circuit-tracer: A New Library for Finding Feature Circuits

Michael Hanna[1*], **Mateusz Piotrowski**[2*], **Jack Lindsey**[2], **Emmanuel Ameisen**[2]
[1]ILLC, University of Amsterdam [2]Anthropic
m.w.hanna@uva.nl

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 multiple 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 feature 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 a variety of phenomena, ranging from subject-verb agreement and gender bias (Marks et al., 2025), parenthesis matching (Huben et al., 2024), and syntactic structure (Hanna and Mueller, 2025). 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.

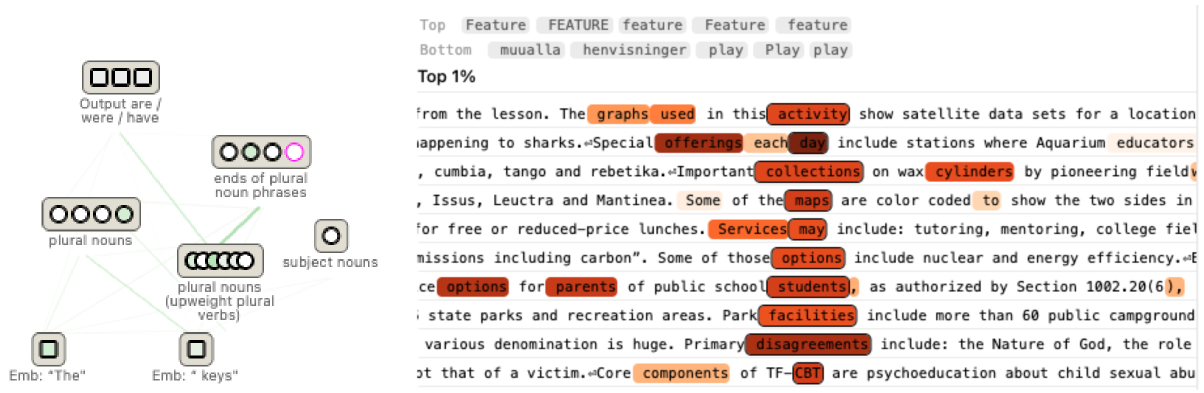
Unfortunately, the adoption of feature circuits has been hampered by the technical complexity of finding them. To find feature circuits, one must (1) decompose model activations into features using auxiliary models; (2) determine which features are causally relevant to the model's output; (3) visualize 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 (Marks et al., 2024; Bloom et al., 2024), fewer exist for finding and visualizing circuits (Marks et al., 2025); 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 (2025) transcoder circuits, rather than 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 (2025) 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.

[*]Equal contribution. Work completed as part of the Anthropic Fellows Program.

[1]https://github.com/safety-research/circuit-tracer



```
Top     Feature  FEATURE  feature  Feature  feature
Bottom   muualla  henvisninger  play  Play  play
Top 1%
from the lesson. The graphs used in this activity show satellite data sets for a location
appening to sharks.⏎Special offerings each day include stations where Aquarium educators
, cumbia, tango and rebetika.⏎Important collections on wax cylinders by pioneering fieldw
, Issus, Leuctra and Mantinea. Some of the maps are color coded to show the two sides in
for free or reduced-price lunches. Services may include: tutoring, mentoring, college fiel
missions including carbon". Some of those options include nuclear and energy efficiency.⏎E
ce options for parents of public school students, as authorized by Section 1002.20(6),
 state parks and recreation areas. Park facilities include more than 60 public campground
 various denomination is huge. Primary disagreements include: the Nature of God, the role
ot that of a victim.⏎Core components of TF-CBT are psychoeducation about child sexual abu
```

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.

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 (2025) 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 (Olah et al., 2017; Bolukbasi et al., 2021), likely because models are pressured to represent many more concepts than they have neurons (Elhage et al., 2022). 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 (Olshausen and Field, 1997; Bricken et al., 2023). 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:

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

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

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 (Rajamanoharan et al., 2024), or Top-k (Gao et al., 2025); 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. They are also trained to limit the number of active features, if their activation function does not do so naturally (as Top-k does); typically, this entails minimizing the 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 first computing all features' activations over a large text dataset. Then, one bins each feature's activations into quantiles, and visualizes a random subset of the text inputs that fall into each quantile; typically, the max-activating texts (in the top quantile) are most informative. 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. Given these, one can assign an interpretation to a feature either via manual inspection or using an LLM (Bills et al., 2023), though how to best evaluate such interpretations remains an open question (Paulo et al., 2025; Heap et al., 2025).

