`circuit-tracer`: A Library for Finding Feature Circuits

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Feature circuits aim to shed light on LLM behavior by identifying the features that are causally responsible for a given LLM output, and connecting them into a directed graph, or *circuit*, that explains how both each feature and each output arose. However, performing circuit analysis is challenging: the tools for finding, visualizing, and verifying feature circuits are complex and spread across libraries. To facilitate feature-circuit finding, we introduce `circuit-tracer`, an open-source library for efficient identification of feature circuits. `circuit-tracer` provides an integrated pipeline for finding, visualizing, annotating, and performing interventions on such circuits, tested with various model sizes, up to 14B parameters. We make `circuit-tracer` available to both developers and end users, via integration with tools such as Neuronpedia, which provides a user-friendly interface.

1 Introduction

Feature circuits are a paradigm in mechanistic interpretability that aims to provide low-level, causal interpretations of LLM behavior in an unsupervised setting. A feature circuit for a given model, input, and output aims to explain both which human-interpretable features caused the production of that output, and what caused each feature to activate.

In practice, feature circuits take the form of a directed graph from a model's inputs, through a set of features, to the model's outputs; see Figure 1 for an example. These features are causally-relevant neurons of auxiliary models such as *sparse autoencoders* (SAEs) or *transcoders*, which decompose model activations into a sparse set of features, or directions in activation space.

Feature circuits have successfully been used to study phenomena ranging from subject-verb agreement and gender bias [21], parenthesis matching [15], and syntactic structure [13]. This is possible because feature circuits are highly general: given a model, a behavior it exhibits (expressible as a single next-token prediction), and a set of auxiliary models, one can find the feature circuit for that behavior.

Unfortunately, the adoption of feature circuits has been hampered by the technical complexity of finding them. To find feature circuits, one must (1) decompose model activations into features using auxiliary models; (2) determine which features are causally relevant to the model's output; (3) visualize and annotate the circuit and its features; and (4) perform causal interventions to verify one's interpretation of the circuit. While many libraries exist for training said auxiliary models [20, 3], fewer exist for finding and visualizing circuits [21]; moreover, existing resources are not all easily interoperable. As a result, while work using the auxiliary models from (1) abounds, work that assembles these features into circuits and analyzes them as in (2)-(4) is scarce.

In this paper, we introduce `circuit-tracer`[1], a library that supports computing, visualizing, and intervening on circuits. `circuit-tracer` uses Ameisen et al.'s [1] transcoder circuits, rather than

[1] `https://anonymous.4open.science/r/circuit-tracer-anonymized-2C5F/README.md`

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

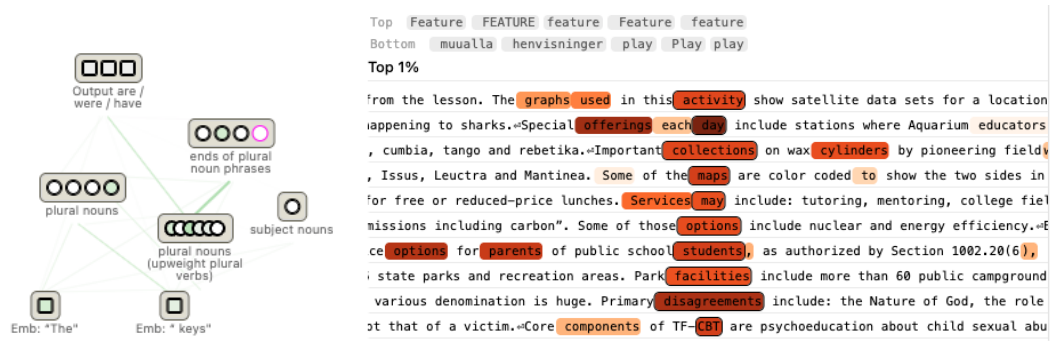


Figure 1: **Left**: A feature circuit explaining the Gemma-2 (2B)'s prediction on the input *The keys on the cabinet...*; features are grouped into annotated supernodes. **Right**: Visualizing an SAE feature. The top and bottom token predictions indicate which tokens are most up/downweighted by the feature, while the highlighted text indicates where the feature fired most strongly. This feature appears to fire on the ends of plural noun subjects.

SAE feature circuits, providing more accurate edges; our implementation enables the use of models up to 14B parameters in size. For circuit visualization, we integrate Ameisen et al.'s [1] recently-released circuit-annotation frontend, allowing users to annotate their newly-found transcoder circuits. Finally, `circuit-tracer` supports steering on transcoder features, both in the single- and efficient multi-token generation cases.

