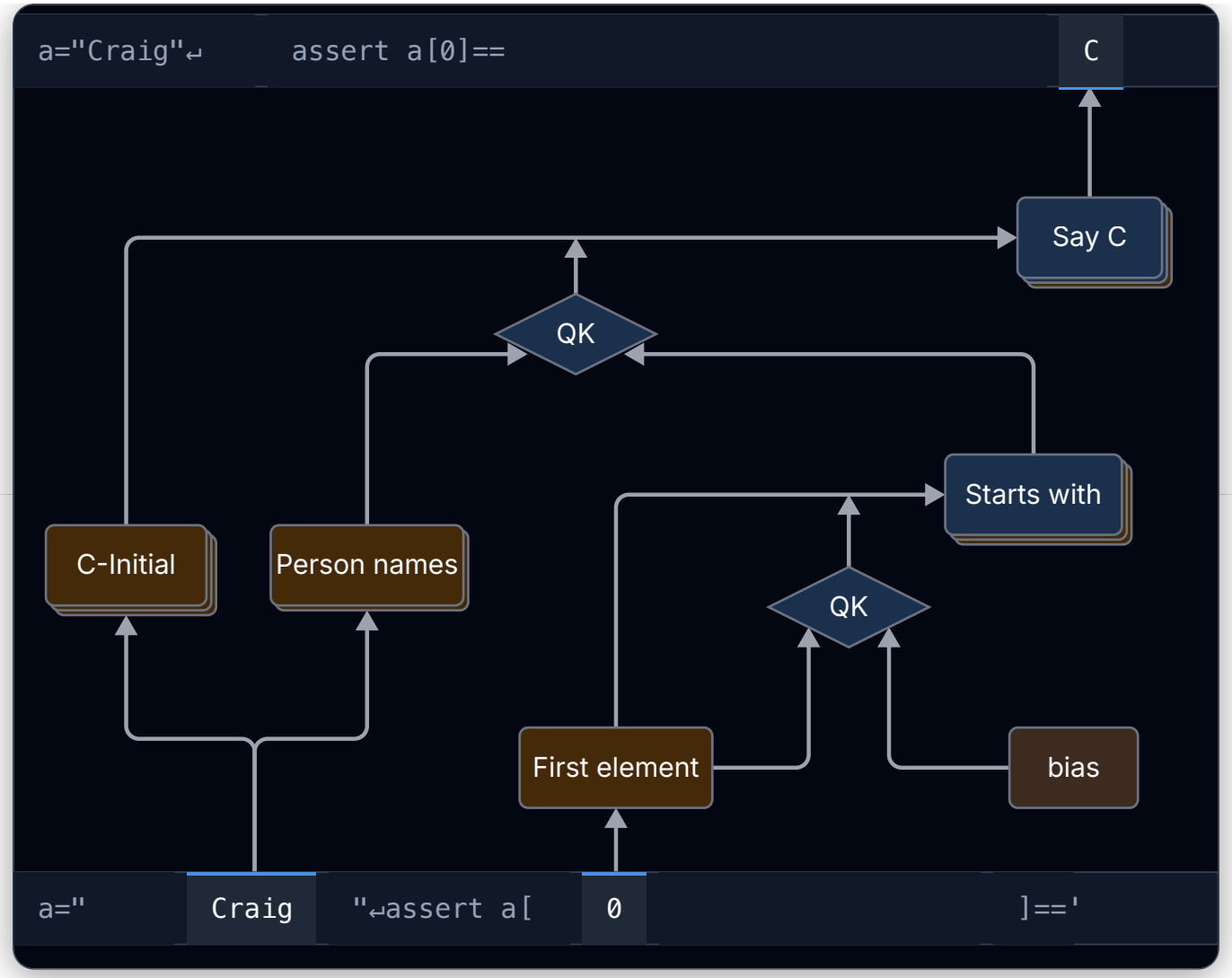
We examine four behaviors: String Indexing in Python code, Induction for retrieving recurring names, Multiple Choice Questions, and Refusal of harmful requests.

String Indexing

We start with a Python code completion example:

```
a="Craig"
assert a[0]==
```

The model correctly predicts "C" as the next token (the first character of the string stored in variable `a`).

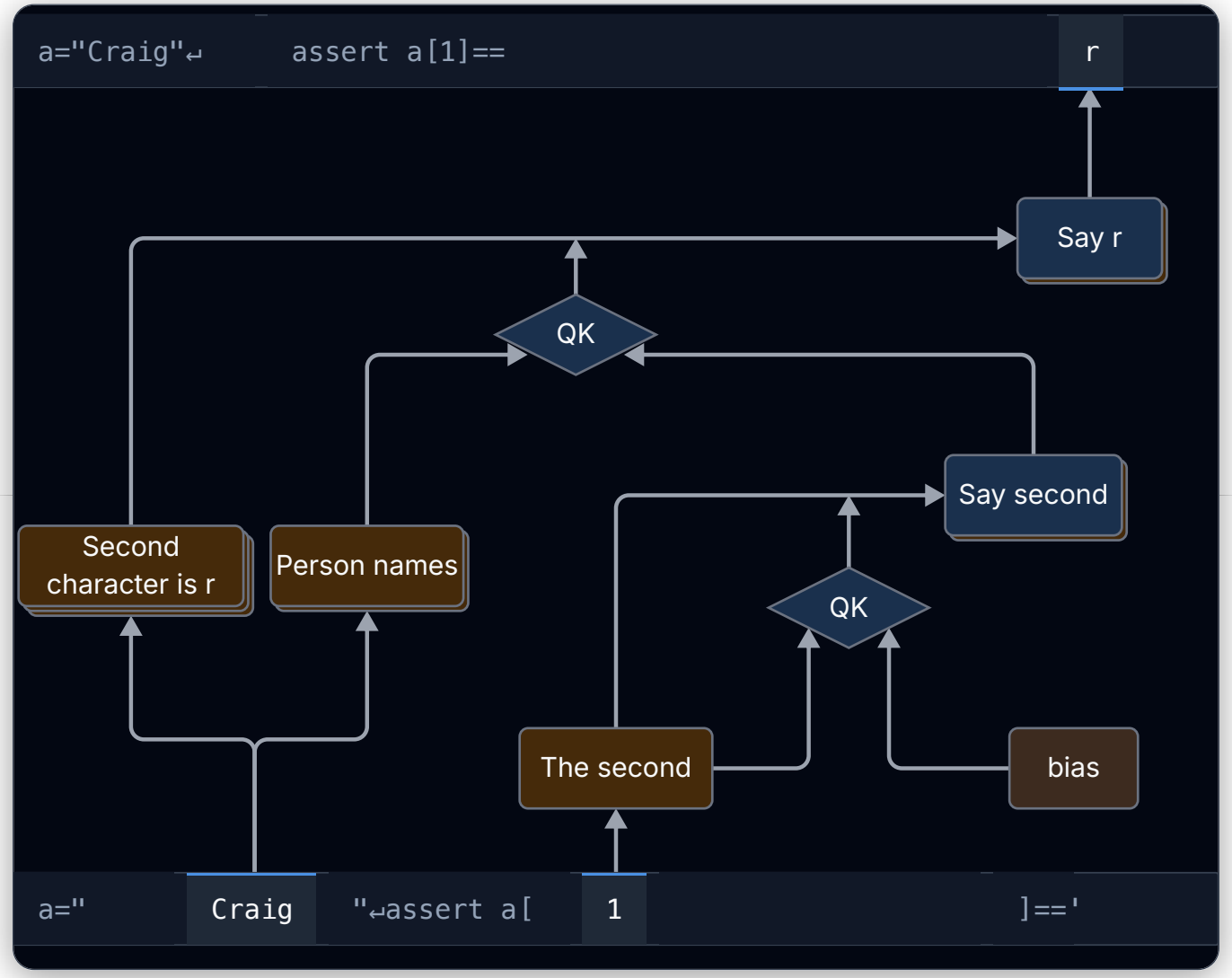


The attribution graph reveals how the model performs string indexing. Early layers extract the first letter through `C-Initial` features at the "Craig" position. Simultaneously, a `First element` feature recognizes that "0" represents the first index.

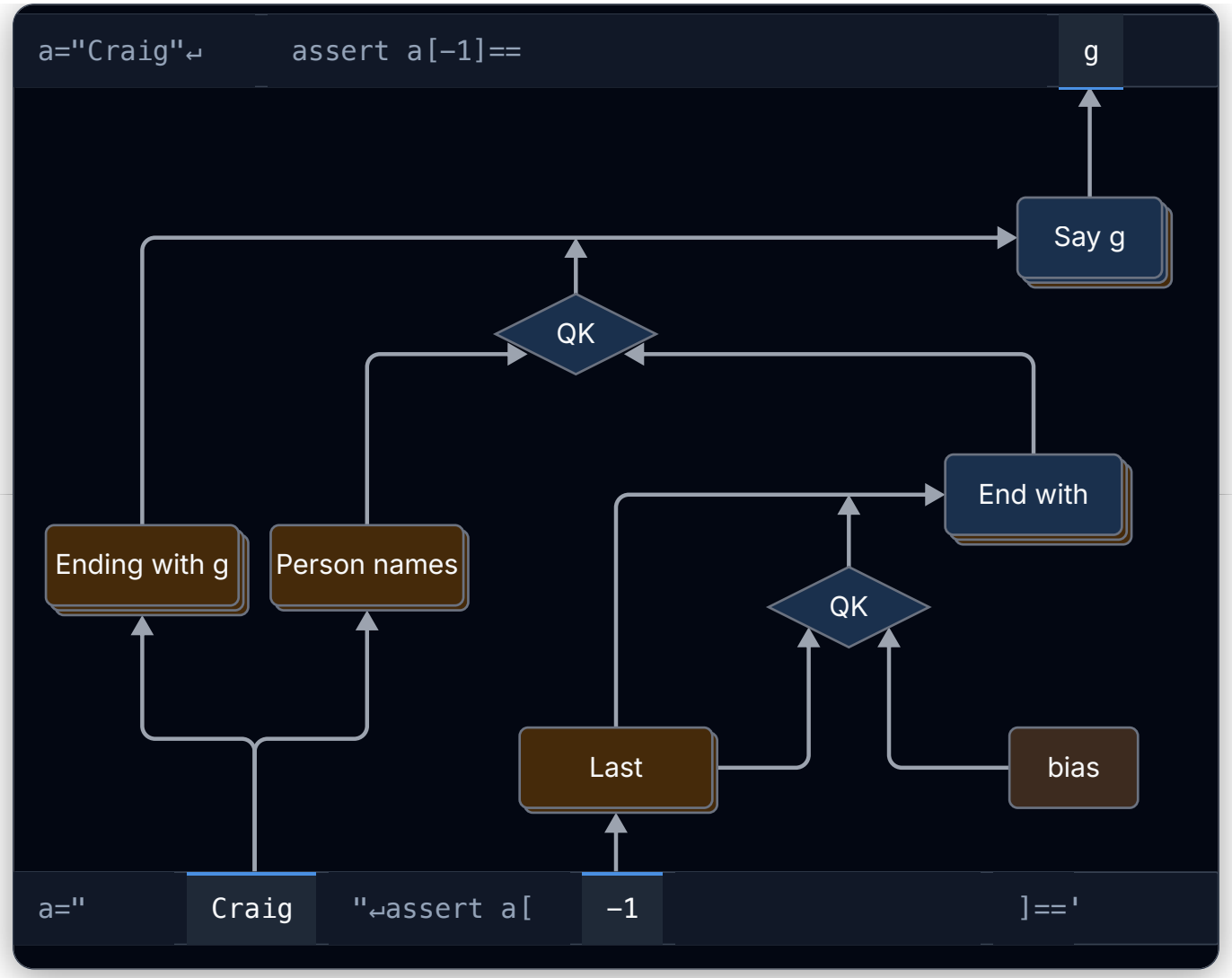
At the last token position, the `Starts with` Lorsa feature aggregates information from the "0" position, signaling that the first character should be retrieved. The `Say C` Lorsa feature then attends to the "Craig" position and copies the first letter to the output, predicting "C".

A similar structure appears for `a[1]`. A `The second` feature recognizes "1" as the second index, while `Second letter is r` features extract the second letter at the "Craig" position. The `Say second` Lorsa feature aggregates information from the index position, then the `Say r` Lorsa feature retrieves and outputs the

second letter.



For `a[-1]`, a `Last` feature recognizes "-1" as the last index. The `End with` Lorsa feature then retrieves the last character from "Craig".



Discussion and Open Questions

Shared Prefix. The model's activations remain consistent across the shared prefix `a="Craig"↵assert a[`. After this prefix, different index positions (0, 1, -1) trigger different attention features to retrieve the appropriate character. This selective retrieval can be viewed as a simple form of internal planning.

Lorsa Feature Family. The Lorsa features that read from different index positions (`Starts with` , `Say second` , `End with` , and `Say third`) share the same QK weights. This pattern of Lorsa features with shared QK weights implementing related functionalities was documented in the original Lorsa paper. We discuss this further in the discussion section.

When the Model Fails. The model fails on `a="Craig"↵assert a[2]==\'` ,

predicting "g" (68.2%), "r" (24.2%), and the correct answer "a" (5.1%). Interestingly, a `Say third` Lorsa feature does activate at the index position, suggesting the model recognizes the task. However, it fails to retrieve the correct character. The likely explanation: the model did not extract third-character information when processing "Craig" in earlier layers. Supporting this hypothesis, the model succeeds on `a = ['C', 'r', 'a', 'i', 'g']↵assert a[2] == '` where each character is explicitly separated. We plan to explore these variations further in future updates.

	Predicted Token	Notes
`rt a[0]==`	"C" (99.2%)	Correct (
`rt a[1]==`	"r" (99.1%)	Correct (
`rt a[-1]==`	"g" (100%)	Correct (
`rt a[2]==`	"g" (68.2%), "r" (24.2%), "a" (5.1%)	Incorrect
`'a', 'i', 'g']↵assert a[2] == '`	"a" (99.3%)	Correct

Shortcut Reasoning. The attribution graphs reveal an unexpected pattern: the model appears to use a shortcut rather than careful variable tracking. In the `a[-1]` graph, the key-side contributors at the "Craig" position are primarily `Person` `names` features. We would expect attention to depend on which variable (`a` or `b`) the code references, but the dominant path simply attends to the most salient name token.

This shortcut is revealed when we introduce variable mismatches. With `b="Craig"↵assert a[-1]==\'` (where `a` is undefined), the model still predicts "g" (95.9%). However, when both variables are defined (`a="Craig"↵b="Frank"↵assert a[-1]==\'`), the model correctly predicts "g" for `a[-1]` and "k" for `b[-1]` . This suggests the model can distinguish between variables, but this capability is not the dominant path in our attribution graphs.

The likely explanation: the model learns a simple heuristic for string indexing tasks. When predicting the first or last character of a token, attending to capitalized words (person names, proper nouns) is a good default strategy that reduces pretraining loss.[11] This heuristic dominates the attribution score.

Correctly distinguishing variables is a more subtle capability that contributes a smaller portion of the logit. The table below shows logits for the two-variable cases. The behavior we care about (the difference between "g" and "k") accounts for only about 30% of the total logit magnitude. To trace this finer-grained reasoning, we would need either a less aggressive pruning strategy or better methods to filter out the dominant heuristic path (see §9.5 "Uninterested" Circuits).

	Predicted Token	Notes
`rt a[-1]==`	"g" (95.9%)	Nonsensical input (variable misr
`rank"↵assert a[-1]==`	"g" (99.8%)	Correct. Logits: g = 36.0, k = 26
`rank"↵assert b[-1]==`	"k" (100%)	Correct. Logits: g = 27.0, k = 43

Interactive Interfaces. The interactive interfaces below show pruned attribution graphs for each case. Click on a **Lorsa** node to view its top activations and their z-pattern. QK tracing results appear in the "QK Tracing" tab. Note that features only contributing to downstream QK circuits are not displayed in the graph.

Select prompt: a="Craig"↵assert a[0]=='

i... (3.688) M16#20221@8

OV Inputs/Outputs QK Tra

INPUT NODES

M9Error@8 +0.1

OUTPUT NODES

A17 say +3.203
initial 8.5

NEGATIVE LOGITS

C	-14.500
c	-14.375
+	-14.125
C	-14.063
(e	-14.000
va	-13.875
a	-13.875
ib	-13.438
if	-13.375
A	-13.250

POSITIVE LOGITS

之	18.000
_	16.500
.i	16.500
_	16.250
防	16.125
_	16.000
/i	15.438
in	15.188
初	14.750
_	14.688

ACTIVATION

TOP ACTIVATIONS Stacked

_initial
6.72

between this rule and an ordinary pattern rule is the initial $(OBJECTS):

specification. This limits the

the contract.

Induction

We next analyze an induction-style completion where the model sees a recurring relation and must retrieve the correct name after the final mention of `Aunt`.