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 (PLTs) predict MLP outputs from their inputs (Dunefsky et al., 2024). Cross-layer transcoders (CLTs) take in each layer's MLP's inputs and predict a contribution to the output of each downstream MLP; the reconstruction of a given MLP output is given by the sum of the contributions of all prior CLTs (Lindsey et al., 2024). The choice of dictionary architecture and input / output location affects the types 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 (Marks et al., 2025; Huben et al., 2024) 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 is its *total effect* through all possible pathways. Each edge's weight is the *direct effect* (DE) 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 direct or total effect, pruning those with low effect. Early work did this by zero-ablating each active feature, and recording the change in model performance (Huben et al., 2024); 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 (2023) activation patching, or Marks et al.'s (2025) extension thereof, produce faster but lower-quality estimates of feature and edge importance.

2.3 Transcoder Feature Circuits

Transcoder feature circuits (Ameisen et al., 2025) 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 Feature Circuit Libraries

Working with feature circuits often involves four steps: 1) training sparse dictionaries, 2) finding feature circuits, 3) visualizing and annotating on said circuits, and 4) intervening on these circuits. Currently, there exist libraries for individual steps in this process, but none that support all steps of it.

Many libraries support the training of sparse dictionaries (1), including `dictionary-learning` (Marks et al., 2024), `SAE-Lens` (Bloom et al., 2024), and `sparsify`. In contrast to these, only one library—`feature-circuits` (Marks et al., 2025)—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 (Lin, 2023) or SAE-Vis (McDougall, 2024) support the latter. Moreover, at the time of `circuit-tracer`'s creation, there was no publicly available implementation of Ameisen et al.'s (2025) circuit-finding algorithm, though contemporaneous work[3] has since provided another open-source implementation.

In light of the abundance of libraries for sparse dictionary training, and the high computational expense associated with that process, we design `circuit-tracer` to support the latter three steps of circuit-finding. However, `circuit-tracer` aims to support transcoders trained with any library.

Compared to past work, we focus on efficiency and accessibility. We minimize `circuit-tracer`'s memory usage, enabling circuit-finding in models with 14B parameters—well over 2B, the largest size in prior open-source work. We also simplify the circuit-finding process, allowing users to find a circuit given just a single prompt, where earlier work required constructing a dataset and attribution metrics. `circuit-tracer` facilitates visualization as well, via an interactive interface that enables users to analyze the circuit and the features composing it at the same time. Finally, we make `circuit-tracer` available via many channels, including three—Google Colab, Neuronpedia, and soon, NDIF (Fiotto-Kaufman et al., 2025)—that require no compute resources on the user's end; see Section 3.2 and Section 3.3 for details.

[2]Recent work has sought to address this by incorporating attention or residual stream SAEs (Kamath et al., 2025).

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

2.5 Interpretability Libraries

In releasing `circuit-tracer`, we contribute to a line of research that makes interpretability more accessible by taking existing methods and releasing open-source implementations, with user-friendly interfaces. Libraries for explainable AI and input attribution are especially abundant: Inseq provides attribution tools for text-based models (Sarti et al., 2023), while Quantus and Zennit (Hedström et al., 2023; Anders et al., 2023) focus on the image domain; the Captum library is modality-agnostic (Kokhlikyan et al., 2020). Our work is more closely related to libraries such as Auto-Circuit (Miller et al., 2024) or EAP-IG (Hanna et al., 2024), which find and visualize component-level LLM circuits. Foundational to this effort are libraries, such as TransformerLens (Nanda and Bloom, 2022), NNSight (Fiotto-Kaufman et al., 2025), Pyvene (Wu et al., 2024), and Penzai (Johnson, 2024), that provide ready access to model internals.