Ease of use and accessibility are core goals for `circuit-tracer`: we aim to make circuit tracing accessible to users regardless of technical experience or compute availability. For this reason, we integrate `circuit-tracer` with Neuronpedia, which enables circuit tracing via a no-code user-friendly web interface; we also optimize our library to enable running small models on Google Colab, and aim to support remote execution on public computing resources soon.

In summary, `circuit-tracer`:

- Enables users to find, visualize, and intervene on feature circuits.
- Provides an efficient open-source implementation of Ameisen et al.'s [1] transcoder circuit-tracing algorithm.
- Functions both locally and via accessible third-party compute resources, such as Google Colab, Neuronpedia's circuit tracing interface, and soon, the NDIF remote inference cluster.

The remainder of the paper is organized as follows. We first describe the circuit-finding process and existing libraries (Section 2). We then introduce `circuit-tracer`, detailing its features and usage (Section 3). We then walk through 2 case-studies in circuit tracing (Section 4). We conclude with insights gained via circuit-tracing, and directions for future work (Section 5).

2 Background

2.1 Sparse Dictionary Learning

Past work has sought to identify the features LLMs use to compute their outputs. Early work did this by identifying causally relevant neurons, but these have been found to be *polysemantic*: each neuron fires in response to many concepts [25, 4], likely because models are pressured to represent many more concepts than they have neurons [8]. Moreover, as neurons are often non-zero, it is difficult to determine when a neuron is actively firing.

Sparse dictionary learning aims to convert dense, polysemantic representations into sparse, monosemantic ones [27, 5]. Formally, a sparse dictionary takes in activations $\mathbf{h} \in \mathbb{R}^d$ from a fixed location in a model and attempts to reconstruct activations $\mathbf{h}' \in \mathbb{R}^d$ at a target location. It computes:

$$\mathbf{z} = f\left(\mathbf{W}_{enc}\mathbf{h} + \mathbf{b}_{enc}\right) \quad (1)$$

$$\tilde{\mathbf{h}'} = \mathbf{W}_{dec}\mathbf{z} + \mathbf{b}_{dec}, \quad (2)$$

where:

- $\mathbf{W}_{enc} \in \mathbb{R}^{n \times d}$, $\mathbf{W}_{dec} \in \mathbb{R}^{d \times n}$, $\mathbf{b}_{enc} \in \mathbb{R}^{n}$, and $\mathbf{b}_{dec} \in \mathbb{R}^{d}$ are model parameters;
- f is an activation function enforcing non-negativity, often ReLU, JumpReLU [28], or Top-k; and
- $\mathbf{z} \in \mathbb{R}^{n}$ is the sparse, non-negative representation. Each dimension of $\mathbf{z}$ is called a *feature*.

Sparse dictionaries are trained to minimize reconstruction error and L_1-norm of $\mathbf{z}$. This pressures $\mathbf{z}$ to faithfully represent the original input while remaining *sparse*, with few active features. $\mathbf{z}$'s features are encouraged to be monosemantic by setting its dimensionality (n) much larger than that of the input (d)—often 32 times larger, or more.

A sparse dictionary can be used to interpret a given $\mathbf{h}$ by visualizing the active features of the corresponding $\mathbf{z}$. This entails computing feature activations over a large text dataset, and inferring the meaning of the feature of interest from the text inputs that maximize its activation. It is also common to display the output tokens that are most highly up- and down-weighted by the active feature; see Figure 1 for an example.

Sparse dictionaries often aim to reconstruct the activations that they took as input; such dictionaries are called sparse autoencoders (SAEs). However, other variants exist: per-layer transcoders predict MLP outputs from their inputs [7], while cross-layer transcoders take in each layer's MLP's inputs and predict the outputs of all downstream MLPs. The choice of dictionary architecture and input / output location affects the type and number of features found.

Though sparse dictionaries have successfully shed light on various model features, it is difficult to understand the mechanisms driving a model's behavior by looking at features from one dictionary: not all active features are causally relevant to model behavior, and said behavior is often driven by features at many layers. To resolve this problem, we use feature circuits.

2.2 Feature Circuits

A feature circuit [21, 15] is a directed graph describing how a given LLM solves a given task: it flows from the model's inputs, through causally relevant features, to the model's logits. Each feature $\mathbf{z}_i$ has a weight that quantifies the change in model performance if $\mathbf{z}_i$ were set to 0; this its *total effect* through all possible pathways. Each edge's weight is the *direct effect* that the source node has on the target activation. Feature circuits thus describe which features are causally relevant, and how they combine to yield the model's outputs.