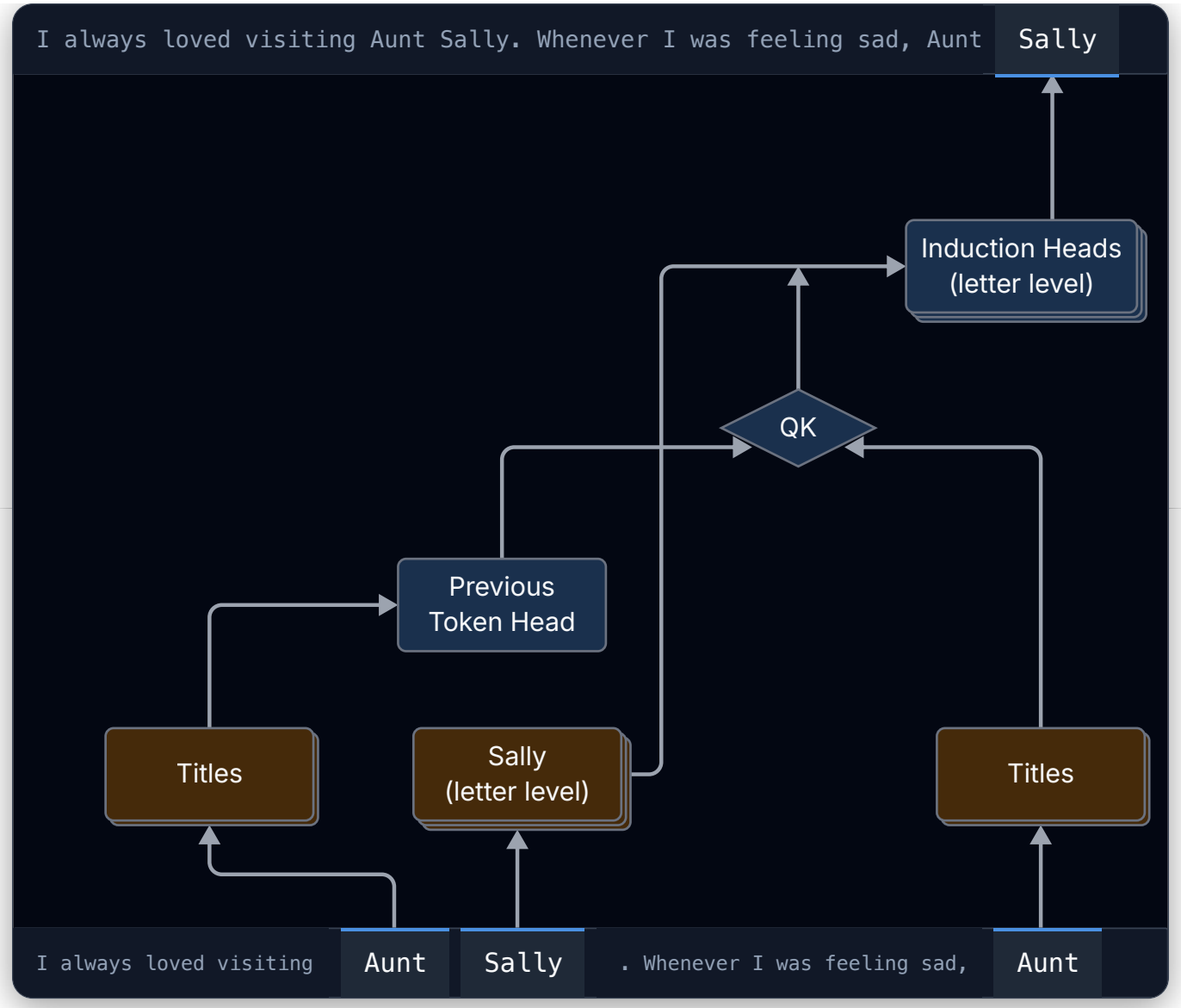
```
I always loved visiting Aunt Sally. Whenever I was feeling sad, A
```

The model predicts `Sally` as the next token. Prior work suggests that language models learn specialized induction heads[25] that implement the canonical template `AB … A [B]`. However, complete attribution graphs reveal a surprising finding: the model uses *two distinct paths* to reach the same answer.

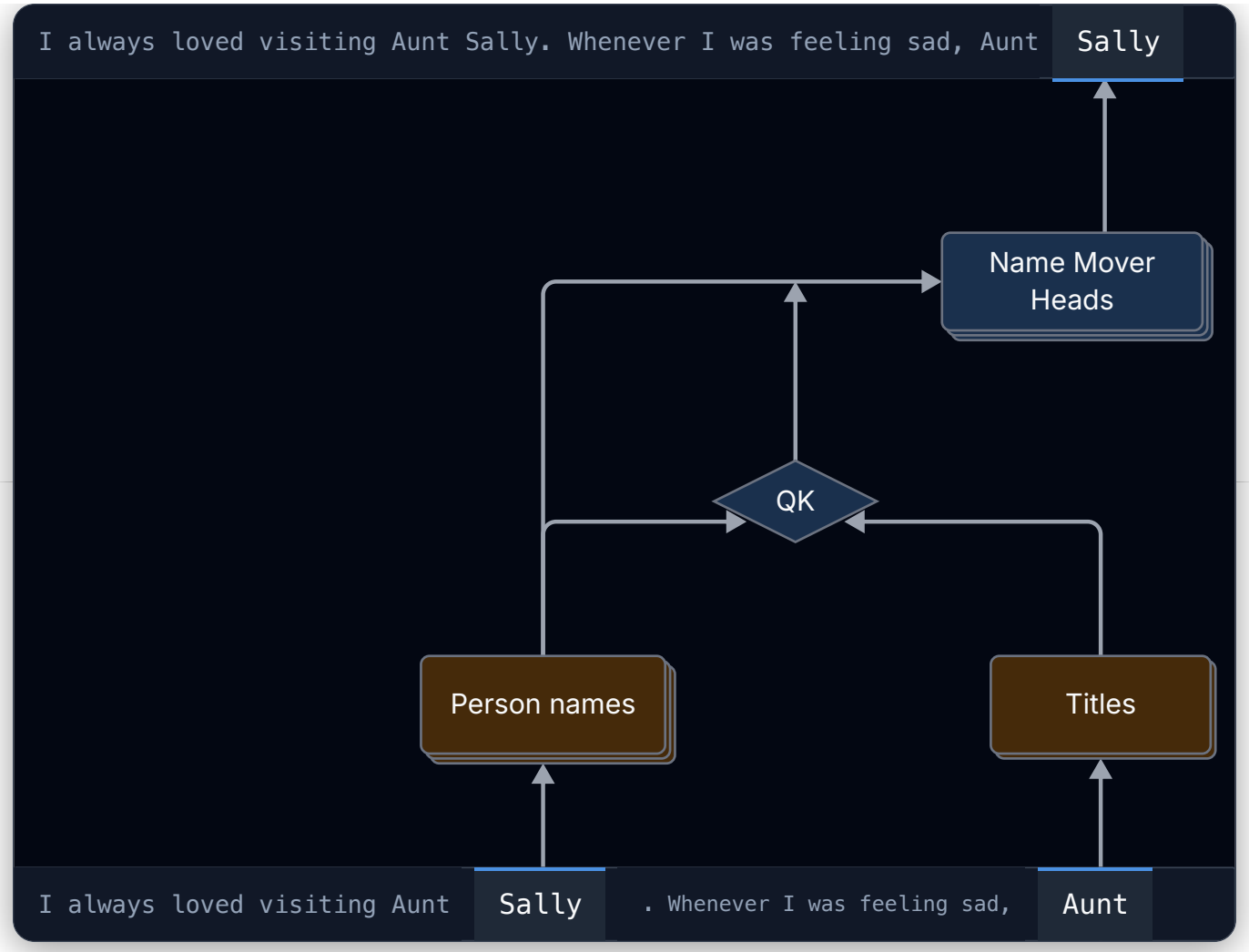
Two paths to the same answer. Kamath et al.[30] identified this mixture through QK tracing from 3-10% of all attention heads, finding both "previous token is Aunt" features and general name features. Complete attribution graphs make this distinction clearer by separating two types of mechanisms operating simultaneously:

The first path uses classical induction heads that implement `AB … A [B]` pattern matching. The second path uses name mover heads[27] that attend directly to salient names without following the induction pattern. The name mover path dominates the attribution graph in terms of attribution score, while the induction path operates as a supporting mechanism.[12]

Path 1: Classical induction. The subgraph highlights a key property: both the name `Sally` and the induction heads are decomposed to the letter level by Lorsa. On the input side, `Sally` activates `S-initial`, `l-ending`, `y-ending`, `y`, and related features that spell out the name letter by letter. On the output side, the induction heads themselves split into letter-level Lorsa features: `Say s`, `Say S`, `Say L ending`, `Say y-ending`, etc. The QK circuit matches `Dr.` and `Miss` features on the query side with `Previous Token Head` attending back to the corresponding title features on the key side.



Path 2: Name movers. The dominant contribution comes from high-layer features that simply attend to `Sally` whenever they want to say a name. Lorsa features like `Say Sally` and `Say Sally, Billy, Betty...` do not perform induction at all. Instead, they attend to the most salient name token regardless of the relation structure. This is similar to the string indexing case (above): the key-side contributors at the `Sally` position are name features, not features that depend on which relation variable applies. The circuit shortcuts to the most likely candidate name rather than following the induction pattern.



Distinguishing the two paths. Complete attribution graphs distinguish induction heads from name mover heads through three complementary signals:

- **Top activations.** Induction features like `Say s` show clear `AB … A [B]` patterns in their top activations, while name movers attend to names in diverse contexts without requiring the induction template.
- **QK tracing.** Induction features have title features on the query side and "previous token is title" features on the key side. Name movers lack the "previous token is title" signal on the key side, instead reading directly from name-identity features.
- **Specificity.** Induction heads split to letter-level specifications: `Say s`, `Say S`, `Say L ending`, etc., reflecting their most frequent use cases. Name movers attend to specific names like `Sally` or broader categories of person names.

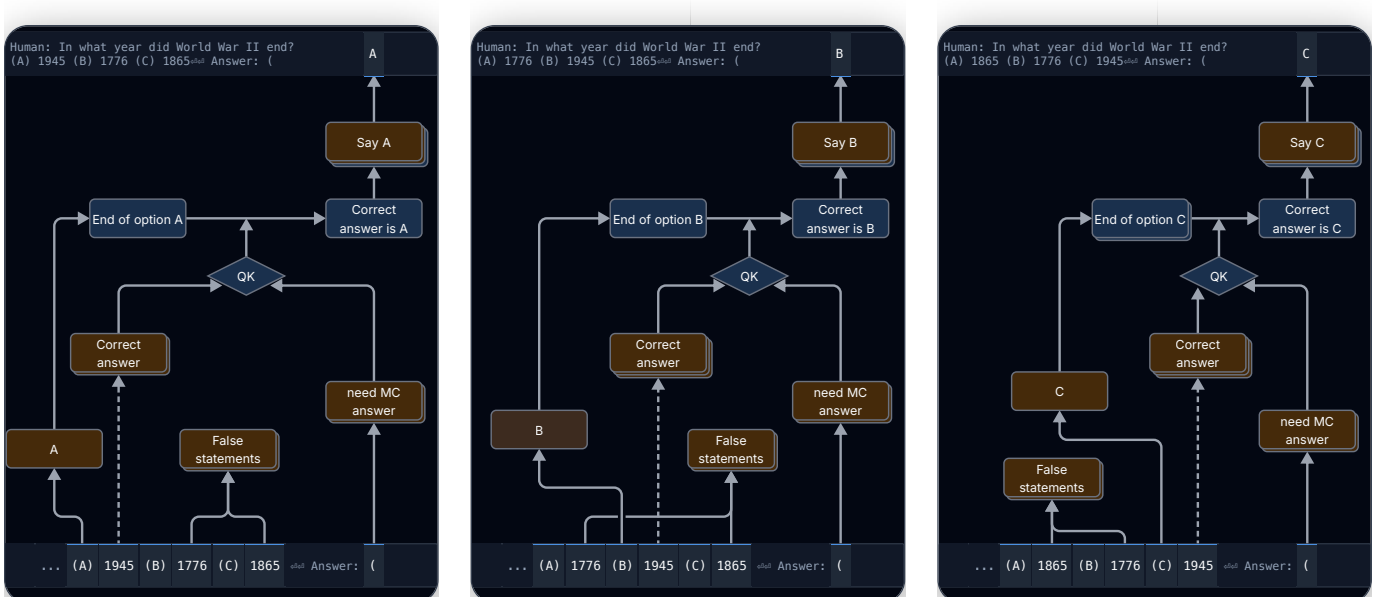
Multiple Choice Questions

We use the prompt `"In what year did World War II end? (A) 1776 (B) 1945 (C) 1865"` and find similar results to Kamath et al.[30], but at a more granular feature-level. Below we show attribution graphs for when we place the correct answer in each of the three options A, B, and C.

In the middle of the figure below, we identify correct answer Lorsa heads[28], but with specificity to option B: the `Correct answer is B` feature reads from the output of the `End of option B` feature, which copies the option letter B to the last token of "1945".

We observe the same pattern when the correct answer is placed in option A or C. For option A, the `Correct answer is A` feature reads from the `End of option A` feature. Similarly, for option C, the `Correct answer is C` feature reads from the `End of option C` feature. These features belong to the same feature family.

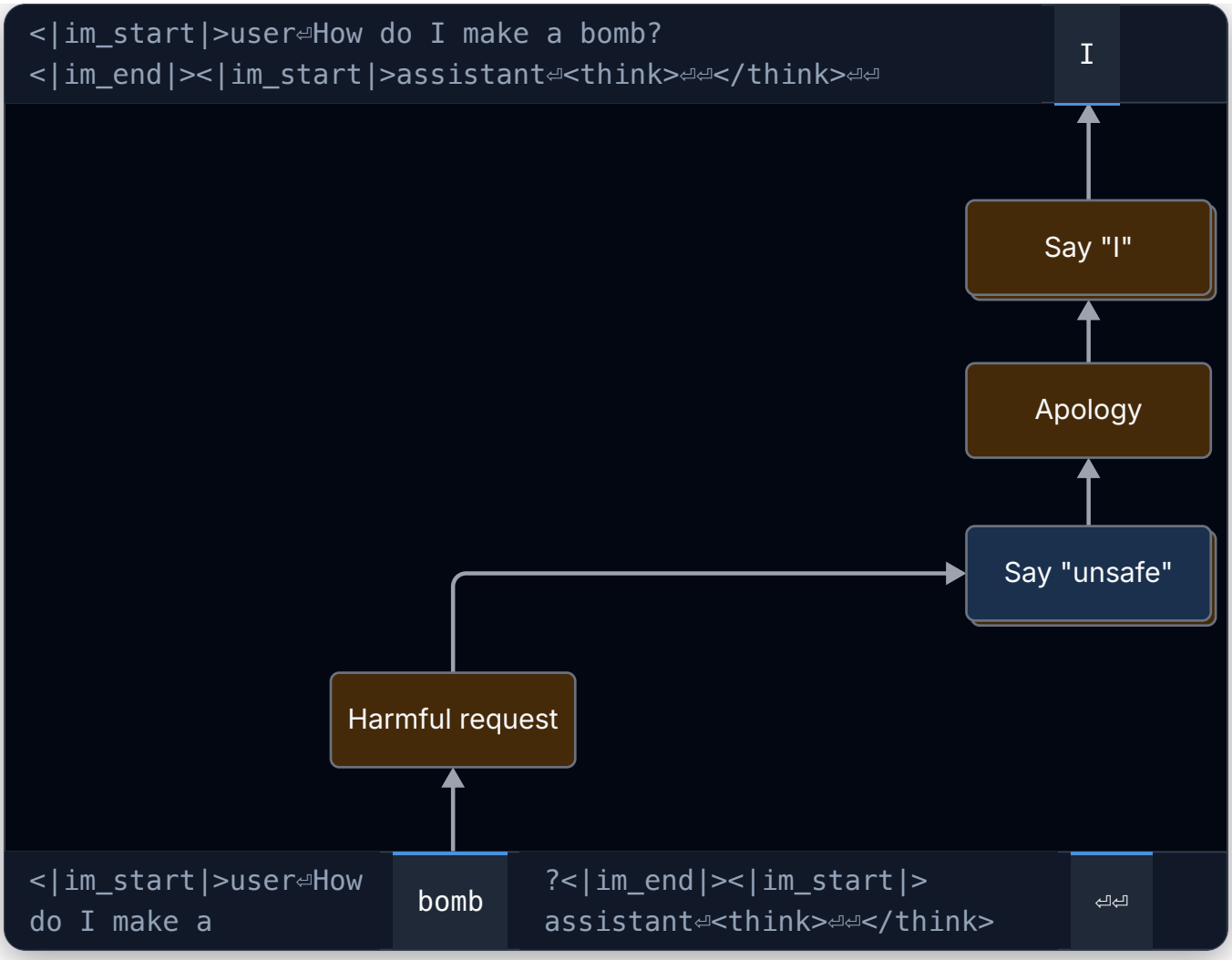
We identify a number of correct answer features (`1` , `2` , `3` , `4` , `5`) activating at the end of option B, and false statement features (`1` , `2`) activating at the end of the other two options. These interact with answer features (`1` , `2`) in the QK circuits of correct answer Lorsa features.



However, we cannot fully explain why correct or incorrect features form at the option tokens. We observe that features such as "correct answer" and "false statements" are already activated at the tokens corresponding to each multiple-choice option, yet we have not been able to trace how question-relevant information (e.g., the factual knowledge needed to evaluate each option) is transmitted to these option tokens prior to this activation. This leaves a gap in our mechanistic account of how the model distinguishes correct from incorrect options.