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` object. Loading such a `ReplacementModel` requires only the name of the model from HuggingFace Transformers (Wolf et al., 2020), 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; 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:

```python
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 (Wang et al., 2023). 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.

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

3.1.3 Visualization and Annotation

Users can visualize and annotate a given attribution graph using the interface introduced by Ameisen et al. (2025). 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 (2025) algorithm. After pruning the graph, users can create the necessary visualization files and start a visualization server:

```python
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 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

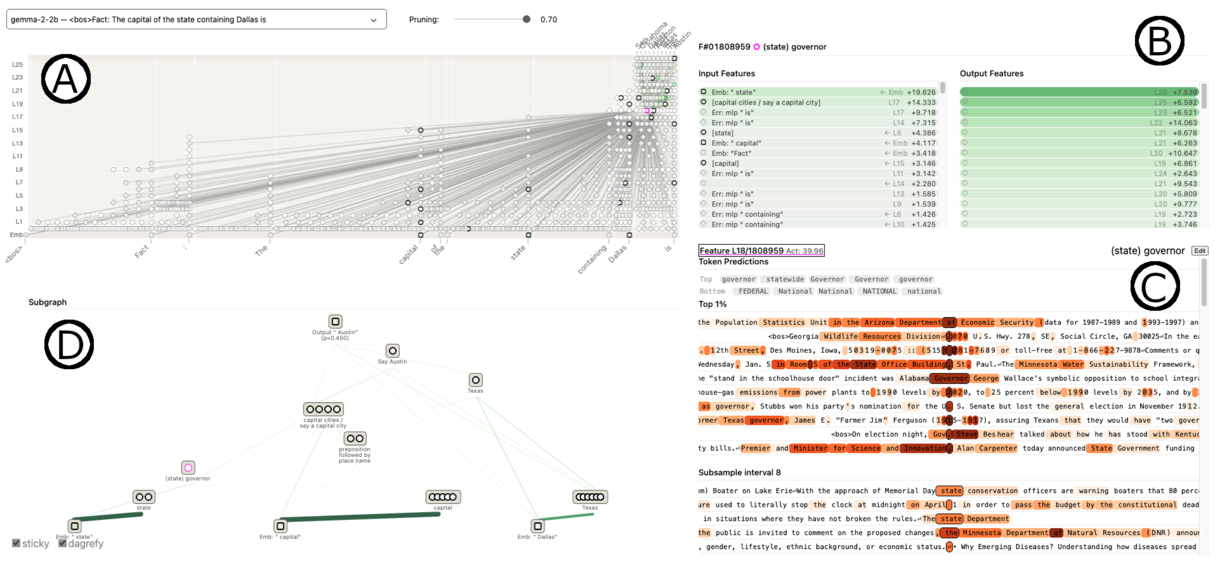


Figure 2: The circuit visualization interface. Pane A displays the entire attribution graph; nodes in the graph can be selected by clicking on them. The level of filtering can also be adjusted, further sparsifying the graph. Pane B displays the nodes that most affect (and are most affected by) the current 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.

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 (2025) 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 (Nanda and Bloom, 2022) 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 (Fiotto-Kaufman et al., 2025) 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, rather than rely on their own.

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, trained by the authors except where otherwise noted[5]:

Per-Layer Transcoders (PLTs)

- Gemma-2 (2B; Gemma Team et al., 2024): JumpReLU PLTs from Lieberum et al. (2024)
- Llama-3.2 (1B; Grattafiori et al., 2024): ReLU PLTs
- Qwen-3 (0.6B-14B; Yang et al., 2025): 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] (Lin, 2023). 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 (Bills et al., 2023) 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 Most advanced users will want to use `circuit-tracer` via local installation from GitHub, where all features are available. We recommend a GPU with at least 15 GB VRAM for circuit tracing with Gemma-2 (2B), and up to 40 GB for larger models; in general more memory will allow for faster attribution.

4 Case Studies

4.1 States and Capitals

Lindsey et al. (2025) observed that, given the prompt s ="Fact: The capital of the state containing Dallas is", 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 2.5M-feature 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 capital of the state containing Oakland is", and similarly found a node corresponding to the state *California*.

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

[5] PLTs available at this link; CLTs available at this link.

[6] `https://www.neuronpedia.org/gemma-2-2b/graph`

[7] View the graph on Neuronpedia.