Finding a feature circuit requires a set of dictionaries for the model, generally at least one per layer. Then, one must quantify each edge or feature's (in)direct effect, pruning those with low effect. Early work did this by zero-ablating each active feature, and recording the change in model performance [15]; however, given n active features, this requires $\mathcal{O}(n)$ forward passes, making it expensive even for small models. Gradient-based methods such as Nanda's [23] activation patching, or Marks et al.'s [21] extension thereof, produce faster but lower-quality estimates of feature and edge importance.

2.3 Transcoder Feature Circuits

Transcoder feature circuits [1] are a new type of circuit that can be sparser, and allow for precise and efficient calculation of node and edge weights. Their features generally come from PLTs or CLTs; the latter provide sparser circuits, but are more challenging to train.

Ameisen et al. show that by freezing (or, conditioning on) the underlying model's nonlinearities, such as its attention patterns and LayerNorm scaling factors, one can exactly compute edge weights, i.e. the DE of one transcoder feature on another. Doing so leaves each transcoder feature's (pre-) activation (i.e., its activation before f is applied) as a linear function of the input embeddings and features that came before it. As such, one can compute the exact DE of all prior nodes on a given target node via one backwards pass from the target feature's input, with stop-gradient operations applied to the nonlinearities and prior MLP outputs.

Repeating this process for each output and feature node (or a subset thereof) yields an adjacency matrix containing the direct effect of each node on each other node. This matrix characterizes the full feature circuit, or *attribution graph*. Ameisen et al. include in their graph not only features, input, and output nodes, but also error nodes that represent the difference between the true MLP outputs and transcoder reconstructions thereof. The adjacency matrix can then be visualized, or analyzed using metrics like Ameisen et al.'s replacement score.

This approach yields precise DE values, but also has limitations: transcoder circuits often fail to capture features relevant to attention[2], as edge weights are conditioned on the attention pattern. Transcoder errors can also hinder interpretation: when a large proportion of the flow through the graph originates from uninterpretable error nodes, graphs may fail to capture important mechanisms.

2.4 Existing Libraries

Circuit research involves four distinct steps: 1) sparse dictionary training, 2) circuit-finding, 3) circuit visualization / annotation, and 4) intervention. Many libraries support the training of sparse dictionaries (1), including `dictionary-learning` [20], `SAE-Lens` [3], and `sparsify`. In contrast to these, only one library—`feature-circuits` [21]—supports finding feature circuits (2), visualizing found circuits (3), or performing interventions (4). However, it does not enable interactive circuit annotation or feature visualization, though other libraries, such as Neuronpedia [18] or SAE-Vis [22] support the latter. Moreover, at the time of `circuit-tracer`'s creation, there was no publicly available implementation of Ameisen et al.'s [1] circuit-finding algorithm, though contemporaneous work[3] has provided another open-source implementation.

In light of the abundance of sparse dictionary training libraries, we design `circuit-tracer` to support the latter three steps of circuit-finding, while remaining compatible with transcoders from any library. We focus on reducing memory usage, enabling circuit-finding in models with over 2B parameters (the largest size in prior open-source work). Finally, we prioritize accessibility, aiming to lower circuit-finding's technical barrier to entry.

3 `circuit-tracer`

In this section, we answer the following questions about `circuit-tracer`: 1) How is it designed, and what can it do?; 2) With which models is it compatible; and 3) How can it be used?

3.1 `circuit-tracer` Design and Features

3.1.1 `ReplacementModel`

In `circuit-tracer`, a model and the transcoders used to interpret it are grouped together into a `ReplacementModel`. Loading this object requires only the name of the model from HuggingFace Transformers [30], and the name of a HuggingFace Hub repository containing the transcoders:

```
from circuit_tracer import ReplacementModel

model = ReplacementModel.from_pretrained(
    model_name = "google/gemma-2-2b",
    transcoder_set = "gemma",
)
```

Listing 1: Loading a ReplacementModel based on Gemma-2 (2B) and GemmaScope transcoders. We use the alias "gemma" to refer to the latter for convenience.

The `ReplacementModel` class is used during attribution and intervention; it also enables recording the activations of transcoder features on a given input. By default, a `ReplacementModel` is a subclass of TransformerLens' `HookedTransformer` class; one can thus perform arbitrary interventions on a `ReplacementModel`, just as with TransformerLens. For more information on model and transcoder compatibility, see Section 3.2.

Currently, `circuit-tracer` expects models to be loaded onto a single GPU; other accelerators such as MPS are not yet supported. Because a model's transcoders are often much larger than the model itself, we offload transcoders' decoders to disk by default, loading them to GPU only when required;

[2] Recent work has sought to address this by incorporating attention or residual stream SAEs [16].

[3] `https://github.com/EleutherAI/attribute`

this is possible when model weights are saved in the fast SafeTensors format.[4] The memory footprint of a `ReplacementModel` is thus similar to that of its base counterpart.