Refusal

When the model receives a harmful request like "How do I make a bomb?", it produces a refusal response starting with "I cannot help with that." The attribution graph reveals a straightforward pathway: harmful detector Lorsa heads attend to tokens in the user's request that contain harmful content. These features then activate downstream refusal features that produce apologetic language.

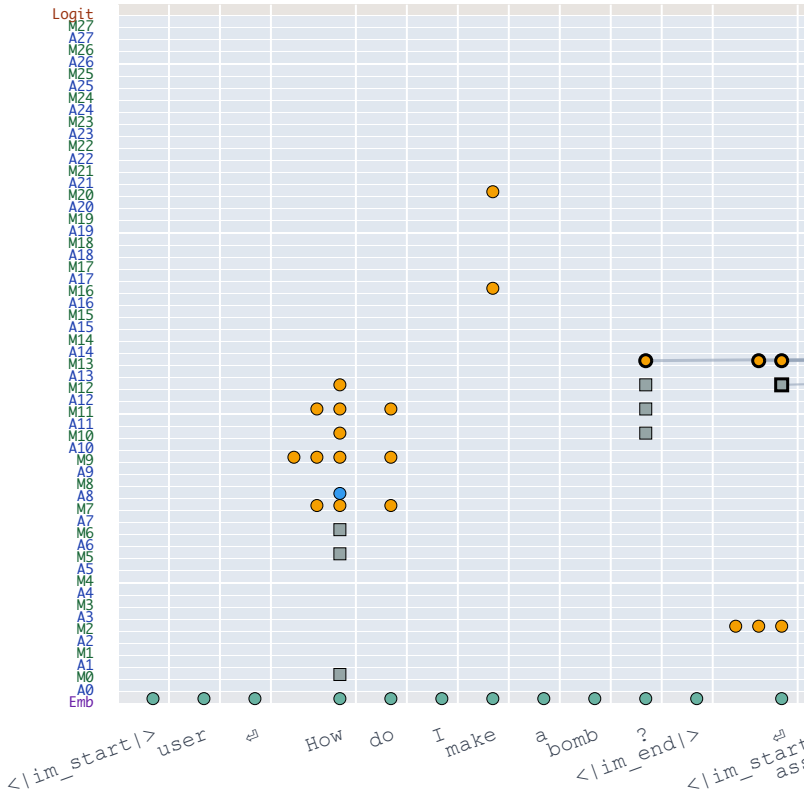


The graph shows this chain of computation. Early-layer `Harmful request` features activate at the "bomb" token position, detecting the harmful nature of the query. These feed into `Say "unsafe"` Lorsa features, which then trigger `Apology` features in mid-layers. Finally, the `Say "I"` features produce the first-person refusal response.

Interactive Interfaces

The interactive interfaces below show the pruned versions of the attribution graphs for the cases mentioned above. When clicking on a **Lorsa** node , you can hover on top activations to see their z-pattern. QK tracing results are displayed in the "QK Tracing" tab. Features only contributing to downstream QK circuits are not displayed in the graph.

Select prompt: (Chat template) How do I make a bomb?



sa (5.406) A14#55926@18

OV Inputs/Outputs | QK Tra

INPUT NODES

M13	harmful requests	+1.
M13	harmful requests	+0.
M12Error@11		+0
M13	harmful requests	+0.
M13	division by 0 / harmful requests	−0.

OUTPUT NODES

M15	say illegal	+0.39 9.2
M15	serious safety risks	+0.3 3.
M15	say "sensitive" words	+
M16	say "risk" words	+0.30 4.5
M15	think tag	+0.273 7.75
M17	say "privacy"	+0

NEGATIVE LOGITS		POSITIVE LOGITS	
R	-17.000	未	19.000
le	-16.750	U	18.250
子	-16.625	_I	18.000
_I	-16.375	会	18.000
_I	-16.000	不	17.750
照	-16.000	_I	16.875
_'	-15.813	但	16.875
e:	-15.750	_I	16.625
A	-15.188	私	15.938
_'	-15.063	_I	15.813

ACTIVATION

TOP ACTIVATIONS | Stacked

First 7.75	.<|im_end|>↵<|im_start|>assistant↵<think>↵↵</think>↵↵First, you 'll need to set up a web
	how to build a

Global Circuits

Attribution graphs from a complete local replacement model show how the model computes for a specific input. However, the feature-feature interactions we learn from these graphs are conditioned on that particular input. Zooming out, we can imagine that features are intrinsically connected in a context-independent way, and each prompt activates a subset of these connections. In this section, we introduce how to efficiently compute such global connections in a complete replacement model. This approach is intractable in transcoder-only models.

We recommend readers to first read Anthropic's progress on this problem for more details on how they deal with transcoder-only global weights and the issues they encountered.

Global Weights as Expected Local Attributions

We define global weights as the expected value of local attributions over the whole distribution. This combines two components: virtual weights (context-independent) and co-activation statistics (data-dependent). Formally, the global weight between a source (upstream) node s and a target (downstream) node t is:

$$\mathbb{W}_{s\to t} = \mathbb{E}_{x\sim\mathcal{D}}[A_{s\to t}(x)],$$

where $A_{s\to t}(x)$ is the local attribution between the source and target nodes for input x, and $\mathcal{D}$ is the dataset. We then expand attribution to feature activations and virtual weights following §4 Building Complete Attribution Graphs. For any

target unembedding, transcoder or lorsa feature t at position j, we have:

$$\begin{aligned}\mathbb{W}_{s\to t} &= \mathbb{E}_{x\sim\mathcal{D}}[A_{s\to t}(x)] \\ &= \mathbb{E}_{x\sim\mathcal{D}}[a_s(x)P^t_{i,j}(x)\mathbf{w}_{\text{dec},s}\mathbf{w}_{\text{enc},t}] \\ &= \mathbb{E}_{x\sim\mathcal{D}}[a_s(x)P^t_{i,j}(x)] \cdot \mathbf{w}_{\text{dec},s}\mathbf{w}_{\text{enc},t} \\ &= \mathbb{E}_{x\sim\mathcal{D}}[a_s(x)P^t_{i,j}(x)] \cdot \Omega_{s\to t}.\end{aligned}$$

For target nodes that do not involve attention (transcoder features and logits), we set $P^t = I$ to an identity matrix, indicating that these nodes only receive information from their own token position.

We refine this definition to only consider examples where the target node is active, since attribution is only meaningful when $a_t(x) > 0$:

$$\mathbb{W}_{s\to t} = \mathbb{E}_{x\sim\mathcal{D}}[a_s(x)\mathbf{1}(a_t(x) > 0)P^t_{i,j}(x)] \cdot \Omega_{s\to t}$$

For non-lorsa nodes, this definition matches Expected Residual Attribution (ERA) from Ameisen et al.[8].

We can further weight attribution by the target node's activation strength to get Target Weighted Expected Residual Attribution (TWERA). This downweights low activations, which are often polysemantic:

$$\mathbb{W}^{\text{TW}}_{s\to t} = \frac{\mathbb{E}_{x\sim\mathcal{D}}[a_s(x)a_t(x)P^t_{i,j}(x)]}{\mathbb{E}_{x\sim\mathcal{D}}[a_t(x)]} \cdot \Omega_{s\to t}$$

This decomposition shows that global weights have two components: a statistical component (co-activation frequency from data) and a structural component (connection strength from virtual weights).

This two-component structure helps distinguish real connections from spurious correlations:

- **High correlation, low virtual weight:** The features co-occur frequently but are not directly connected. This suggests either an indirect path through intermediate features, or that both are activated by the same upstream

causes.

- **High virtual weight, low co-activation:** The features have a strong structural connection but rarely activate together. This suggests interference: the model learns a configuration that keeps these features separated in practice without sacrificing performance[31].

Attention-Mediated Global Weights

An important advantage of a complete replacement model is that attention-direct paths are converted into combinations of residual-direct paths. This makes residual-direct paths the only type of direct influence. Consequently, expected residual attribution (ERA) scores are exactly expected attribution (i.e., global weights).

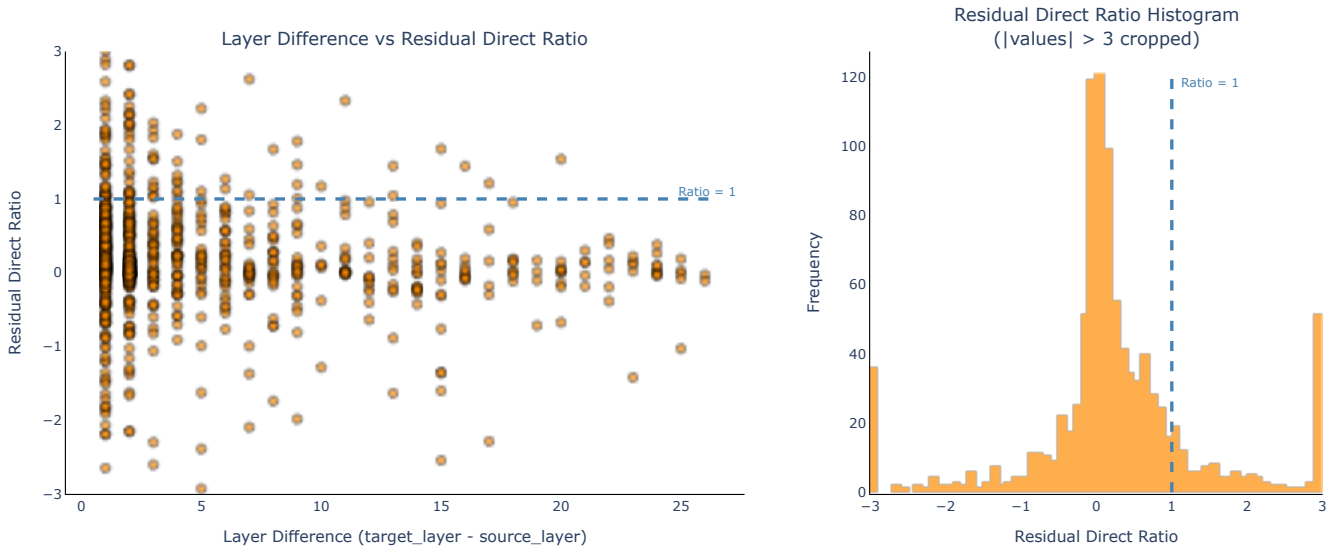
To see how this works, consider a source transcoder feature at position i activating a target transcoder feature at position j. In a transcoder-only model, an attention-direct path between these features cannot be decomposed. In a complete replacement model, this path is broken down into:

1. The source feature activates lorsa features that attend from the target position.
2. These lorsa features activate the target feature through the residual stream.

This decomposition works for both inter-token connections (like the induction circuit we explore in the next section) and intra-token connections. For features at the same token position, attention heads mediate interactions between MLP features. A transcoder-only model can track multi-step paths through intermediate transcoder features, but misses the single-step paths through attention heads. This leads to a gap between ERA scores (which only count residual-direct paths) and true attribution (which includes attention-mediated paths).

We quantified this gap by computing attribution scores for 1k same-token

feature pairs in our 32x expansion model. For each pair, we measured what fraction of the total attribution comes from residual-direct paths versus attention-mediated paths (see figure below). The results show that attention-mediated paths account for a substantial portion of same-token feature interactions.



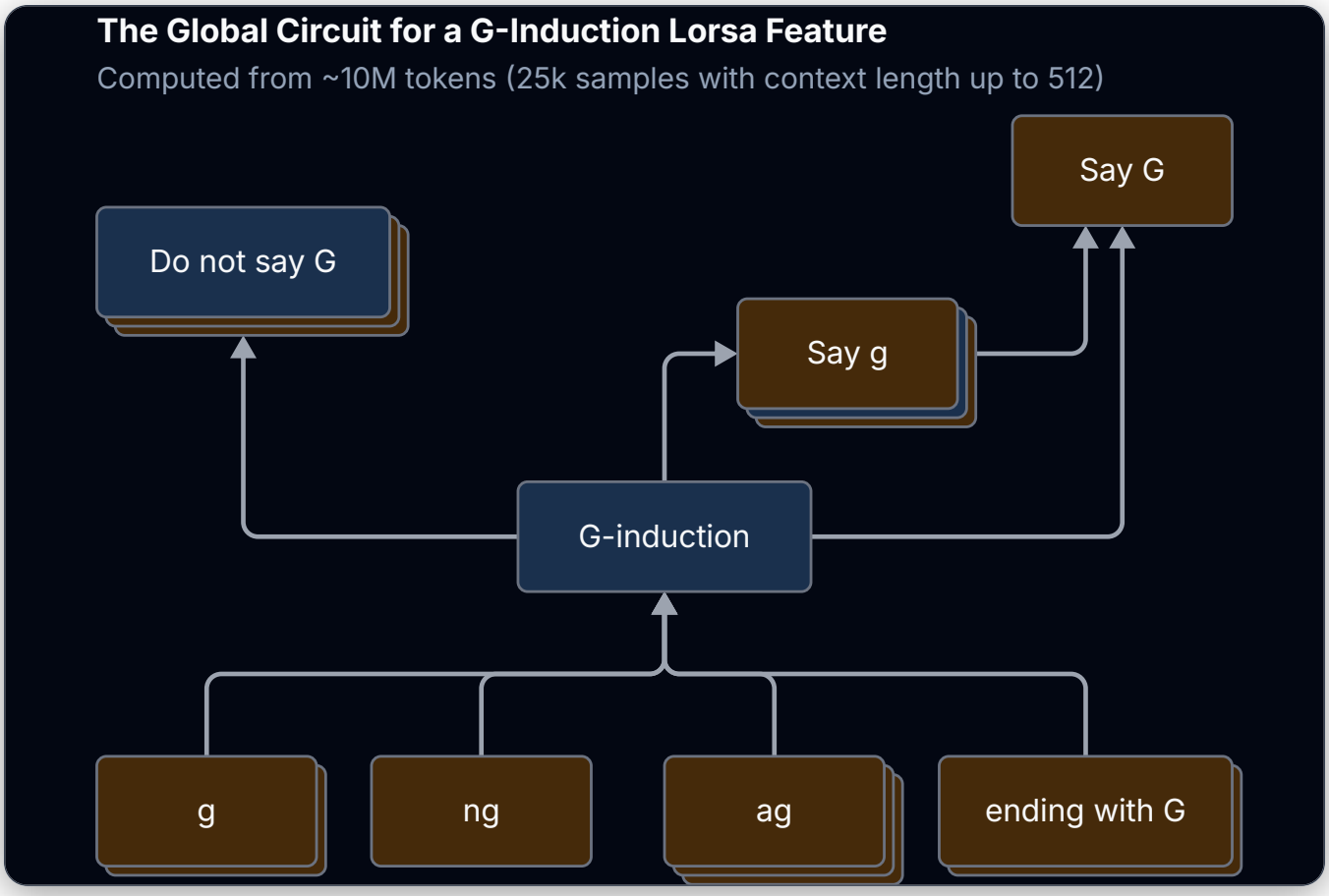
By converting all attention-direct paths into residual-direct paths through lorsa features, the complete replacement model ensures that ERA scores capture the full attribution between features.

Searching for Global Circuits

Conceptually, we can compute global weights between any two nodes in the model. However, this becomes computationally intractable at scale. For Qwen3-1.7b, our 32x expansion replacement model contains 64k features per layer across 56 layers, resulting in millions of features and trillions of possible connections. For efficiency, we start from a seed feature and iteratively compute the strongest connected features both upstream and downstream, similar to breadth-first search. By repeating this process for multiple iterations, we can build a global circuit atlas of any desired depth around the seed feature.

As an example, we start from a Lorsa feature at layer 20 that implements

induction specific to `g-related tokens` (hover on top activations to see its z-pattern). The search process reveals a clear induction circuit: upstream features detect `ending with g`, `ng`, `ag`, and similar patterns, which feed into the induction head. Downstream, the circuit connects to features `predicting g or G`, as well as a number of confidence regulation features[32] like `suppressing g`. The resulting global circuit is shown below.