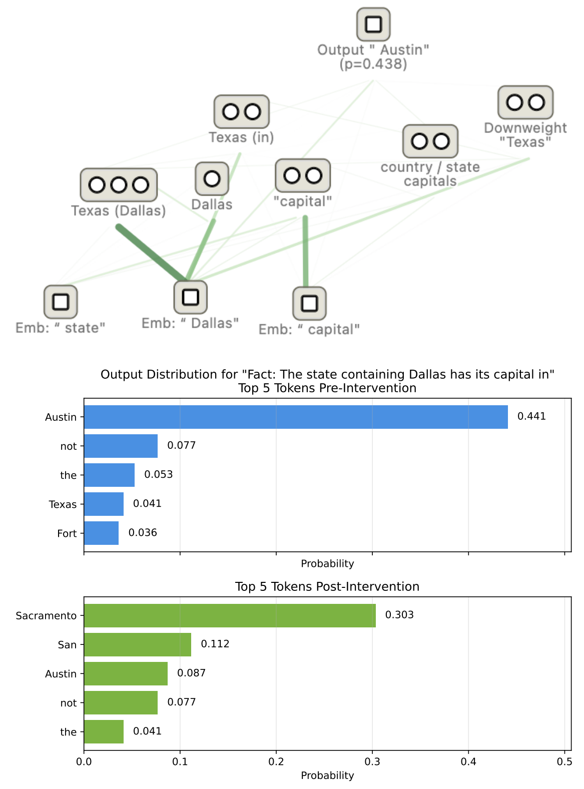


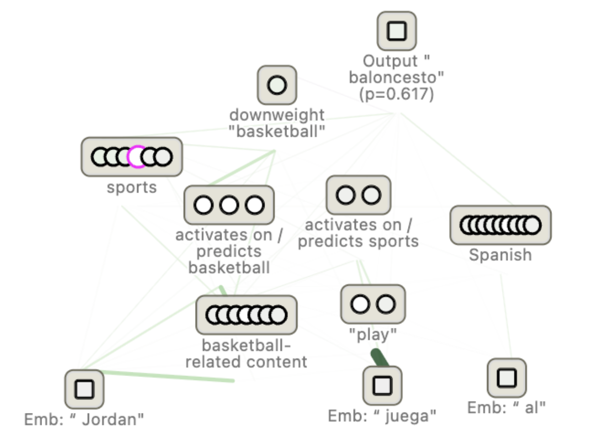
Figure 3: **Top**: Feature circuit for *s*=*Fact: The capital of the state containing Dallas is*, demonstrating the existence of intermediate *Texas* nodes. **Bottom**: The next-token distributions for s pre-intervention, and post-intervention, with *Texas* nodes ablated and *California* features upweighted.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

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. (2025) 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 (2024) 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)[8]. Once more, we identified the expected nodes (representing *basketball* and *Spanish*).


Output " baloncesto" (p=0.617)
downweight "basketball"
sports
activates on / predicts basketball
activates on / predicts sports
Spanish
basketball-related content
"play"
Emb: " Jordan"
Emb: " juega"
Emb: " al"

`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.

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 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

[8]Graph available on Neuronpedia

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 (Lindsey et al., 2025), 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 (Costa et al., 2025; Hindupur et al., 2025; Fel et al., 2025; Oldfield et al., 2025). 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 (Kamath et al., 2025). 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.

Acknowledgments

The authors thank Johnny Lin for his invaluable help and support in making circuit-tracer available on Neuronpedia; the authors also thank Curt Tigges and David Chanin for their contributions to this process. The authors also thank those who have contributed PRs to circuit-tracer. We thank Jaden Fiotto-Kauffman, Adam Belfki, and Emma Bortz from the NNSight team for their support in porting circuit-tracer to NNSight. Finally, the authors thank the Anthropic Fellows Program for enabling and supporting this project.

References

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, and 8 others. 2025. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread.*

Christopher J. Anders, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. 2023. Software for dataset-wide xai: From local explanations to global insights with zennit, corelay, and virelay. *Preprint*, arXiv:2106.13200.

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

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

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. 2021. An interpretability illusion for bert. *Preprint*, arXiv:2104.07143.