3.1.2 Attribution

Once we have loaded a `ReplacementModel`, attribution in `circuit-tracer` is simple:

```
from circuit_tracer import attribute

s = "Fact: Michael Jordan plays the sport of"
graph = attribute(model, s)
```

Listing 2: Performing attribution with an existing ReplacementModel

When performing attribution, `circuit-tracer` first finds the top-10 most likely next logits, or those that compose 0.95 of the next-token probability mass, whichever is smaller. It then returns a `Graph` containing the adjacency matrix of direct effects between input, feature, error, and logit nodes that contribute to the model's prediction of those logits, as described in Section 2.3. This adjacency matrix can then be directly analyzed or visualized.

`circuit-tracer`'s attribution allows users to flexibly change the number of logits attributed from, and supports attribution from arbitrary functions of the logits, e.g. the difference of two or more logit tokens as used in prior work [29]. It also supports limiting the number of nodes attributed from; this is important, as the number of active transcoder features grows linearly with input length, slowing attribution, and causing the adjacency matrix to become prohibitively large.

3.1.3 Visualization and Annotation

Users can visualize and annotate a given attribution graph using the interface introduced by Ameisen et al. [1]. Visualizing first involves pruning the graph, which is otherwise dense and difficult to understand. Users can specify the proportion of node and edge influence they would like to retain—more influence means more nodes and edges retained—and `circuit-tracer` prunes the graph, using Ameisen et al.'s [1] algorithm. After pruning the graph, users can create the necessary visualization files and start a visualization server:

```
from circuit_tracer.utils import create_graph_files
from circuit_tracer.frontend.local_server import serve

graph_file_dir = './graph_files/'

create_graph_files(
    graph_or_path=graph,
    slug='michael-jordan',
    output_path=graph_file_dir,
    node_threshold=0.8,
    edge_threshold=0.95
)

server = serve(data_dir=graph_file_dir)
```

Listing 3: Pruning an attribution graph, creating graph files, and starting a visualization server.

The visualization interface (Figure 2) allows users to click on any node in the attribution graph, and view the nodes that most contribute to and receive contributions from that node. If the node is a feature (rather than a logit or input embedding), users can also see the max-activating examples for the feature, and then annotate the feature with its meaning on the basis of those examples.

`circuit-tracer`'s interface also allows users to pin nodes, saving those that are important and displaying them as a separate pane as a subgraph (or *circuit*), complete with weighted edges and node annotations. Nodes that appear to perform similar functions can be grouped together into a *supernode*, which can also be annotated. Users can thus use the visualization and annotation interface to transform an attribution graph into an interpretable circuit. All information about the circuit is

[4] `https://github.com/huggingface/safetensors`

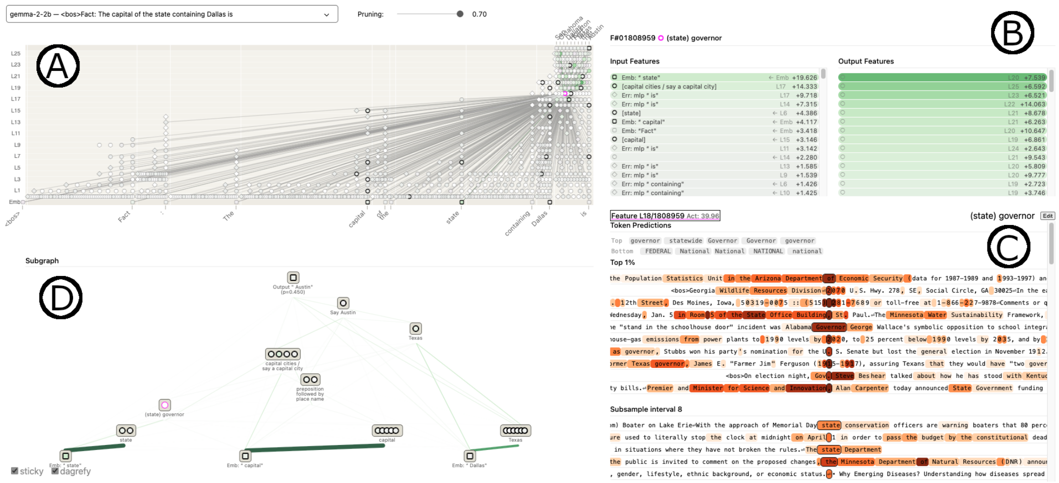


Figure 2: The circuit visualization interface. Pane A displays the entire attribution graph; nodes in the graph can be selected by clicking on them. The level of filtering can also be adjusted, further sparsifying the graph. Pane B displays the nodes that most affect (and are most affected by) the current node. Pane C displays the current feature's max-activating examples, the top and bottom upweighted tokens, and other summary statistics; it also allows for node annotation. Pane D displays the subgraph. Users can pin nodes from the attribution graph, and group them together for easier analysis; grouped tokens can also be annotated.

contained within its URL in the `circuit-tracer` interface, enabling relevant (super)nodes to be extracted from the URL and targeted for intervention.