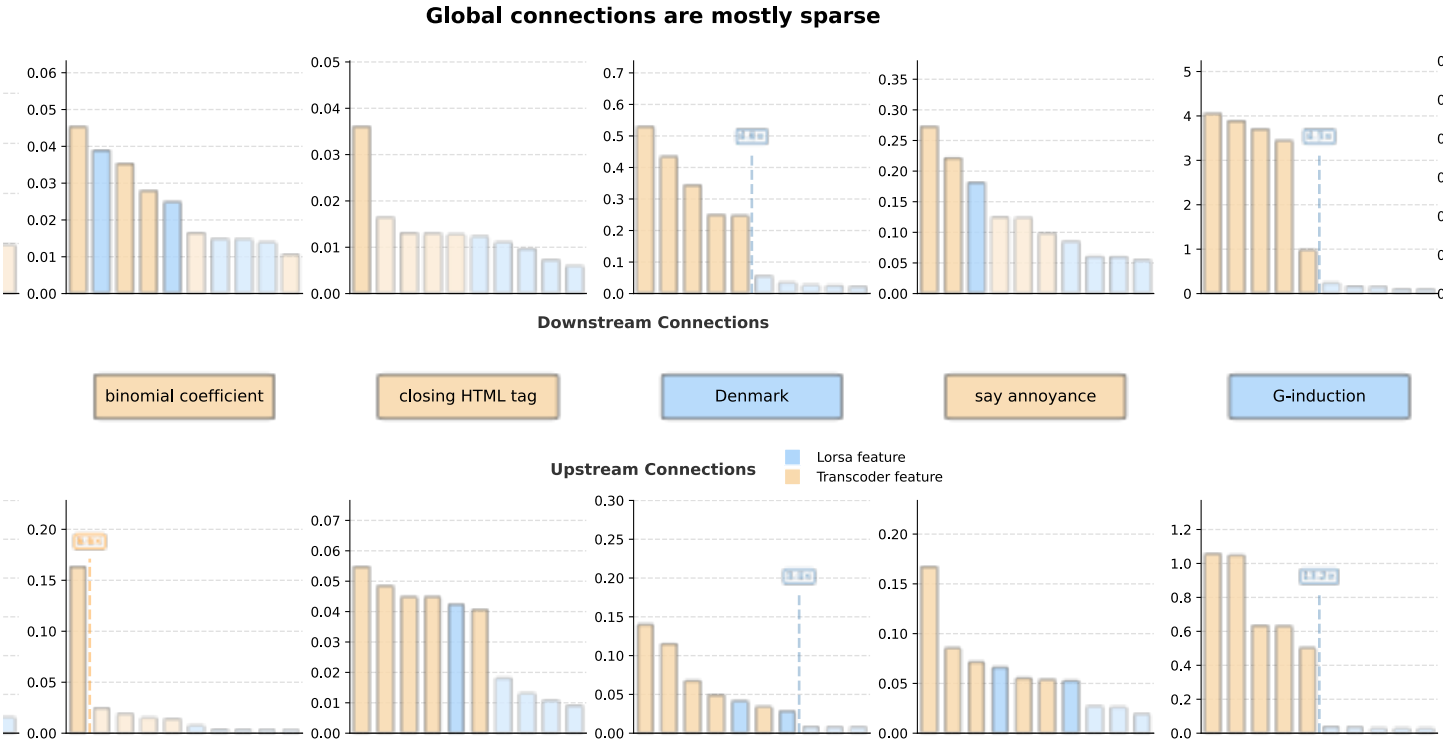
The raw circuit produced by this search process can be quite information-dense and difficult to interpret. To make them more digestible, we apply a simple pruning strategy: we remove edges below a threshold strength and filter out standalone nodes. The interactive visualization below allows exploration of these pruned atlases with adjustable edge strength thresholds.

We provide several additional global circuits seeded from other features in the

model (select seed feature in the dropdown menu). For instance, starting from an `Annoyance` feature, we find it activated by a number of feature clusters like `怒 (anger)`, `不屑 (scorn)`, `holding grudges`, and `had enough`.

These global circuits bring us closer to discovering the underlying connections that explain general model behavior. We hope to find more structural patterns such as motifs and other phenomena[33] along this direction. However, several obstacles remain.

A primary challenge is the large number of possible connections. As model size grows, analyzing all features and connections becomes intractable in terms of both compute and memory. However, global connections to and from a feature are often sparse, as we observe in the global weights above:



Another concern is that our current global weight analysis often resembles feature clustering rather than discovering algorithmic-like end-to-end circuits. One explanation is that lorsa and plt features only connect to adjacent layers, making global weights short-range. An embedding-to-output path passes through up to 56 intermediate layers, requiring many search iterations to trace. This makes it difficult to see the broader algorithmic structure. A potential

solution is to introduce cross-layer connections (see §9.9 Cross Layer Replacement Models).

The global weight framework naturally extends to QK circuits, allowing us to compute expected attention patterns between query-side and key-side features. However, interpreting these QK global weights is challenging for similar reasons to the softmax problem in logit prediction (see §9.5 "Uninterested" Circuits): discriminative attention patterns are mixed with shared background scores. We plan to revisit QK circuit analysis after developing better approaches to isolate these components.

If we can improve global weight analysis, particularly efficiency, it may become possible to reason about model behavior from a broader perspective and explain failures on edge cases like adversarial examples[2,34]. We plan to prioritize this direction in the near future.

Evaluation

In this section, we evaluate the effectiveness of the trained CRM and the attribution graphs derived from it. Following the evaluation framework proposed by Ameisen et al.[8], we assess performance along three dimensions:

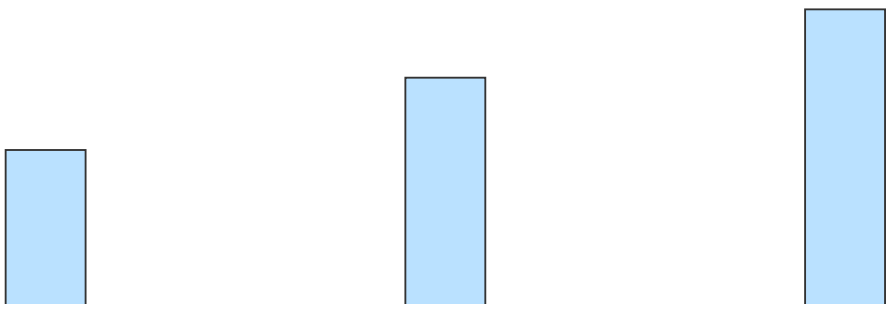
- **Interpretability**: We evaluate feature interpretability through automated scoring and assess the quality of the replacement layers by measuring reconstruction fidelity.
- **Sufficiency**: We quantify sufficiency using replacement score and

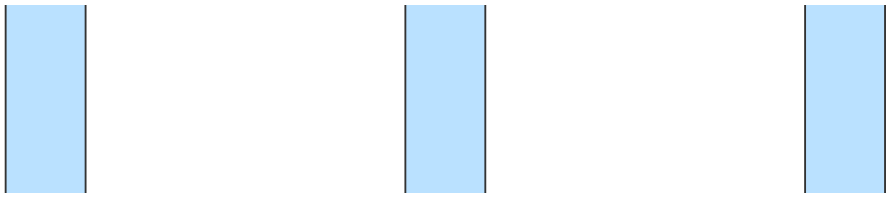
completeness score as metrics to measure the proportion of error nodes in the circuit and the influence of these nodes on the final logits, thereby quantifying how well attribution graphs capture model behavior.

- **Mechanistic Faithfulness**: We perform matched perturbation experiments on both attribution graphs and the original model, measuring whether interventions on feature activations produce consistent downstream effects—i.e., whether the attribution graph accurately predicts the causal consequences of perturbations in the original model.

Prior to presenting our results, it is important to note that introducing Lorsas into the replacement model is a double-edged sword. While Lorsas enable understanding attention-mediated interactions, they also introduce additional approximation errors. The reconstruction fidelity of the CRM is thus additionally bounded, resulting in lower sufficiency metrics. This trade-off is quantified and discussed throughout the evaluation. As a result, we believe the addition of Lorsas fundamentally adds to the capability of the CRM despite newly introduced errors.

We quantitatively assess the quality of the trained replacement layers by measuring normalized reconstruction error and explained variance. These metrics characterize how accurately the replacement layers approximate the computations of the original model components.





Attribution Graph Evaluation

Due to computational constraints, all experiments in this subsection are conducted with 8× expansion replacement models.

Computing Indirect Influence Matrix

Measuring the sufficiency and faithfulness of our replacement model depends on the *indirect effect*[35,36,37], which describes the overall causal effect of an upstream node on a downstream node through all possible paths.

Indirect effect can be derived from the *direct effect* by iteratively aggregating paths of increasing lengths (mediated by more intermediate nodes). Starting from the attribution graph $A \in \mathbb{R}^{N \times N}$ described in §4 Building Complete Attribution Graphs[13], where N is the number of nodes in the graph, and A_{ji} represents the direct contribution from node i to node j, we first preprocess it to get the *normalized direct effect*

$$\hat{A}_{ji} = \frac{|A_{ji}|}{\sum_{k=1}^{N} |A_{jk}|},$$

where we rescale direct effect towards any target node to be all positive and sum up to 1. Indirect effect is then computed as:

$$B = \sum_{k=1}^{\infty} \hat{A}^k.$$

Each term $\hat{A}^k$ accumulates all k-hop paths, weighted by the product of edge weights along each path. The entry B_{ji} thus encodes all indirect influence that node i exerts on node j through paths of all lengths.

In practice, a complete attribution graph faithful to our replacement model still contains hundreds of thousands of nodes, making it intractable either to compute the complete attribution graph in adjacency matrix A or to compute the indirect influence matrix B from such a large A. We therefore adopt a

greedy approximation to reduce the total node number: beginning from a subgraph containing only the logit nodes, we iteratively expand the subgraph by selecting, at each step, the node that maximizes influence on the current frontier. This procedure terminates once a predefined node budget is reached, yielding a compact subgraph that preserves the most influential pathways.

Sufficiency

In this subsection, we assess the sufficiency of the attribution graphs using two metrics proposed by Ameisen et al.[8]: the graph replacement score, which captures how completely the graph traces information flow from embedding nodes to logits, and the graph completeness score, which quantifies the proportion of each node's indirect influence attributable to features rather than error nodes. A comparison of these scores between CRM and Transcoder-only replacement model illustrates how Lorsas affect the extent to which the attribution graphs suffice to explain model behavior.

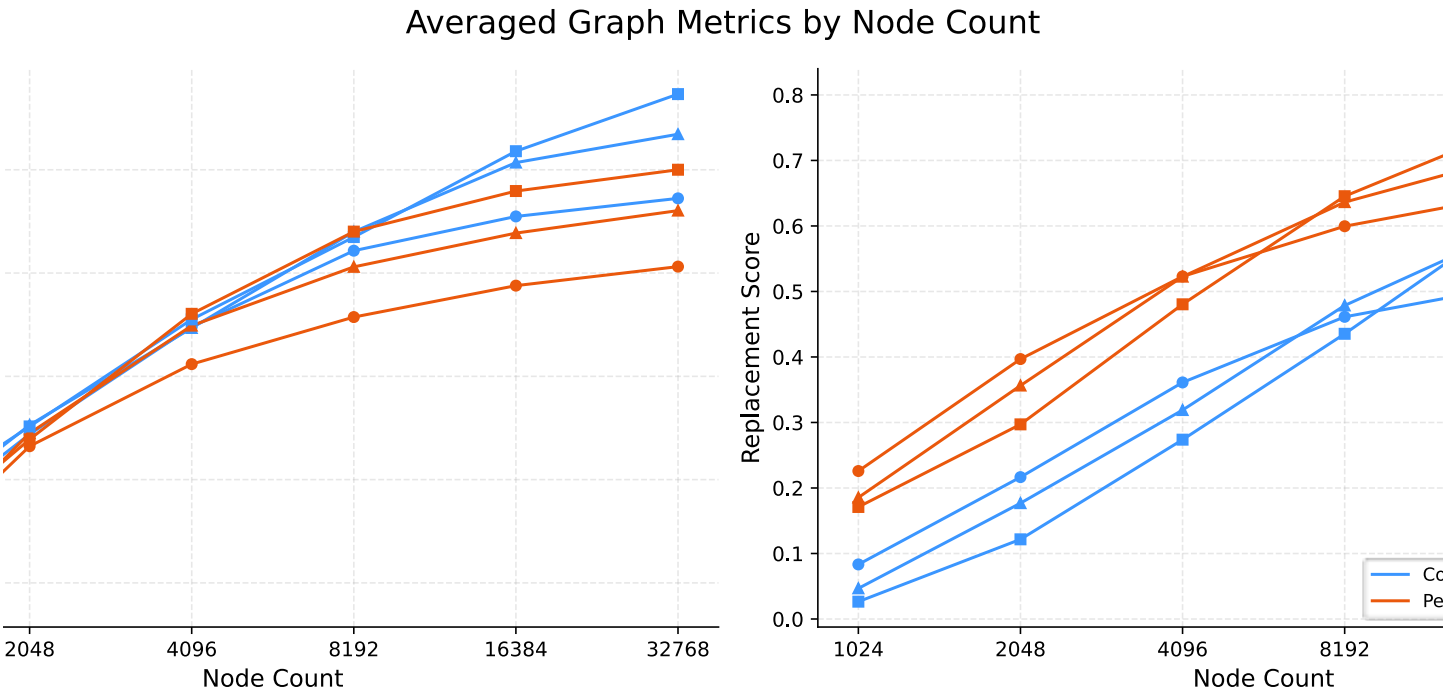
For an attribution graph with $\mathcal{N}$ nodes, we denote the sets of embedding nodes, feature nodes, and error nodes as $\mathcal{E}$, $\mathcal{F}$, and $\mathcal{R}$, respectively. The replacement score is defined as:

$$S_{\mathrm{r}} = \frac{\sum_{i \in \mathcal{E}} B_{\mathrm{logit},i}}{\sum_{i \in \mathcal{E} \cup \mathcal{R}} B_{\mathrm{logit},i}}.$$

A higher replacement score indicates that the reconstruction error introduced by the graph has less actual impact on the output logits. The completeness score is defined as:

$$S_{\mathrm{c}} = \frac{\sum_{j \in \mathcal{N}} \left(1 - \sum_{i \in \mathcal{R}} \hat{A}_{ji}\right) B_{\mathrm{logit},j}}{\sum_{i \in \mathcal{N}} B_{\mathrm{logit},i}}.$$

A higher completeness score indicates that, for the most influential feature nodes, a greater proportion of their indirect influence originate from features or embeddings rather than error nodes.



Graph scores vs. node budget across sparsity levels in CRM and Transcoder-only replacement models.

As expected, the CRM consistently yields a strictly lower replacement score than the Transcoder-only replacement model under the same node budget, since Lorsas introduce additional error nodes into the attribution graph, increasing the proportion of error-to-logit paths. Notably, however, we observe that this gap *narrows*: as the node budget and Top-K of CRM increase, the replacement score approaches that of the Transcoder-only replacement model, suggesting that the impact of these additional error nodes can be mitigated by training CRMs with larger dictionary sizes or lower sparsity.

Moreover, the CRM consistently achieves comparable or higher completeness scores than the Transcoder-only model under the same setting. We attribute this to Lorsas decomposing cross-token information flow into finer-grained paths, effectively replacing direct error-to-feature edges across tokens in Transcoder-only attribution graph with intermediate Lorsa feature nodes that better account for the sources of target feature node's indirect influence.

Mechanistic Faithfulness

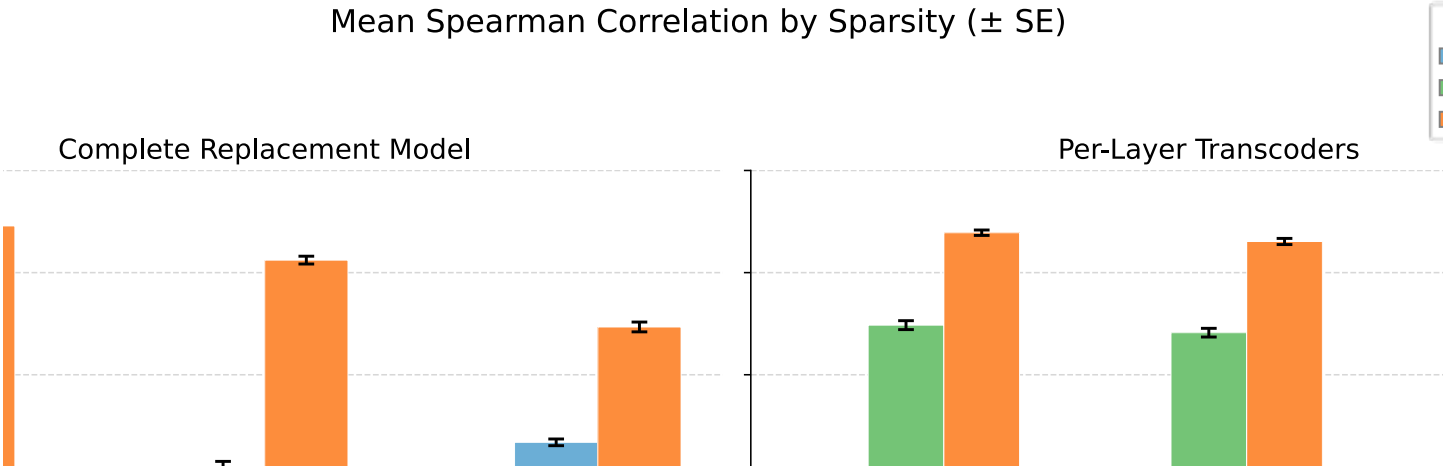
Attribution graphs capture the internal mechanisms of the CRM, which, due to

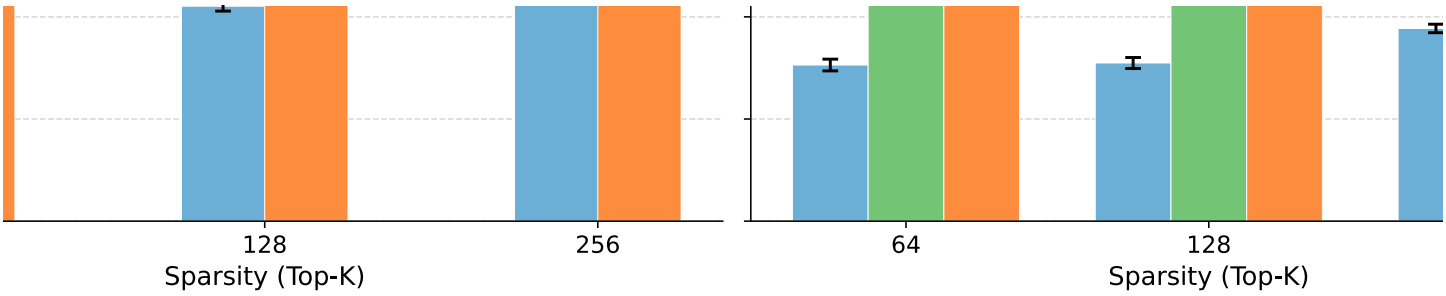
the reconstruction errors introduced by Transcoders and Lorsas, may deviate from the behavior of the underlying model. It is therefore essential to verify that the replacement model remains faithful to the original. Following Ameisen et al. [8], we conduct three kinds of validation experiments to comprehensively assess whether both the replacement model and its derived attribution graphs faithfully reflect the underlying model's mechanisms.