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, and 6 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Valérie Costa, Thomas Fel, Ekdeep Singh Lubana, Bahareh Tolooshams, and Demba Ba. 2025. From flat to hierarchical: Extracting sparse representations with matching pursuit. *Preprint*, arXiv:2506.03093.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. 2024. Transcoders find interpretable LLM feature circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

Thomas Fel, Ekdeep Singh Lubana, Jacob S. Prince, Matthew Kowal, Victor Boutin, Isabel Papadimitriou, Binxu Wang, Martin Wattenberg, Demba E. Ba, and Talia Konkle. 2025. Archetypal SAE: Adaptive and stable dictionary learning for concept extraction in large vision models. In *Forty-second International Conference on Machine Learning*.

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. 2025. NNsight and NDIF: Democratizing access to open-weight foundation model internals. In *The Thirteenth International Conference on Learning Representations*.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2025. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*.

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, and 179 others. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

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, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Michael Hanna and Aaron Mueller. 2025. Incremental sentence processing mechanisms in autoregressive transformer language models. In *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. Association for Computational Linguistics.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. 2024. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. In *First Conference on Language Modeling*.

Thomas Heap, Tim Lawson, Lucy Farnik, and Laurence Aitchison. 2025. Sparse autoencoders can interpret randomly initialized transformers. *Preprint*, arXiv:2501.17727.

Anna Hedström, Leander Weber, Daniel Krakowczyk, Dilyara Bareeva, Franz Motzkus, Wojciech Samek, Sebastian Lapuschkin, and Marina M.-C. Höhne. 2023. Quantus: An explainable ai toolkit for responsible evaluation of neural network explanations and beyond. *Journal of Machine Learning Research*, 24(34):1–11.

Sai Sumedh R. Hindupur, Ekdeep Singh Lubana, Thomas Fel, and Demba E. Ba. 2025. Projecting assumptions: The duality between sparse autoencoders and concept geometry. In *ICML 2025 Workshop on Methods and Opportunities at Small Scale*.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*.

Daniel D. Johnson. 2024. Penzai + Treescope: A toolkit for interpreting, visualizing, and editing models as data. *ICML 2024 Workshop on Mechanistic Interpretability*.

Harish Kamath, Emmanuel Ameisen, Isaac Kauvar, Rodrigo Luger, Wes Gurnee, Adam Pearce, Sam Zimmerman, Joshua Batson, Thomas Conerly, Chris Olah, and Jack Lindsey. 2025. Tracing attention computation: Attention connects features, and features direct attention. *Transformer Circuits Thread*.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for pytorch. *Preprint*, arXiv:2009.07896.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. 2024. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 278–300, Miami, Florida, US. Association for Computational Linguistics.

Johnny Lin. 2023. Neuronpedia: Interactive reference and tooling for analyzing neural networks. Software available from neuronpedia.org.

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, and 8 others. 2025. On the biology of a large language model. *Transformer Circuits Thread*.

Jack Lindsey, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. 2024. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*.

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

Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. 2025. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *The Thirteenth International Conference on Learning Representations*.

Callum McDougall. 2024. SAE Visualizer. `https://github.com/callummcdougall/sae_vis`.

Joseph Miller, Bilal Chughtai, and William Saunders. 2024. Transformer circuit evaluation metrics are not robust. In *First Conference on Language Modeling*.

Neel Nanda. 2023. Attribution Patching: Activation Patching At Industrial Scale.

Neel Nanda and Joseph Bloom. 2022. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. 2017. Feature visualization. *Distill*.

James Oldfield, Shawn Im, Yixuan Li, Mihalis A. Nicolaou, Ioannis Patras, and Grigorios G Chrysos. 2025. Towards interpretability without sacrifice: Faithful dense layer decomposition with mixture of decoders. *Preprint*, arXiv:2505.21364.

Bruno A. Olshausen and David J. Field. 1997. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325.

Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. 2025. Automatically interpreting millions of features in large language models. *Preprint*, arXiv:2410.13928.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. 2024. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *Preprint*, arXiv:2407.14435.

Gabriele Sarti, Nils Feldhus, Ludwig Sickert, and Oskar van der Wal. 2023. Inseq: An interpretability toolkit for sequence generation models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 421–435, Toronto, Canada. Association for Computational Linguistics.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.

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, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah Goodman, Christopher Manning, and Christopher Potts. 2024. pyvene: A library for understanding and improving PyTorch models via interventions. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pages 158–165, Mexico City, Mexico. Association for Computational Linguistics.

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, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.