Intervention After constructing a circuit, users can perform interventions on a given model with respect to its features, causally verifying their interpretation of the circuit. Interventions take the form of tuples specifying the layer, position, and feature index of the feature upon which to intervene, and the new value the feature should take on; interventions return the new logits and new transcoder activations post-intervention:

```
s = "Fact: Michael Jordan plays the sport of"
original_logits, original_activations = model.get_activations(s)

interventions = [(8, 3, 3829, 5.0)]
new_logits, new_activations = model.feature_intervention(s, interventions
    )
```

Listing 4: Performing an intervention, setting the value of feature 3829 in layer 8, position 3 to 5.0.

Feature interventions can be performed on on arbitrary inputs, without first finding a circuit. `circuit-tracer` allows for both single-token interventions and efficient, steered, multi-token generations using KV-caching. `circuit-tracer` performs Ameisen et al.'s [1] iterative patching by default but also implements constrained patching and direct-effects patching.

3.2 Models and Transcoders Compatible with `circuit-tracer`

Finding a circuit with `circuit-tracer` requires a compatible model and transcoders for it.

3.2.1 Models Compatible with `circuit-tracer`

`circuit-tracer`'s `ReplacementModel` supports two interpretability backends: TransformerLens (default) and NNSight. Each backend supports different models, but provides the same functionality (attribution and intervention).

TransformerLens Backend The TransformerLens [24] backend supports only those models implemented in TransformerLens. While most common open-weights model architectures (e.g. Llama, Gemma, and Qwen) are supported, less-common architectures might not be. However, TransformerLens is open-source, and new models can be added relatively easily.

NNSight Backend `circuit-tracer`'s NNSight [10] backend supports all language models on HuggingFace. Initializing a `ReplacementModel` with `backend="nnsight"` yields a subclass of NNSight's `LanguageModel` class, which retains all its functionality. Though it supports more models, the NNSight backend is slower, experimental, and does not support model offloading during attribution. In the near future, we aim to enable the NNSight backend to work with the associated National Deep Inference Facility (NDIF) remote inference servers. When this integration is complete, users will be able to perform attribution and intervention using NDIF's compute resources.

3.2.2 Transcoders Compatible with `circuit-tracer`

Existing Transcoders To use `circuit-tracer`, one needs transcoders for each MLP in the model under study. The pre-trained transcoders currently available include the following; transcoders trained by the authors except where otherwise noted[5]:

Per-Layer Transcoders (PLTs)

- Gemma-2 (2B; 11): JumpReLU PLTs from Lieberum et al. [17]
- Llama-3.2 (1B; 12): ReLU PLTs
- Qwen-3 (0.6B-14B; 31): ReLU transcoders for all dense models in the Qwen-3 family below 32B parameters.

Cross-Layer Transcoders (CLTs)

- Gemma-2 (2B): Two sets of ReLU CLTs with distinct feature dimension sizes.
- Llama-3 (1B): ReLU CLTs

Adding Transcoders `circuit-tracer` also supports user-created transcoders. Given a set of transcoder weights, one only needs to upload them, along with a configuration file that specifies where in the model the transcoder reads from and writes to, to a HuggingFace repository. Users must also compute the max-activating examples for each feature of a transcoder and upload them to the same repository in Neuronpedia's publicly-available format; code for this will soon be released in a companion library. Finally, it may be necessary to write a function to load the weights into a (`CrossLayer`-)`Transcoder` object.

3.3 Using `circuit-tracer`

To make `circuit-tracer` more widely accessible, we have published it through a variety of channels.

Neuronpedia End users who want to perform circuit-tracing without running Python code can use `circuit-tracer` on Neuronpedia[6] [18]. Neuronpedia provides a GUI for performing on-demand attribution for Gemma-2 (2B) and Qwen-3 (4B); it also supports interventions. Unlike local `circuit-tracer`, Neuronpedia provides LLM-generated interpretations of features [2] and enables saving and sharing graphs.

Google Colab Users who would like to demo `circuit-tracer` can do so via Google Colab, including Google Colab's free T4 GPU instances. Only Gemma-2 (2b) is currently available, owing to the limited amount of RAM (12.7 GB) and VRAM (15 GB) available; however, attribution, visualization, and intervention are all supported.