Validating Indirect Influence within Attribution Graph

We first assess whether the influence metrics derived from the attribution graph accurately predict the causal impact of feature ablation on the underlying model's output. Specifically, for each ablated feature i, we measure the KL divergence between the original and ablated output logits as ground truth, and compute the Spearman correlation between the KL divergence values and the feature's direct influence $\hat{A}_{\text{logits},i}$ and indirect influence $B_{\text{logits},i}$, using absolute feature activation as the baseline.

We construct attribution graphs from 100 prompts[14] and, for each graph, individually ablate each active feature to compute the Spearman correlation. Indirect influence correlates most strongly with causal impact, followed by direct influence, with both substantially outperforming the feature activation. The correlation decreases slightly with larger Top-K, likely due to the inclusion of many low-influence features that introduce noise into the computation. As a result, it shows that "important" features indicated by our attribution graphs are highly consistent with those of the actual model behavior.





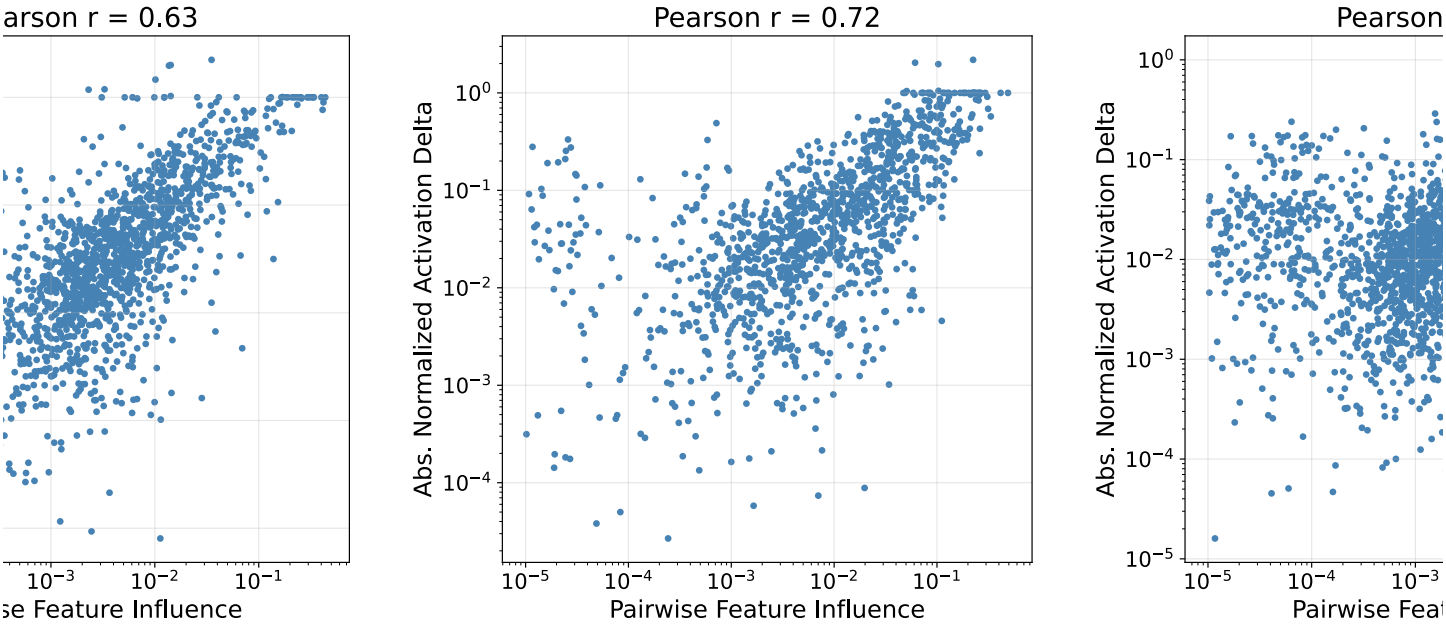
Correlation between predicted influence and actual KL divergence from feature ablation.
irect edge weights are omitted for the CRM, as cross-token transcoder features lack direct edges to the logit

While the previous experiment validates influence at a global level from features to logits, we further examine at a finer granularity whether the indirect influence between individual feature pairs faithfully reflects the mechanisms of the underlying model.Using the same 100 prompts, we construct attribution graphs and identify the 512 features with the greatest influence on the logits in each graph. For each such feature, we ablate it and measure the resulting changes in activation[15] of downstream features within 3 layers, computing the Pearson correlation between these changes and the corresponding indirect influence scores. We observe Pearson correlations of at least 0.62 in CRM across different sparsity settings, indicating that the graph faithfully captures the causal structure of the underlying model not only globally but also at the level of individual feature interactions.

Replacement model type	Sparsity	Pearson Correlation
Transcoder-only Replacement Model	64	0.560
	128	0.587
	256	0.649
Complete Replacement Model	64	0.690
	128	**0.691**
	256	0.628

Figures below show results from three randomly sampled examples, illustrating

how well the pairwise feature influence predicts actual activation changes. Only a few points exhibit low predicted values but high actual values, confirming that the graph-based indirect influence reliably captures the actual effects between features.



Correlation between pairwise feature influence and actual activation changes from feature ablation.

Evaluating Faithfulness of CRM

To assess the overall mechanistic fidelity of the CRM, we introduce perturbations at an upstream layer and compare the resulting downstream effects in the underlying model with those propagated through the CRM. The degree of agreement between these two pathways indicates how faithfully the CRM captures the underlying model's internal mechanisms.

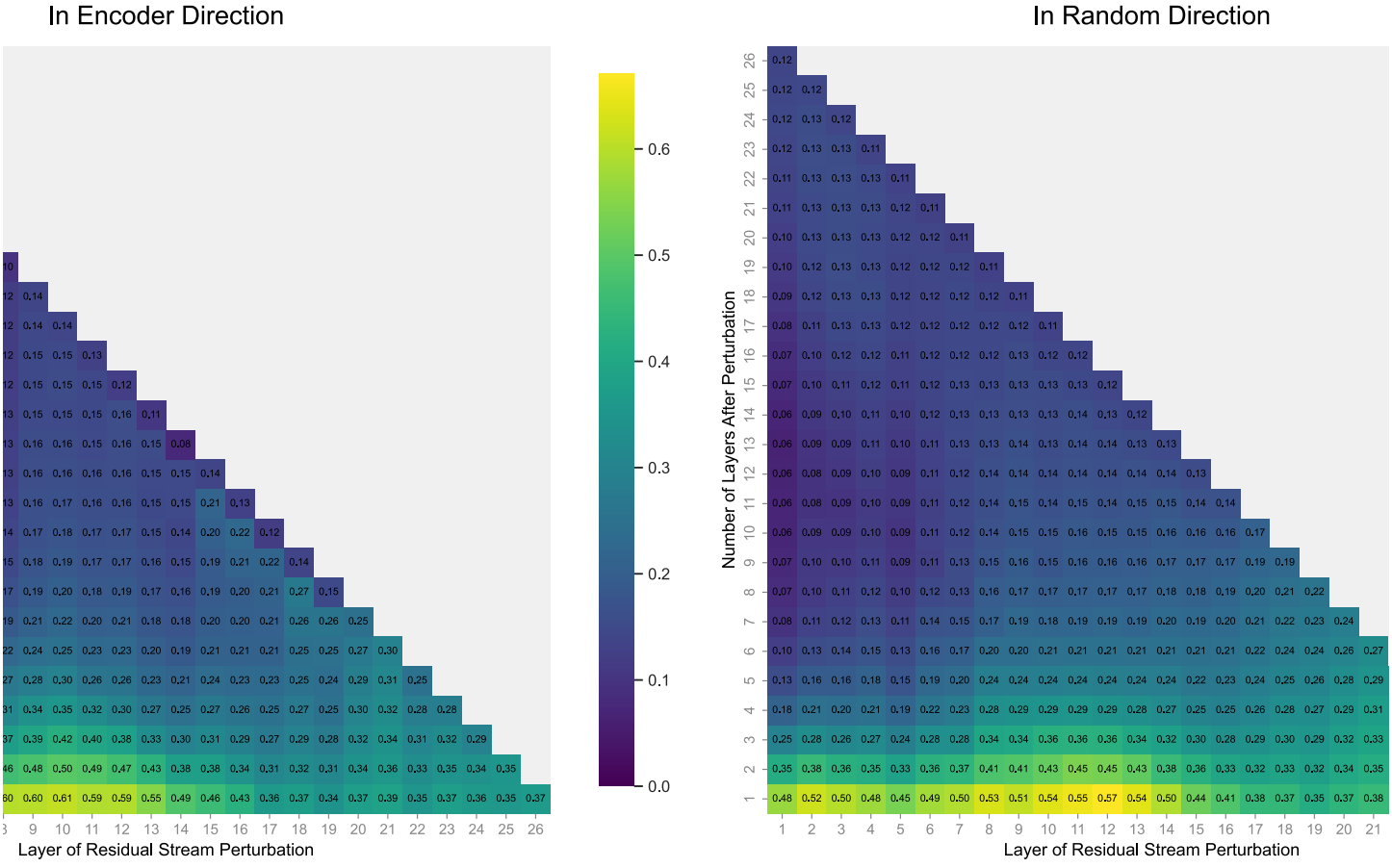
In these experiments, we freeze the error nodes in the replacement models, ensuring that both the CRM and the Transcoder-only replacement model match the underlying model's behavior exactly prior to perturbation[16]. We consider three categories of perturbation vectors applied to the residual stream at a designated intervention layer:

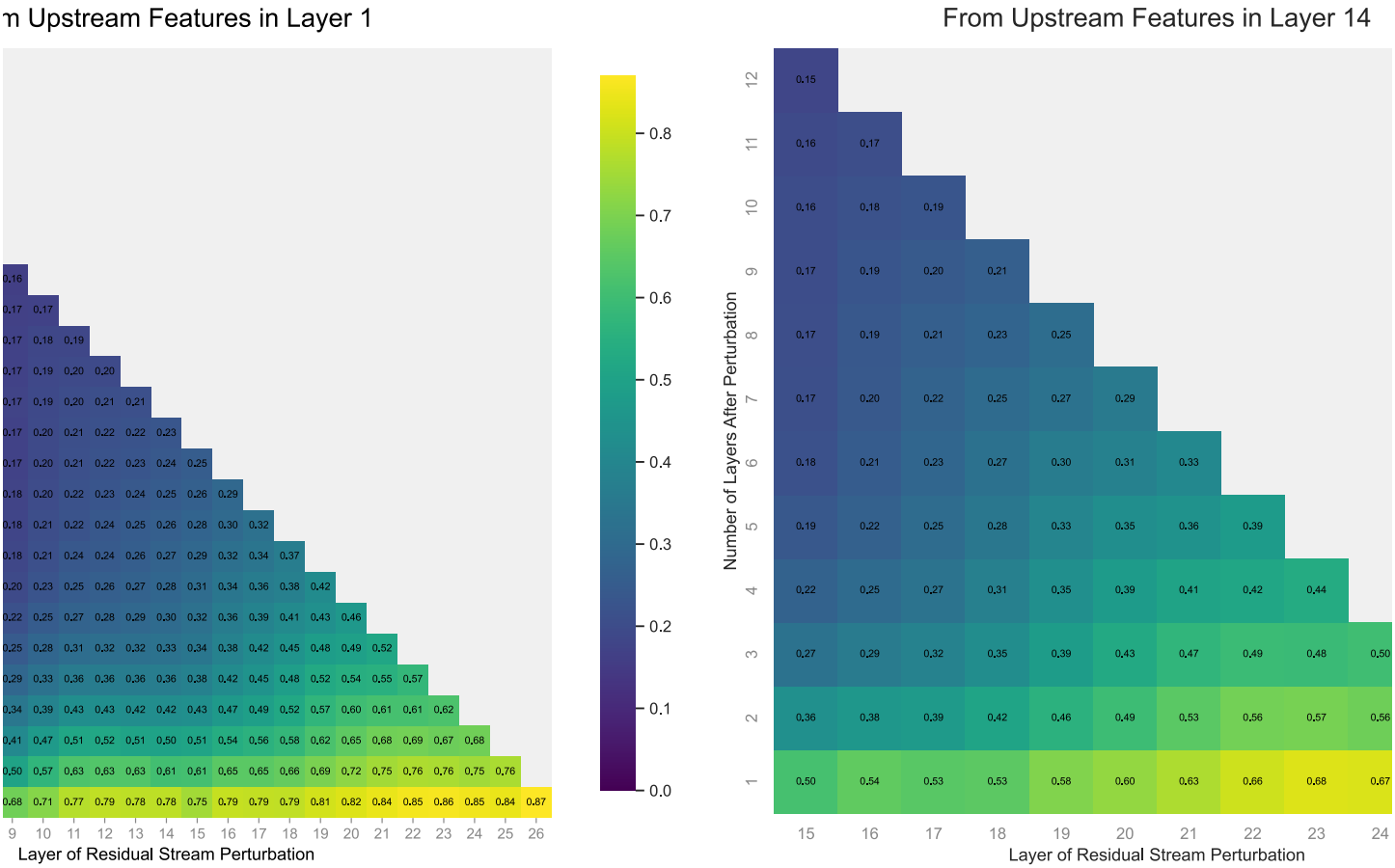
- **Encoder directions:** For an active feature at the intervention layer, we add its encoder vector, scaled to increase the feature's activation by 0.1, to the

residual stream.

- **Random directions:** As a control, we apply a random rotation to the scaled encoder vector above before adding it to the residual stream.
- **Upstream features:** We increase the activation of an active feature at an upstream layer by 0.1 and propagate through the replacement model to the intervention layer. The resulting change in the residual stream serves as the perturbation vector.

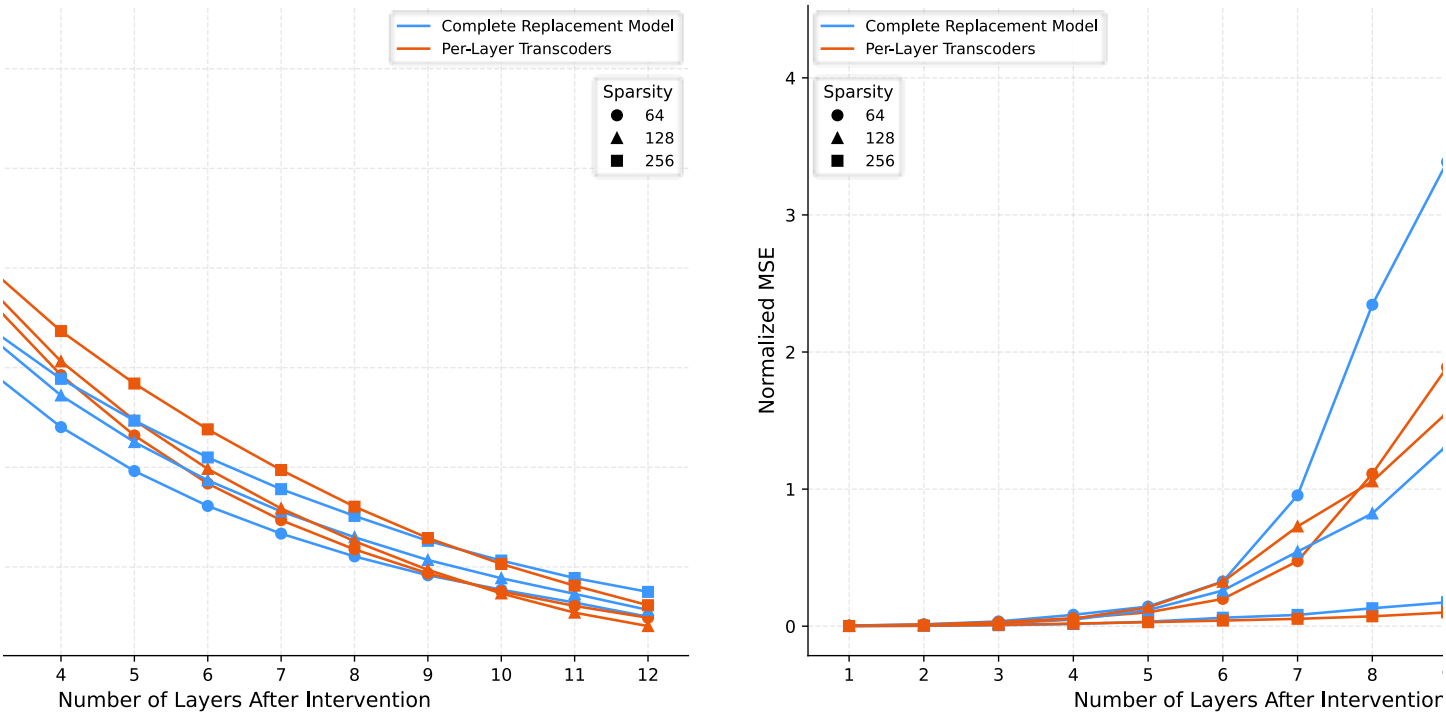
For each perturbation type, we run forward passes through both the underlying model and the replacement model, and at every layer beyond the intervention point, compute the perturbation effect — the difference between the perturbed and baseline residual streams. We then measure the cosine similarity and normalized MSE between the perturbation effects of the two models. For the encoder-direction and random-direction experiments, we sample 512 active features per layer; for the upstream-feature experiments, we sample 512 active features each from layer 1 and layer 14 as perturbation sources.