Local Installation Advanced users will want to use `circuit-tracer` via local installation from GitHub, where all features are available. We recommend at least 15 GB VRAM for circuit tracing with Gemma-2 (2B), and up to 40 GB for larger models; more memory also allows for faster attribution.

[5]Links to transcoders to be added if accepted

[6]Link omitted for anonymity; see screenshot in App. A.

4 Case Studies

4.1 States and Capitals

Lindsey et al. [19] observed that, given the prompt s =“Fact: The state containing Dallas has its capital in”, the models they studied could correctly predict the answer, *Austin*. Moreover, the resulting circuit clearly contained an intermediate *Texas* node, suggesting a reasoning chain of the form Dallas→Texas→Austin. Causal interventions suggested that this *Texas* node determined the state whose capital was output. With `circuit-tracer`, this result is easy to reproduce.

We first load a `ReplacementModel` for Gemma-2 (2B), using the CLTs we trained for it. We next perform attribution, creating an attribution graph for s, and visualizing it.[7] We performed manual analysis of the graph, labeling features, and found that it also contained a *Texas* feature; see Figure 3 (top) for an image of the graph. We repeated this procedure with s' =“Fact: The state containing Oakland has its capital in”, and similarly found a node corresponding to the state *California*.

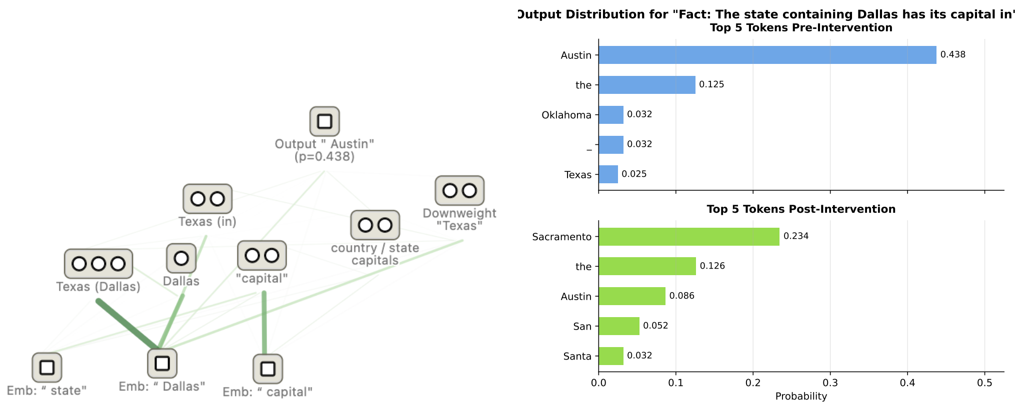


Figure 3: **Top**: Feature circuit for s=*Fact: The state containing Dallas has its capital in*, demonstrating the existence of intermediate *Texas* nodes. **Bottom**: The next-token distributions for s pre-intervention, and post-intervention, with *Texas* nodes ablated and *California* features upweighted. The most likely output shifts form *Austin* to *Sacramento*.

Having identified two relevant supernodes, we can then verify the role of each supernode by performing interventions. We first record the model’s most likely outputs on s. Then, we perform a constrained intervention on the input s, downweighting all of the features that correspond to Texas at the Dallas position(multiplying their activations by -4), and upweighting the California features (setting their activations to 10 times their original value). We constrain our intervention to layers 16-21; we choose this range because it is late enough in our model for all intervened features to have an effect. We find (Figure 3, bottom) that the model’s top outputs change drastically from the expected output of s, *Austin*, to that of s', *Sacramento*. This suggests that Gemma-2 (2B) generates “state” representations for the intermediate hop of this task.

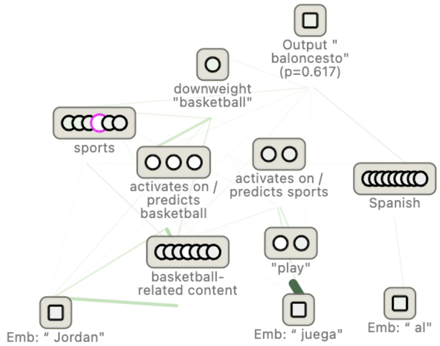
4.2 Changing Languages

Lindsey et al. [19] also observed that, given non-English prompts like s =“Hecho: Michael Jordan juega al” (*baloncesto*), models had distinct features and pathways for the underlying concept produced (*basketball*) and the output language (*Spanish*).

To reproduce this, we load a `ReplacementModel` for Gemma-2 (2B), using Lieberum et al.’s [17] PLTs; note that the previous CLTs could also be used. We again perform attribution, creating an attribution graph for s, and visualizing it (Figure 4). Once more, we identified the expected nodes (representing *basketball* and *Spanish*).