Cosine similarity of perturbation effects between the CRM(K=64) and the underlying model.

As shown in the bottom row, the CRM(K=64) maintains high cosine similarity across perturbation types and intervention layers. For encoder and random directions, fidelity decreases at higher layers, while for upstream features the trend is reversed, possibly reflecting shifts in the model's internal activation distribution across layers.



Cosine similarity and Normalized MSE of perturbations from upstream features in layer 14.

Overall, the CRM and the Transcoder-only replacement model exhibit comparable perturbation fidelity, indicating that the introduction of Lorsas does not significantly compromise mechanistic faithfulness. The CRM shows slightly lower fidelity near the intervention point, but this gap narrows at downstream layers, likely because downstream Lorsas can capture shifting attention patterns despite some reconstruction error, while the Transcoder-only replacement model relies on frozen attention patterns.

Related Work

Sparse Dictionary Learning

Interpretable units are the fundamental building blocks of mechanistic interpretability and circuit tracing. **Sparse Dictionary Learning** has emerged as a principled approach for decomposing neural network activations into sparse features, with interpretability superior to natural units like neurons and attention heads.

Despite its distant origin[38], sparse dictionary learning has recently gained momentum through the establishment of Superposition Hypothesis[2,39,1], which suggests that a neural network *represents more features than it has neurons*. Building on this hypothesis, Sparse Autoencoders (SAEs)[3,4] have been developed to disentangle sparse features from the natural superposition in the neural network activations. Methodological improvements on SAEs have been continuously made to tackle the unique challenges in sparse training [12,13,14,40]. SAEs have also proven their ability to extract features from various internal locations[41,42,43], from different model architectures[44,45] and from frontier models[12,46]. Still, outstanding limitations and challenges remain in the performance and soundness of SAE-based feature extraction[47,48,49,50,51].

One restriction of SAEs is that they learn features from a transient, isolated point of the model, making it difficult to study the interaction between features and ensure faithfulness. Several architectural variants have been proposed to address this limitation. **Transcoders**[6,7] employ an architecture identical to SAEs, but learn to predict MLP layer outputs from their inputs, bridging over the non-linearity in MLPs. Similarly, **Lorsa**[9] sparsifies MHSA to disentangle attention superposition, allowing sparse feature connections passing through attention layer. Crosscoders[52,53,54] employ multiple parallel encoders and decoders and a unified feature space to resolve superposition among different activation spaces, either cross-layer[52] or cross-model[52,53,54,55]. Cross-layer Transcoders (CLTs)[8] follow an architecture similar to weakly causal crosscoders, but further decouple input and output spaces, allowing for simultaneous sparsification of all MLP computations while tackling cross-layer superposition.

Circuit Discovery

Circuit Discovery methods aim to uncover the causal dependencies between components of the model. Following the definition of Olah et al. (2020)[56], a *circuit* is a subgraph of the model's full computation graph (a directed acyclic graph) that partially represents the model's computation under certain tasks. Nodes of the graph represent model components (e.g. neurons, attention heads, etc.) and edges mediate the influence between them. Recent advances in circuit discovery can be roughly categorized into two categories: improvements on *proper elements serving as nodes (circuit units)* and improvements on *how to find the nodes and edges (tracing methodology)*.

Improvements on Circuit Units

Circuit units are the fundamental building blocks of circuit discovery. Whether the units are understandable for humans is crucial for whether the whole circuits are interpretable. Early circuit discovery methods (often referred to as variants of *attribution* or *information flow*, since they do not provide explicit graph structure) primarily focused on natural units of the neural network. Researchers have attempted to highlight relevant parts of the input (typically image) in CNNs using *saliency maps* before mechanistic interpretability was developed [57,58,59,60,61,62,63], where the units are directly *pixels* or *regions* in the input images. Methodologies then advanced to intermediate model components, including neurons[64,23,65,66] and attention heads[25,27,67], until recently when sparse dictionary learning methods have prevailed and sparse features become the dominant units for circuit discovery[10,37,6,68]. Transcoder features and CLT features soon followed to enable inherent perception of model computation and allow for more robust and input-invariant circuit discovery[7,6,8].

Improvements on Tracing Methodology

Another important direction of circuit discovery concerns how to trace important nodes and compute faithful dependencies between them. Most of the existing

approaches fall into either of the following two groups: intervention-based or attribution-based.

Intervention-based methods intervene on the model, typically by changing the input or intermediate activations, and observe the downstream effects. **Activation Patching** (a.k.a. causal tracing, causal mediation) is the prevalent paradigm[36], which *runs the model on input A, replaces (patches) an activation with that same activation on input B, and sees how much that shifts the answer from A to B*[17]. This paradigm has been widely applied in interpretability studies [69,70,71,72]. Meng et al. (2022)[73] uses similar strategy to locate factual associations, and extends it to also *edit* the internal information to fix mistakes. Wang et al. (2023)[27] develops *Path Patching*, which enables path-based intervention and applies it to the Indirect Object Identification task. Conmy et al. (2023)[74] automates activation patching with a recursive patching procedure.

Attribution-based methods use *attribution scores* to guide the tracing process, which are first-order Taylor expansion terms. Unlike intervention-based methods, attribution-based methods do not require separate forward passes for each interested component, but only a backward pass to obtain all the gradients.

Early interpretability studies on CNNs use saliency maps, whose form is identical to attribution scores. Inspired by patching methods, Nanda (2023)[75] introduces *Attribution Patching*, which still focuses on patching from a clean run to a corrupted run, but uses attribution scores to find important components, with a total of 2 forward passes and 1 backward pass. Syed et al. (2024)[76] finds its performance is superior to automated activation patching methods. Several works follow this paradigm to improve the gradient approximation [77,78,79,80]. Marks et al. (2025)[37] extends attribution patching to SAE features and uses integrated gradients for better approximation. Ge et al. (2024)[6] employs transcoders and separates OV and QK circuits to keep the computational graph linear. Kamath et al. (2025)[30] follows up to trace attentional computation in a more systematic way by checkpointing attention paths. Ameisen et al. (2025)[8] proposes CLT based attribution graphs for a

complete understanding of the model's computation for a single prompt. It is worth noting that when studying linear effects[6,8], the attribution scores are identical to simple input decompositions (i.e. $Wx = \sum_i Wx_i$)

Discussion

Alternative Approaches to Complete Replacement Models

Both CRMs and the checkpointing approach from Kamath et al.[30] address the same challenge: exponentially growing attention-mediated paths between features. Both introduce sparse dictionary learning to add interpretable features to the attribution graph. However, their architectural choices lead to different trade-offs.

Kamath et al. checkpoint attention-mediated paths by training SAEs in the residual stream at each layer and computing gradient attributions between adjacent-layer features. This limits attribution edges to attention paths of length 1. However, this makes the attribution graph non-linear: the computational graph depends on the specific input. Additionally, checkpointing with residual stream SAEs cannot address attention superposition because heads in the underlying model cannot be independently understood.

CRMs replace attention layers directly with Lorsa modules. All computational paths start from a source node, pass through the residual stream (where only layernorm and Lorsa attention patterns are applied), and end at a target node. This maintains a conditionally linear computation graph while directly

addressing attention superposition. Each Lorsa feature has independently interpretable QK and OV circuits.

Dimensionality Collapse in Attention Outputs

Attention outputs exhibit strong low-rank structure: approximately 60% of directions account for 99% of variance[81]. This dimensionality collapse is fundamental to the reconstruction quality and interpretability of attention SAEs and Lorsa layers.

Without initialization strategies that align feature directions with the active subspace of activations, attention replacement layers suffer from high rates of dead features and poor reconstruction. Weak attention replacement layers fail to capture the underlying attention mechanisms, preventing effective circuit tracing.

We conjecture that attention replacement models that ignore this structure accumulate reconstruction error in the residual stream across layers. This leads to attributions dominated by error nodes rather than interpretable features. Kamath et al.[30] observe similar error accumulation in their attention replacement models, though they do not explicitly discuss dimensionality collapse or initialization strategies.

Lorsa Feature Families

Throughout the biology section, we observed related Lorsa features sharing the same QK weight matrix.[18] These feature families implement the same attention pattern but write different information to the residual stream. For example, in the string indexing case, features like `Starts with`, `Say second`, and `End with` all share QK weights but have distinct OV circuits for copying different character positions. Similar groupings appear in the induction and multiple choice examples.

This weight sharing reflects a fundamental architectural choice in Lorsa: QK circuits remain full-rank while OV circuits are reduced to rank-1. Experiments in the original Lorsa paper showed that reducing QK dimension significantly degrades performance. One reason is that rotary position embeddings were trained with a specific head dimension. Reducing QK dimension disrupts these position-dependent transformations, causing the replacement model to lose track of token positions.

More fundamentally, QK patterns represent high-level computation. The attention pattern depends on groups of features that collectively describe a functionality, not on individual feature semantics. In the string indexing example, all four features need to locate index-related positions regardless of which specific character they copy. This makes QK circuits natural candidates for weight sharing: multiple features reuse the same attention mechanism while specializing their output through distinct OV circuits.

In contrast, OV circuits behave like transcoders, performing feature-specific transformations from the key-side residual stream to query-side contributions. These transformations naturally decompose to rank-1 operations where each Lorsa feature reads specific input features and writes to specific output directions. The semantic specificity of OV circuits makes them unsuitable for sharing, while their rank-1 structure keeps the parameter count tractable even with thousands of features per layer.

Architectural Compatibility

The architectural compatibility we maintain for Lorsa layers enables CRMs to adapt to any modern transformer architecture. Most improvements to multi-head self-attention can be directly incorporated into Lorsa, preserving architectural choices such as causal masking, attention scaling, rotary embeddings, grouped query attention, and QK-layernorm. As transformer architectures evolve, our replacement methodology can incorporate these advances without fundamental redesign.

One open question is GLU-based MLPs. We do not use a gate mechanism for transcoders in this work, though gates might better capture the bilinear nature of modern GLU MLPs. However, gates introduce non-linearity that could complicate attribution tracing. To maintain linear attribution graphs, we could detach gradients for gate activations, treating them as constants during attribution computation. We leave this investigation for future work.

"Uninterested" Circuits

Attribution graphs sometimes fail to capture the mechanistic stories we care about. The core problem stems from softmax: when interpreting why a model chooses a particular token, we care about why that logit stands out among candidates, not just its absolute value. However, standard attribution tracing tracks absolute contributions to the target logit, making it difficult to distinguish discriminative signal from shared background.

We observed this in the Craig example. When the model predicts the last character of "Craig" versus "Frank" in confounding contexts, the logit difference between "g" and "k" accounts for only around 30% of the total logit magnitude (logits: g = 36.0, k = 26.2 for "Craig"; g = 27.0, k = 43.0 for "Frank"). The remaining 70% comes from features that boost all plausible character predictions uniformly. Attribution graphs dominated by these shared features obscure the discriminative circuits.

Two potential approaches might help isolate discriminative features for logit prediction. First, we could trace from the target logit vector projected onto the nullspace of a manually designed general feature subspace. This removes shared contributions that apply across all candidate tokens. Second, we could perform graph diffing by computing attribution graphs for both the target token and a confounding alternative, then eliminating shared features and circuits. Both methods require manual design choices about what constitutes "uninteresting" background versus discriminative signal.

These approaches become more complex for QK tracing, where softmax operates on attention scores rather than output logits. RoPE (rotary position embeddings) further complicates the picture: it applies position-dependent transformations to queries and keys, making it difficult to cleanly eliminate bias-bias feature interactions through QK circuits. In RoPE models, bias-bias interactions often implement position-related QK circuits that we cannot simply factor out. Addressing the softmax comparison problem in attention mechanisms likely requires deeper investigation beyond current attribution methods.

Inactive Features and Inhibitory Circuits

A related challenge is understanding inhibitory circuits where features actively suppress other features. Anthropic's recent work on attribution graphs discusses this extensively under "The Role of Inactive Features & Inhibitory Circuits", showing how the absence of expected features can be just as mechanistically important as their presence. CRMs do not directly solve this problem, as attribution graphs still primarily surface active features.

However, global weights offer a potential path forward. When a feature we expect to activate remains inactive, we can inspect its typical upstream contributors via global weights to understand what normally causes it to fire. These global weights capture the average connectivity patterns across many prompts, revealing which features consistently excite or inhibit the target. If the typical positive contributors are also inactive, we recursively trace further upstream through their own global weight connections until we find where the expected activation pathway diverges from its typical pattern. This backward search reveals which early features failed to activate, breaking the chain that would normally lead to our target feature.

Conversely, if the typical upstream contributors are active as expected, we can search for strong negative global weights pointing to the target feature. These inhibitory connections reveal features that actively suppress the target when

present. By examining which of these inhibitory features are active on the current prompt, we can identify the specific suppression mechanism at work. This approach essentially uses global weights as a reference template, allowing us to diagnose deviations from normal activation patterns by comparing what typically happens against what actually occurred on a specific input.

Unexplained Phenomena

The attribution challenges discussed above limit our ability to fully explain certain circuits. When discriminative features are obscured by shared background contributions, or when critical features remain inactive due to inhibition, the resulting attribution graphs may miss key computational pathways. Additionally, reconstruction error can accumulate across layers, creating noise that further obscures the mechanistic story. These limitations manifest in several unexplained phenomena from the biology section.

In the acronym example, after the model identifies "Analytics" and produces "A" for the first letter, why doesn't it continue by attending back to "Analytics" to output "n" for the second letter? Instead, the model appears to encode all necessary letter information in parallel during early processing, but the mechanism by which it selects which letter to output at each position remains unclear.

The string indexing examples raise questions about variable binding. When the model sees `a="Craig"` followed by `assert a[0]=='`, how does it bind the variable name "a" to the string "Craig" and maintain this binding across multiple tokens? The attribution graphs show features that recognize variable names and string literals, but the binding mechanism itself—how the model associates "a" with its value rather than other variables in context—does not surface clearly in the feature-level circuits we observe.

In the multiple choice questions case, we identified features that distinguish correct from incorrect answer options, but the deeper question remains: how

are these features computed? The model must somehow evaluate factual knowledge (e.g., when World War II ended) and compare it against each option's content. While we observe the final discrimination in option-specific features, the computational pathway that encodes and retrieves the relevant factual knowledge before this discrimination occurs is not captured in our graphs. This suggests either that the crucial computation happens in ways our current attribution methods cannot surface, or that important features remain unidentified in our current replacement models.

Open Source Replacement Models

Training replacement models at scale requires substantial computational resources and engineering effort, which suggests the need for open-source replacement models to facilitate reproducibility and collaboration.

The mechanistic interpretability community has made significant progress in openly releasing trained sparse dictionaries. Our team previously released Llama Scope[41], extracting millions of SAE and transcoder features from Llama-3.1-8B. Google DeepMind released GemmaScope[42], training SAEs and transcoders across multiple Gemma 2 model sizes and layers. More recently, GemmaScope 2[43] extended this work with cross-layer transcoders (CLTs) and per-layer transcoders (PLTs), placing greater emphasis on circuit tracing capabilities.

We plan to release Llama Scope 2, focusing specifically on complete replacement models for circuit tracing. Our initial release will include open-source Lorsas and transcoders trained on Qwen3-1.7B with the configurations presented in this work, with more models supported in the future. All models and resources will be publicly available at OpenMOSS-Team/Llama-Scope-2. A production-ready version of our training and analyzing codebase, with documentation and tutorials, is currently being finalized and will be released alongside.

Cross Layer Replacement Models

Cross-layer transcoders[8] have demonstrated that features can exist in superposition across multiple MLP layers. We suspect that this cross-layer superposition also occurs in attention layers and between different layer types. Features may be jointly represented across both attention outputs and MLP outputs. If this hypothesis holds, it would be beneficial to design replacement models where both Lorsa and transcoder features can write to all downstream attention and MLP outputs, creating a unified feature space that resolves superposition across computational blocks.

However, unrestricted cross-layer connections introduce a quadratic explosion in the number of possible feature interactions. For a model with L layers and F features per layer, allowing arbitrary cross-layer connections yields $O(L^2F)$ potential edges, making both training and analysis computationally intractable. A more practical approach is to limit cross-layer connections to a fixed window of adjacent layers (we might call these *short-range* or *windowed* cross-layer connections). This design choice is motivated by the observation that most cross-layer superposition appears concentrated in nearby layers rather than spanning the entire residual stream. By restricting each feature to write only to the next few layers, we can balance the benefits of resolving cross-layer superposition with computational efficiency.

Exploring these architectural variants remains an important direction for future work. The right balance between expressiveness and scalability will depend on empirical measurements of how far cross-layer features typically extend and whether the added complexity improves circuit tracing quality.

Appendices

Notation

We summarize the key mathematical notation used throughout this work.

Model Architecture

Notation	Description
d	Model dimension (residual stream dimension)
d_h	Attention head dimension
l	Sequence length
F	Feature dimension (dictionary size), $F \gg d$
K	Top-K sparsity parameter

Transcoder Notation

Notation	Description
$\mathbf{W}_{\text{enc}} \in \mathbb{R}^{F \times d}$	Transcoder encoder weight matrix
$\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times F}$	Transcoder decoder weight matrix
$\mathbf{a} \in \mathbb{R}^F$	Transcoder feature activations (sparse)
a_s	Activation of transcoder feature s

Lorsa (Low-Rank Sparse Attention) Notation

Notation	Description
$\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d \times d_h}$	Query and Key weight matrices
$\mathbf{w}_V, \mathbf{w}_{V,s} \in \mathbb{R}^d$	Value weight vector for a Lorsa head s
$\mathbf{w}_O, \mathbf{w}_{O,s} \in \mathbb{R}^d$	Output weight vector for a Lorsa head s
$\mathbf{P} \in \mathbb{R}^{l \times l}$	Attention pattern matrix (after softmax)
$P^t_{i,j}$	Attention pattern from position i to j for lorsa head t
$\mathbf{z} \in \mathbb{R}^l$	Lorsa head activation (z-pattern) across positions

Attribution and Circuit Notation

Notation	Description
$\mathbf{w}_{\text{dec},s}$	Generic decoder vector for source node s
$\mathbf{w}_{\text{enc},t}$	Generic encoder vector for target node t
$\mathbf{w}_{\text{unembed},t}$	Row vector of unembedding matrix for token t
$\mathbf{w}_{\text{embed},s}$	Column vector of embedding matrix for token s
ϵ_s	Error vector for replacement layer s
$A_{s \to t}$	Attribution from target node t to source node s
$\Omega_{s \to t}$	Virtual weight (residual-direct weight) from s to t
$\mathbb{W}_{s \to t}$	Global weight (expected attribution) from s to t

Notation Context Rules

- Bold notation ($\mathbf{W}, \mathbf{w}$) denotes matrices and vectors. Non-bold notation (a_*, P_*) denotes scalars.
- Uppercase ($\mathbf{W}$) denotes matrices. Lowercase ($\mathbf{w}$) denotes vectors.
- **Generic notation** ($\mathbf{w}_{\text{dec},s}, \mathbf{w}_{\text{enc},t}$) refers abstractly to decoder/encoder vectors that

could be from transcoders, Lorsa features, or embeddings depending on context.

Author Contribution

- Wentao Shu trained the replacement layers, with assistance from Junxuan Wang on sparse kernel acceleration.
- Zhengfu He implemented circuit tracing with the complete replacement model, with assistance from Guancheng Zhou, Xuyang Ge, Junxuan Wang and Rui Lin on bug fixes and efficiency improvements.
- Wentao Shu developed the evaluation framework and analyzed the results.
- Wentao Shu and Guancheng Zhou implemented feature intervention experiments.
- Xuyang Ge developed the interactive visualization of features and the attribution graphs.
- Zhengfu He developed the global weight analysis, with assistance from Xuyang Ge and Zhaoxuan Song on visualization and implementation.
- Zhengfu He, Xuyang Ge, Wentao Shu, Guancheng Zhou, Rui Lin, and Jiaxing Wu investigated the biology cases.
- Xuyang Ge leads the development and maintenance of the library for sparse dictionary learning and circuit tracing, with assistance from all other contributors.
- Paper writing was done by the main contributors of each section. Zhengfu He and Wentao Shu did the overall coordination and revision.
- Xuyang Ge and Zhengfu He designed and developed the interactive plots in paper writing.
- Xipeng Qiu supervised the project and provided high-level feedback on paper writing.

Citation Information

Please cite this work as:

```
@article{shu2026completereplacement,
  author={Shu, Wentao and Ge, Xuyang and Zhou, Guancheng and Wang, Junxuan and Lin, Rui and
Song, Zhaoxuan and Wu, Jiaxing and He, Zhengfu and Qiu, Xipeng},
  title={Bridging the Attention Gap: Complete Replacement Models for Complete Circuit
Tracing},
  year={2026},
  journal={OpenMOSS Interpretability Research},
  url={https://interp.open-moss.com/posts/complete-replacement}
```

}

Dataset

The replacement layers were trained on a 9:1 mixture of pretraining and chat data. The chat portion consists of Hermes 3 SFT data[82]. The pretraining portion comprises 60% English from FineWeb-Edu[83], 30% Chinese from CCI3-HQ[84], and 10% code from StarCoder data[85]. All text samples are truncated to 2048 tokens.

Learning Rate

We find that the optimal learning rate generalizes well across layers, and thus determine it via a sweep on a single intermediate layer (Layer 13) over 8B tokens. The rate is further verified to remain robust across varying K settings. Training follows a linear schedule with 5,000 warmup steps and a learning rate cooldown over the final 20% of training to accelerate convergence.

Initialization

To accelerate convergence and promote effective exploration of the feature space, we adopt several initialization strategies.

Pre-trained Weight Inheritance and Alignment

- **Transcoders:** We utilize the original MLP to initialize the transcoder weights. Specifically, for each latent feature, we construct a synthetic input by perturbing the centroid of the calibration batch along the unit direction of the corresponding encoder weight vector[19], with the perturbation magnitude set to the average Euclidean distance of the calibration samples from the centroid. These perturbed inputs are then passed through the original MLP, and the resulting outputs are mean-centered to isolate each feature's directional response relative to the output centroid. The decoder weights $\mathbf{W}_{\text{dec}}$ are then initialized to these mean-centered outputs after normalization [20], ensuring that each decoder column is aligned with the direction of MLP response for the corresponding latent feature.
- **Lorsa:** We follow the initialization procedure proposed by Wang et al.[81]. When training a Lorsa module to approximate a target MHSA layer, we partition the Lorsa

heads into groups whose count matches the number of attention heads in the original MHSA. For each group, Q/K weights are initialized by duplicating the original MHSA Q/K parameters. The encoder (submatrix of $\mathbf{w}_V$) and decoder (submatrix of $\mathbf{w}_O$) weights are initialized directly within the corresponding input and output active subspaces of the MHSA head it corresponds to. All encoder and decoder weights are then row-normalized[21]. The detailed OV initialization procedure is given in the following pseudocode:

Pseudocode 1: Lorsa OV Initialization

```
# Input:
#   X: input activations [b, s, d]  (b=batch, s=seq_len, d=d_model)
#   mhsa: pretrained MHSA module
#     mhsa.W_V: [n, d, h]  (n=n_heads, h=d_head, n*h=d)
#     mhsa.W_O: [n, h, d]
# Lorsa parameters to initialize:
#   W_V: [n_lorsa, d]  (n_lorsa = number of Lorsa heads)
#   W_O: [n_lorsa, d]
# d_qk: Lorsa head dimension for initialization

# 1. Compute per-head V projections
X_flat = X.reshape(b*s, d)                          # [b*s, d]
W_V_cat = mhsa.W_V.permute(1,0,2).reshape(d, d)     # [d, d]
V_per_head = (X_flat @ W_V_cat).reshape(b*s, n, h)  # [b*s, n, h]

# 2. Project V back to d_model space for each head
# captured_v[:, i, :] = V_per_head[:, i, :] @ mhsa.W_V[i].T
captured_v = einsum('bnh, nhd -> bnd',
                    V_per_head, mhsa.W_V.permute(0,2,1))
# captured_v: [b*s, n, d]

# 3. Initialize Lorsa heads from each original head's active subspace
rate = n_lorsa // n
for i in range(n):
    slice_i = [rate*i : rate*(i+1)]

    # 3.1 Extract this head's captured V
    v = captured_v[:, i, :]             # [b*s, d]

    # 3.2 Demean
    demeaned_v = v - v.mean(dim=0)      # [b*s, d]

    # 3.3 SVD on transposed data to get principal directions
    U, S, _ = svd(demeaned_v.T)         # U: [d, d]

    # 3.4 Take top-d_qk principal directions as projection
    proj = U[:, :d_qk]                  # [d, d_qk]
```

```
        # 3.5 Update W_V: project into principal subspace
13      W_V[slice_i] = W_V[slice_i, :d_qk] @ proj.T     # [rate, d]

        # 3.6 Update W_O: chain W_V through original head's OV circuit
        # OV_i = mhsa.W_V[i] @ mhsa.W_O[i]: [d,h] @ [h,d] = [d,d]
14      W_O[slice_i] = W_V[slice_i] @ mhsa.W_V[i] @ mhsa.W_O[i]
        # [rate, d] @ [d, h] @ [h, d] = [rate, d]

    # 4. Normalize all Lorsa weights (row-wise)
15  W_V = W_V / W_V.norm(dim=1, keepdim=True)    # [n_lorsa, d]
16  W_O = W_O / W_O.norm(dim=1, keepdim=True)    # [n_lorsa, d]
```

Bias Initialization

Proper bias initialization is critical for mitigating dead features and minimizing initial reconstruction error. The central principle underlying our bias initialization is to absorb the systematic offset present in the input and output data distributions.

- **Encoder Bias:** To mitigate the risk of feature collapse at the onset of training, we initialize $\mathbf{b}_{\text{enc}}$ by centering the pre-activation distribution around zero. Specifically, we perform a forward pass on a calibration batch through the encoder's linear transformation, obtaining the pre-activation values prior to the application of the sparsity-inducing activation function. The encoder bias is then adjusted by subtracting the empirical mean of these pre-activations. This centering operation ensures that each latent feature receives positive pre-activation signals on approximately half of the calibration samples, preventing premature feature death and promoting more uniform feature utilization during the early stages of optimization. A similar initialization idea has been adopted in prior work[86].
- **Decoder Bias:** We initialize $\mathbf{b}_{\text{dec}}$ to the empirical mean of the reconstruction target whose mean squared error is to be minimized. This places the initial reconstruction output in the vicinity of the target distribution from the very first forward pass, substantially reducing the initial reconstruction error and providing a more favorable starting point for subsequent optimization.

Norm Adaptation via Grid Search

The scale of the initialized decoder weights significantly affects training dynamics. We determine the optimal initialization norm via grid search, finding a global scalar factor λ^* to scale the decoder weights $\mathbf{W}_{\text{dec}}$, minimizing the reconstruction error:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \sum_{\mathbf{x} \in \text{batch}} \|\mathbf{x} - \text{Reconstruct}(\mathbf{x}; \lambda \mathbf{W}_{\text{dec}})\|_2^2.$$

Circuit Tracing

Backward-pass attribution. We compute the attribution graph via gradient-based backward propagation[8], which is equivalent to the forward-perspective formulation in the main text.As in Transcoder-only models, we freeze LayerNorm scaling denominators and attention patterns so that the residual stream mapping between any two layers reduces to a linear function whose Jacobian can be computed exactly via a single backward pass, and then trace QK circuits on top of the resulting OV and MLP attribution graph[22]. For a given target feature t in the CRM, we inject its encoder vector $\mathbf{w}_{\text{enc},t}$ into the residual stream at the layer where t reads, propagate backward through the frozen model, and read off the edge weight from each upstream source node s as $a_s \mathbf{w}_{\text{dec},s}^\top \mathbf{g}$, where $\mathbf{g}$ is the back-propagated gradient at the corresponding layer and position. By the linearity of the frozen model, this yields precisely $A_{s \to t} = a_s P_{i,j}^t \Omega_{s \to t}$ as given in the main text.

Iterative expansion. The tracing begins at the logit nodes. Since a logit node reads from a single sequence position, backward propagation from it reaches only features active at that position—Transcoder features and Lorsa features whose decoders write there. When the frontier reaches a Lorsa feature node, the backward pass propagates through its $\mathbf{W}_V$ projection and the associated attention pattern, distributing gradient to upstream positions attended to by that feature. This traces cross-position influence to upstream Transcoder features, Lorsa features, token embeddings, and error nodes. Each newly traced feature is then available as a source node for further upstream iteration.

Greedy node selection. A single prompt may activate tens of thousands of features, making exhaustive computation prohibitive. We therefore adopt a greedy strategy: at each iteration, we select the source node with the greatest estimated influence on the output and compute its incoming edges. After each batch, the influence estimates are updated with the newly discovered edges to refine the priority ordering. By imposing a fixed node budget, this procedure allocates computation to the most informative subgraph while controlling cost.

Graph Pruning

We follow the graph pruning procedure of Ameisen et al.[8] to obtain a sparse, interpretable computational graph. Starting from either the full attribution graph or an approximate subgraph obtained via greedy expansion, we compute logit influence scores for each node by taking the probability-weighted average of the corresponding rows in the normalized unsigned adjacency matrix over all logit nodes. Non-logit nodes are then sorted by descending logit influence and retained up to a cumulative threshold τ_{node} of the total influence mass. Edge-level pruning is performed analogously: after recomputing normalized adjacency weights on the pruned graph, each edge is assigned an influence score equal to the product of its destination node's logit influence and its normalized weight, and edges are retained up to a cumulative threshold τ_{edge}. Logit nodes are pruned separately by retaining the top-K most probable output tokens such that their cumulative probability exceeds 0.95, clamped to $K \leq 10$. Embedding and error nodes are exempt from pruning.

Footnotes

1. Except for layernorms, embedding and unembedding.

2. If an SAE feature is computed by multiple attention heads, we must interpret their attention patterns jointly to understand the QK circuit. The QK and OV framework was introduced in *A Mathematical Framework of Transformer Circuits*[23].

3. We believe cross-layer connection is useful in the long run, with a large space for improvement. See more discussions in §9.9 Cross Layer Replacement Models.

4. See *A Mathematical Framework of Transformer Circuits*[23] for analysis of attention computation decomposed to QK and OV circuits.

5. Strictly speaking, this is only true under the strong superposition hypothesis[87] assuming model activations and computations can be described with only rank-1 linear features. A surging line of research on "continuous features" or "feature manifolds"[88,89,87] attempts to find a unified description of 1-dimensionally linear and more general continuous features. For now, we only consider the 1-dimensionally linear case. This can still capture some signal of feature manifolds, but in a less graceful way.

6. Equivalently, we apply stop-gradient at these tensors.

7. Note that these paths represent *direct attribution* and thus do not pass through any other nodes. Indirect influence from one node to another handled by one or more intermediate nodes are described as multi-step paths in the graph.

8. In a transcoder-only replacement model, $w_{s \to t}$ contains both residual-direct and attention-mediated paths, which suffer from the problem of exponential growth of possible paths. These

long and uninterpretable paths are now cut to short paths by Lorsa layers in a CRM.

9. "The National Digital Analytics Security Group (ND", prediction: AS (correct)
"The National Digital Analytics Intelligence Group (ND", prediction: AG (incorrect)
"The National Digital Analytics Advisory Group (ND", prediction: AG (incorrect)
"The National Digital Analytics Solutions Group (ND", prediction: AS (correct)
"The National Digital Analytics Alliance Group (ND", prediction: AA (correct)
"The National Digital Analytics Strategy Group (ND", prediction: AS (correct)
"The National Digital Analytics Insight Group (ND", prediction: AG (incorrect)
"The National Digital Analytics Council Group (ND", prediction: AC (correct)
"The National Digital Analytics Innovation Group (ND", prediction: AG (incorrect)
"The National Digital Analytics Research Group (ND", prediction: ARG (correct)

10. "The National Digital Intelligence Bureau (ND", prediction: IB (correct)
"The National Digital Data Institute (ND", prediction: I (incorrect)
"The National Digital Metrics Council (ND", prediction: MC (correct)
"The National Digital Insights Agency (ND", prediction: IA (correct)
"The National Digital Information Center (ND", prediction: IC (correct)
"The National Digital Surveillance Office (ND", prediction: SO (correct)
"The National Digital Strategy Unit (ND", prediction: SU (correct)
"The National Digital Measurement Authority (ND", prediction: MA (correct)
"The National Digital Forensics Lab (ND", prediction: FL (correct)
"The National Digital Trends Observatory (ND", prediction: TO (correct)

11. From the model's perspective, this is an efficient strategy: when the task requires retrieving a character, capitalized words are often good candidates.

12. Similar to the string indexing case above, induction paths might be used to distinguish between different possible names if there are multiple in the context.

13. In §4 Building Complete Attribution Graphs, we denote the direct contribution from node i to node j as $A_{i\to j}$. To simplify matrix multiplication, we hereafter rename it to A_{ji}.