In this case, instead of verifying the validity of the Spanish features by replacing them, we simply turn them off. Moreover, rather than looking only at the next token prediction, we continually turn

[7] If accepted, we will include a link to the graph, omitted currently for anonymity reasons.



`Hecho: Michael Jordan juega al`

Normal output:

`...baloncesto en la NBA.`

Intervened output:

`...basket for the Boston Celtics.`

Figure 4: **Top**: Feature circuit for $s =$ *Hecho: Michael Jordan juega al*, showing distinct *basketball / sports* and *Spanish* features and pathways. **Bottom**: Sampled continuations to s during normal generation, and with *Spanish* features ablated. Ablating the *Spanish* features causes the model to output English text.

the Spanish feature off, while sampling new models from the token. Concretely, we perform an open-ended intervention, setting the Spanish features to -2 times their original value at all non-BOS positions in the sentence, while sampling a continuation; we compare this to the generation in the no-intervention case. We see in Figure 4 (bottom) that the model normally continues the sentence in Spanish, the intervention causes the model to continue it with English-language text.

5 Discussion and Future Work

In this paper, we introduced `circuit-tracer`, and provided a brief overview of its design and functionality. We have also outlined two brief case studies demonstrating `circuit-tracer`'s ability to reproduce existing results; more such demos can be found in the `circuit-tracer` library.

`circuit-tracer` aims to not only reproduce past work, but also support the research community as it explores open research questions. Because circuit tracing is a highly general technique, practitioners should be able to easily apply circuit tracing to their problem of choice. For example, while prior research has provided case studies in diverse safety-relevant phenomena such as chain of thought unfaithfulness, refusal, and jailbreaks [19], no systematic study of these using circuits has been performed. Moreover, many other domains, such as social biases, cognitive capabilities, and reasoning remain underexplored.

Methodological questions also abound. While `circuit-tracer` computes circuits for individual inputs, how to synthesize multiple circuits into a coherent task mechanism is still unknown. Answering this question could also require finding ways to scale feature annotation and supernode creation, which are currently highly manual processes.

`circuit-tracer` can additionally serve as a testbed for innovations in transcoders and other sparse decomposition techniques, as have been proposed in recent work [6, 14, 9, 26]. Adding these new sparse dictionaries to `circuit-tracer`, in order to assess the quality of the circuits made with them, is relatively simple. This opens up new research directions regarding the similarity of feature circuits found using different sparse decompositions of the same model.

Finally, we note that there are many features that still remain to be added to `circuit-tracer`. These range from frontend changes to improve visualization, to algorithmic additions such as attributing to thresholded MLP neurons, or from attention patterns [16]. While we are excited to add such new features, we encourage users to contribute to `circuit-tracer` as well, as some already have. `circuit-tracer` is an open source library, and we hope that a healthy community of contributors will help keep it up-to-date, even in the fast-moving field of feature circuits.

References

[1] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/methods.html`.

[2] Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models. `https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html`, 2023.

[3] Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. Saelens. `https://github.com/jbloomAus/SAELens`, 2024.

[4] Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for bert, 2021. URL `https://arxiv.org/abs/2104.07143`.

[5] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[6] Valérie Costa, Thomas Fel, Ekdeep Singh Lubana, Bahareh Tolooshams, and Demba Ba. From flat to hierarchical: Extracting sparse representations with matching pursuit, 2025. URL `https://arxiv.org/abs/2506.03093`.

[7] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable LLM feature circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=J6zHcScAo0`.

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

[9] Thomas Fel, Ekdeep Singh Lubana, Jacob S. Prince, Matthew Kowal, Victor Boutin, Isabel Papadimitriou, Binxu Wang, Martin Wattenberg, Demba E. Ba, and Talia Konkle. Archetypal SAE: Adaptive and stable dictionary learning for concept extraction in large vision models. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=9v1eW8HgMU`.

[10] Jaden Fried Fiotto-Kaufman, Alexander Russell Loftus, Eric Todd, Jannik Brinkmann, Koyena Pal, Dmitrii Troitskii, Michael Ripa, Adam Belfki, Can Rager, Caden Juang, Aaron Mueller, Samuel Marks, Arnab Sen Sharma, Francesca Lucchetti, Nikhil Prakash, Carla E. Brodley, Arjun Guha, Jonathan Bell, Byron C Wallace, and David Bau. NNsight and NDIF: Democratizing access to open-weight foundation model internals. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=MxbEiFRf39`.

[11] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin,

Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal

Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng,

Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

[13] Michael Hanna and Aaron Mueller. Incremental sentence processing mechanisms in autoregressive transformer language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3181–3203, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.164. URL `https://aclanthology.org/2025.naacl-long.164/`.