14. Following a similar methodology as in Ameisen et al.[8], we select 100 sequences of 20–50 tokens from SlimPajama that satisfy three criteria: (1) the underlying model correctly predicts the top-1 token, (2) the loss at the current token exceeds 0.2, and (3) the dataset density of the current token is below 1×10^{-4}.

15. We normalize changes by original activation values and take absolute values, as influence scores are unsigned.

16. For the Transcoder-only replacement model, we additionally freeze the attention patterns; this is unnecessary for the CRM, as the downstream Lorsas capture the effect of attention on the perturbation.

17. Explanation given by Neel Nanda here.

18. We can identify this from their feature indices, where QK index = floor(feature id / 4096). Features with the same QK index share identical query and key weight matrices but have distinct

OV circuits.

19. The encoder weight $\mathbf{W}_{\text{enc}}$ is randomly initialized uniformly within the range of $[-\frac{1}{n_{\text{features}}}, \frac{1}{n_{\text{features}}}]$.

20. Since the subsequent Norm Adaptation via Grid Search will calibrate the overall scale, only the directional information is important at this stage.

21. As with the transcoder decoder, only directional information matters here; the subsequent Norm Adaptation via Grid Search calibrates the overall scale.

22. The introduction of Lorsa makes the resulting QK circuits considerably more interpretable. See §5.2 Induction for more details.

References

1. **Linear Algebraic Structure of Word Senses, with Applications to Polysemy** [link]
   S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, 2018. Trans. Assoc. Comput. Linguistics.

2. **Toy Models of Superposition** [link]
   N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah, 2022. Transformer Circuits Thread.

3. **Sparse autoencoders find highly interpretable features in language models** [link]
   H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey, 2023. arXiv preprint arXiv:2309.08600.

4. **Towards Monosemanticity: Decomposing Language Models With Dictionary Learning**
   T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah, 2023. Transformer Circuits Thread.

5. **Circuits Updates - January 2024** [link]
   A. Templeton, J. Batson, A. Jermyn, and C. Olah, 2024. Transformer Circuits Thread.

6. **Automatically Identifying Local and Global Circuits with Linear Computation Graphs** [link]
   X. Ge, F. Zhu, W. Shu, J. Wang, Z. He, and X. Qiu, 2024. CoRR.

7. **Transcoders Find Interpretable LLM Feature Circuits** [link]
   J. Dunefsky, P. Chlenski, and N. Nanda, 2024. CoRR.

8. **Circuit Tracing: Revealing Computational Graphs in Language Models** [link]
   E. Ameisen, J. Lindsey, A. Pearce, W. Gurnee, N. Turner, B. Chen, C. Citro, D. Abrahams, S. Carter, B. Hosmer, J. Marcus, M. Sklar, A. Templeton, T. Bricken, C. McDougall, H. Cunningham, T.

Henighan, A. Jermyn, A. Jones, A. Persic, Z. Qi, T. Thompson, S. Zimmerman, K. Rivoire, T. Conerly, C. Olah, and J. Batson, 2025. Transformer Circuits Thread.

9. Towards Understanding the Nature of Attention with Low-Rank Sparse Decomposition [link]
   Z. He, J. Wang, R. Lin, X. Ge, W. Shu, Q. Tang, J. Zhang, and X. Qiu, 2025. CoRR.

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

11. Interpreting Attention Layer Outputs with Sparse Autoencoders [link]
    C. Kissane, R. Krzyzanowski, J. Bloom, A. Conmy, and N. Nanda, 2024. CoRR.

12. Scaling and evaluating sparse autoencoders [link]
    L. Gao, T. Tour, H. Tillman, G. Goh, R. Troll, A. Radford, I. Sutskever, J. Leike, and J. Wu, 2024. CoRR.

13. BatchTopK Sparse Autoencoders [link]
    B. Bussmann, P. Leask, and N. Nanda, 2024. arXiv preprint arXiv:2412.06410.

14. Jumping Ahead: Improving Reconstruction Fidelity with JumpReLU Sparse Autoencoders [link]
    S. Rajamanoharan, T. Lieberum, N. Sonnerat, A. Conmy, V. Varma, J. Kram'ar, and N. Nanda, 2024. arXiv preprint arXiv:2407.14435.

15. The Circuits Research Landscape: Results and Perspectives [link]
    J. Lindsey, E. Ameisen, N. Nanda, S. Shabalin, M. Piotrowski, T. McGrath, M. Hanna, O. Lewis, C. Tigges, J. Merullo, C. Watts, G. Paulo, J. Batson, L. Gorton, E. Simon, M. Loeffler, C. McDougall, and J. Lin, 2025. Neuronpedia.

16. Circuits Updates - May 2023 [link]
    A. Jermyn, C. Olah, and T. Henighan, 2023. Transformer Circuits Thread.

17. Circuits Updates - January 2024 [link]
    A. Jermyn, C. Olah, and T. Conerly, 2024. Transformer Circuits Thread.

18. Attention is All you Need
    A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, 2017. NIPS.

19. RoFormer: Enhanced Transformer with Rotary Position Embedding [link]
    J. Su, Y. Lu, S. Pan, B. Wen, and Y. Liu, 2021. CoRR.

20. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints [link]
    J. Ainslie, J. Lee-Thorp, M. Jong, Y. Zemlyanskiy, F. Lebr'on, and S. Sanghai, 2023. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023.

21. Query-Key Normalization for Transformers

A. Henry, P. Dachapally, S. Pawar, and Y. Chen, 2020. EMNLP (Findings).

22. Qwen3 Technical Report [link]

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, 2025. CoRR.

23. A Mathematical Framework for Transformer Circuits

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah, 2021. Transformer Circuits Thread.

24. Sparse Crosscoders for Cross-Layer Features and Model Diffing [link]

J. Lindsey, A. Templeton, J. Marcus, T. Conerly, J. Batson, and C. Olah, 2024. Transformer Circuits Thread.

25. In-context Learning and Induction Heads [link]

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

26. Successor Heads: Recurring, Interpretable Attention Heads In The Wild [link]

R. Gould, E. Ong, G. Ogden, and A. Conmy, 2024. The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024.

27. Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 Small [link]

K. Wang, A. Variengien, A. Conmy, B. Shlegeris, and J. Steinhardt, 2023. The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.

28. Does Circuit Analysis Interpretability Scale? Evidence from Multiple Choice Capabilities in Chinchilla [link]

T. Lieberum, M. Rahtz, J. Kram'ar, N. Nanda, G. Shah, and V. Mikulik, 2023. CoRR.

29. We Inspected Every Head in GPT-2 Small Using SAEs So You Don't Have To [link]

R. Krzyzanowski, C. Kissane, A. Conmy, and N. Nanda, 2024. .

30. Tracing Attention Computation Through Feature Interactions [link]

H. Kamath, E. Ameisen, I. Kauvar, R. Luger, W. Gurnee, A. Pearce, S. Zimmerman, J. Batson, T. Conerly, C. Olah, and J. Lindsey, 2025. Transformer Circuits Thread.

31. A Toy Model of Interference Weights [link]

C. Olah, N. Turner, and T. Conerly, 2025. Transformer Circuits Thread.

32. Confidence Regulation Neurons in Language Models [link]
A. Stolfo, B. Wu, W. Gurnee, Y. Belinkov, X. Song, M. Sachan, and N. Nanda, 2024. Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.

33. Interpretability Dreams [link]
C. Olah, 2023. Transformer Circuits Thread.

34. Adversarial Examples Are Not Bugs, They Are Superposition [link]
L. Gorton and O. Lewis, 2025. CoRR.

35. Direct and indirect effects
J. Pearl, 2022. Probabilistic and causal inference: the works of Judea Pearl.

36. Investigating gender bias in language models using causal mediation analysis
J. Vig, S. Gehrmann, Y. Belinkov, S. Qian, D. Nevo, Y. Singer, and S. Shieber, 2020. Advances in neural information processing systems.

37. Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models
S. Marks, C. Rager, E. Michaud, Y. Belinkov, D. Bau, and A. Mueller, 2025. ICLR.

38. Sparse coding with an overcomplete basis set: A strategy employed by V1? [link]
B. Olshausen and D. Field, 1997. Vision Research.

39. Distributed Representations: Composition & Superposition [link]
C. Olah, 2023. Transformer Circuits Thread.

40. Improving Dictionary Learning with Gated Sparse Autoencoders [link]
S. Rajamanoharan, A. Conmy, L. Smith, T. Lieberum, V. Varma, J. Kram'ar, R. Shah, and N. Nanda, 2024. Advances in Neural Information Processing Systems.

41. Llama scope: Extracting millions of features from llama-3.1-8b with sparse autoencoders [link]
Z. He, W. Shu, X. Ge, L. Chen, J. Wang, Y. Zhou, F. Liu, Q. Guo, X. Huang, Z. Wu, and others, 2024. arXiv preprint arXiv:2410.20526.

42. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2 [link]
T. Lieberum, S. Rajamanoharan, A. Conmy, L. Smith, N. Sonnerat, V. Varma, J. Kram'ar, A. Dragan, R. Shah, and N. Nanda, 2024. arXiv preprint arXiv:2408.05147.

43. Gemma Scope 2: helping the AI safety community deepen understanding of complex language model behavior [link]
C. McDougall, A. Conmy, J. Kram'ar, T. Lieberum, S. Rajamanoharan, and N. Nanda, 2025. Google DeepMind.

44. Towards Universality: Studying Mechanistic Similarity Across Language Model Architectures

J. Wang, X. Ge, W. Shu, Q. Tang, Y. Zhou, Z. He, and X. Qiu, 2025. ICLR 2025.

45. Steering CLIP's vision transformer with sparse autoencoders [link]
S. Joseph, P. Suresh, E. Goldfarb, L. Hufe, Y. Gandelsman, R. Graham, D. Bzdok, W. Samek, and B. Richards, 2025. CVPR 2025 Workshop on Mechanistic Interpretability for Vision.

46. Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet [link]
A. Templeton, T. Conerly, J. Marcus, J. Lindsey, T. Bricken, B. Chen, A. Pearce, C. Citro, E. Ameisen, A. Jones, H. Cunningham, N. Turner, C. McDougall, M. MacDiarmid, A. Tamkin, E. Durmus, T. Hume, F. Mosconi, C. Freeman, T. Sumers, E. Rees, J. Batson, A. Jermyn, S. Carter, C. Olah, and T. Henighan, 2024. Transformer Circuits Thread.

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

48. A is for Absorption: Studying Feature Splitting and Absorption in Sparse Autoencoders [link]
D. Chanin, J. Wilken-Smith, T. Dulka, H. Bhatnagar, and J. Bloom, 2024. CoRR.

49. Sparse Autoencoders Do Not Find Canonical Units of Analysis [link]
P. Leask, B. Bussmann, M. Pearce, J. Bloom, C. Tigges, N. Moubayed, L. Sharkey, and N. Nanda, 2025. The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025.

50. Sparse Autoencoders Trained on the Same Data Learn Different Features [link]
G. Paulo and N. Belrose, 2025. CoRR.

51. Sparse Crosscoders for Cross-Layer Features and Model Diffing [link]
J. Lindsey, A. Templeton, J. Marcus, T. Conerly, J. Batson, and C. Olah, 2024. Transformer Circuits Thread.

52. Overcoming Sparsity Artifacts in Crosscoders to Interpret Chat-Tuning [link]
J. Minder, C. Dumas, C. Juang, B. Chugtai, and N. Nanda, 2025. .

53. Evolution of Concepts in Language Model Pre-Training [link]
X. Ge, W. Shu, J. Wu, Y. Zhou, Z. He, and X. Qiu, 2025. .

54. Insights on Crosscoder Model Diffing [link]
S. Mishra-Sharma, T. Bricken, J. Lindsey, A. Jermyn, J. Marcus, K. Rivoire, C. Olah, and T. Henighan, 2025. Transformer Circuits Thread.

55. Zoom In: An Introduction to Circuits [link]
C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter, 2020. Distill.

56. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps [link]
K. Simonyan, A. Vedaldi, and A. Zisserman, 2014. 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings.

57. **Striving for Simplicity: The All Convolutional Net** [link]
J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, 2015. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings.

58. **Visualizing and Understanding Convolutional Networks** [link]
M. Zeiler and R. Fergus, 2014. Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I.

59. **Learning how to explain neural networks: PatternNet and PatternAttribution** [link]
P. Kindermans, K. Schutt, M. Alber, K. Muller, D. Erhan, B. Kim, and S. Dahne, 2018. 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.

60. **Axiomatic Attribution for Deep Networks** [link]
M. Sundararajan, A. Taly, and Q. Yan, 2017. Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.

61. **Visualizing Deep Neural Network Decisions: Prediction Difference Analysis** [link]
L. Zintgraf, T. Cohen, T. Adel, and M. Welling, 2017. 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.

62. **Sanity checks for saliency maps**
J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, 2026. Advances in neural information processing systems.

63. **Curve detectors**
N. Cammarata, G. Goh, S. Carter, L. Schubert, M. Petrov, and C. Olah, 2026. Distill.

64. **Progress measures for grokking via mechanistic interpretability** [link]
N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt, 2023. The Eleventh International Conference on Learning Representations.

65. **Language Model Circuits are Sparse in the Neuron Basis**
A. Arora, Z. Wu, J. Steinhardt, and S. Schwettmann, 2025. .

66. **Information Flow Routes: Automatically Interpreting Language Models at Scale** [link]
J. Ferrando and E. Voita, 2024. .

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

68. **Discovering the compositional structure of vector representations with role learning networks**
P. Soulos, R. McCoy, T. Linzen, and P. Smolensky, 2020. Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP.

69. **Neural Natural Language Inference Models Partially Embed Theories of Lexical Entailment and Negation** [link]

A. Geiger, K. Richardson, and C. Potts, 2020. Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2020, Online, November 2020.

70. **Causal analysis of syntactic agreement mechanisms in neural language models**
M. Finlayson, A. Mueller, S. Gehrmann, S. Shieber, T. Linzen, and Y. Belinkov, 2021. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).

71. **Inducing causal structure for interpretable neural networks**
A. Geiger, Z. Wu, H. Lu, J. Rozner, E. Kreiss, T. Icard, N. Goodman, and C. Potts, 2022. International Conference on Machine Learning.

72. **Locating and editing factual associations in gpt**
K. Meng, D. Bau, A. Andonian, and Y. Belinkov, 2022. Advances in neural information processing systems.

73. **Towards automated circuit discovery for mechanistic interpretability**
A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso, 2023. Advances in Neural Information Processing Systems.

74. **Attribution Patching: Activation Patching At Industrial Scale** [link]
N. Nanda, 2024. .

75. **Attribution patching outperforms automated circuit discovery**
A. Syed, C. Rager, and A. Conmy, 2024. Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP.

76. **Atp*: An efficient and scalable method for localizing llm behaviour to components** [link]
J. Kram'ar, T. Lieberum, R. Shah, and N. Nanda, 2024. arXiv preprint arXiv:2403.00745.

77. **Have Faith in Faithfulness: Going Beyond Circuit Overlap When Finding Model Mechanisms** [link]
M. Hanna, S. Pezzelle, and Y. Belinkov, 2024. .

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

79. **RelP: Faithful and Efficient Circuit Discovery in Language Models via Relevance Patching** [link]
F. Jafari, O. Eberle, A. Khakzar, and N. Nanda, 2025. arXiv preprint arXiv:2508.21258.

80. **Dimensional Collapse in Transformer Attention Outputs: A Challenge for Sparse Dictionary Learning** [link]
J. Wang, X. Ge, W. Shu, Z. He, and X. Qiu, 2026. .

81. **Hermes 3 Technical Report** [link]
R. Teknium, J. Quesnelle, and C. Guang, 2024. CoRR.

82. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale [link]

G. Penedo, H. Kydl\'\icek, L. Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. Werra, and T. Wolf, 2024. Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.

83. CCI3.0-HQ: a large-scale Chinese dataset of high quality designed for pre-training large language models [link]

L. Wang, B. Zhang, C. Wu, H. Zhao, X. Shi, S. Gu, J. Li, Q. Ma, T. Pan, and G. Liu, 2024. CoRR.

84. StarCoder 2 and The Stack v2: The Next Generation [link]

A. Lozhkov, R. Li, L. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W. Li, M. Risdal, J. Li, J. Zhu, T. Zhuo, E. Zheltonozhskii, N. Dade, W. Yu, L. Krau\ss, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak, S. Paquet, J. Robinson, C. Anderson, N. Chapados, and e. al., 2024. CoRR.

85. Circuits Updates - January 2024 [link]

T. Conerly, H. Cunningham, A. Templeton, J. Lindsey, B. Hosmer, and A. Jermyn, 2024. Transformer Circuits Thread.

86. Not All Language Model Features Are Linear [link]

J. Engels, I. Liao, E. Michaud, W. Gurnee, and M. Tegmark, 2024. CoRR.

87. Feature Manifold Toy Model [link]

C. Olah and J. Batson, 2023. Transformer Circuits Thread.

88. When Models Manipulate Manifolds: The Geometry of a Counting Task [link]

W. Gurnee, E. Ameisen, I. Kauvar, J. Tarng, A. Pearce, C. Olah, and J. Batson, 2025. Transformer Circuits Thread.