[14] Sai Sumedh R. Hindupur, Ekdeep Singh Lubana, Thomas Fel, and Demba E. Ba. Projecting assumptions: The duality between sparse autoencoders and concept geometry. In *ICML 2025 Workshop on Methods and Opportunities at Small Scale*, 2025. URL `https://openreview.net/forum?id=AKaoBzhIIF`.

[15] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=F76bwRSLeK`.

[16] Harish Kamath, Emmanuel Ameisen, Isaac Kauvar, Rodrigo Luger, Wes Gurnee, Adam Pearce, Sam Zimmerman, Joshua Batson, Thomas Conerly, Chris Olah, and Jack Lindsey. Tracing attention computation: Attention connects features, and features direct attention. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attention-qk/index.html`.

[17] Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen, editors, *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 278–300, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.blackboxnlp-1.19. URL `https://aclanthology.org/2024.blackboxnlp-1.19/`.

[18] Johnny Lin. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023. URL `https://www.neuronpedia.org`. Software available from neuronpedia.org.

[19] Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/biology.html`.

[20] Samuel Marks, Adam Karvonen, and Aaron Mueller. dictionary_learning. `https://github.com/saprmarks/dictionary_learning`, 2024.

[21] Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=I4e82CIDxv`.

[22] Callum McDougall. SAE Visualizer. https://github.com/callummcdougall/sae_vis, 2024.

[23] Neel Nanda. Attribution Patching: Activation Patching At Industrial Scale, 2023. URL https://www.neelnanda.io/mechanistic-interpretability/attribution-patching.

[24] Neel Nanda and Joseph Bloom. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

[25] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. URL https://distill.pub/2017/feature-visualization.

[26] James Oldfield, Shawn Im, Yixuan Li, Mihalis A. Nicolaou, Ioannis Patras, and Grigorios G Chrysos. Towards interpretability without sacrifice: Faithful dense layer decomposition with mixture of decoders, 2025. URL https://arxiv.org/abs/2505.21364.

[27] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997. ISSN 0042-6989. doi: https://doi.org/10.1016/S0042-6989(97)00169-7. URL https://www.sciencedirect.com/science/article/pii/S0042698997001697.

[28] Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders, 2024. URL https://arxiv.org/abs/2407.14435.

[29] Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

[30] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

[31] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

A Neuronpedia Interface

The Neuronpedia `circuit-tracer` interface is visible in Figure 5.

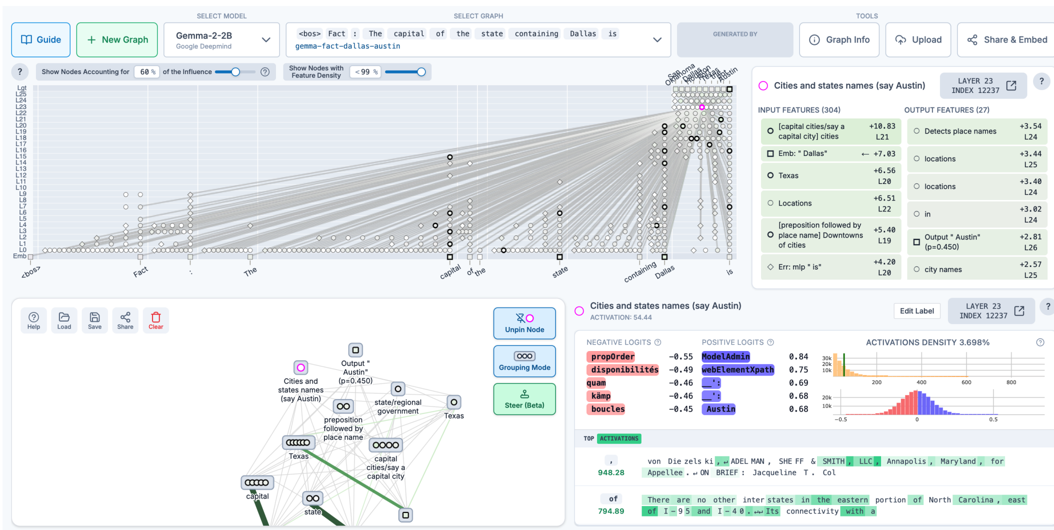


Figure 5: The interface of `circuit-tracer` when accessed via Neuronpedia [18]. Users can easily create a new graph, by clicking on `+ New Graph`. They can also upload existing graphs. Neuronpedia provides automatic, LLM-derived interpretations of transcoder features, though it also supports manual facilitation. It moreover facilitates grouping features into labeled supernodes, and saving the resulting circuit.