IMPROVING UPON OPEN-SOURCE LLM ATTRIBUTION GRAPH VISUALIZATION TOOLS

Z. Goel [*], Min Lu [†], and Tara Liu [‡]

[1]Georgia Institute of Technology

1 INTRODUCTION

Recent advances in mechanistic interpretability have demonstrated that large language models (LLMs) encode human-interpretable features within their internal representations. Tools such as sparse autoencoders (SAEs) and transcoders have made it possible to extract and analyze these features, offering new insights into how models process and generate language. In particular, the work by Anthropic on scaling monosemanticity [11] showed that SAEs trained on large models can recover discrete, semantically meaningful features that correspond to high-level concepts, while the circuit tracing framework [1] introduced cross-layer transcoders (CLTs) that enable the construction of attribution graphs tracing the flow of information across layers.

These developments provide a foundation for understanding model behavior in a structured and causally grounded way. However, they are not without limitations. As SAE width increases, models tend to exhibit feature absorption, where general features are subsumed by more specific ones, and feature fragmentation, where a single coherent concept is split across multiple latents. These phenomena degrade interpretability and pose challenges for downstream applications such as feature-based circuit discovery. These issues have been discussed in recent work by Nabeshima , Bussmann et al. ([3], [8]).

To address these challenges, the Matryoshka SAE framework introduces a hierarchical training objective wherein the autoencoder is optimized over multiple prefixes of its latent dictionary. This encourages early latents to capture broad, general features while later latents specialize on more specific details. Empirical results suggest that Matryoshka SAEs reduce feature absorption and preserve abstract features better than standard approaches, even at scale.

While promising, existing implementations of Matryoshka SAEs are typically restricted to a single layer of the model. Transcoders, in contrast, operate across layers, enabling attribution and feature tracing throughout the entire network. This raises the question: can the Matryoshka principle be extended beyond SAEs to cross-layer transcoders, thereby yielding a more globally consistent and hierarchical feature decomposition?

In this project, we aim to develop such an approach by integrating Matryoshka SAEs and Matryoshka CLTs into EleutherAI's open-source interpretability tools **clt-training** and **attribute**, which is built on top of Anthropic's work on circuit tracing via attribution graphs ([1], [2], [6]). Our goal is to implement and evaluate these methods, and to design an interactive visualization interface that enables users to explore attribution graphs at multiple levels of abstraction. Specifically, we aim to support the decomposition of complex features into constituent subfeatures, facilitating a more granular and interpretable view of model behavior.

2 BACKGROUND AND RELATED WORK

In traditional works of mechanistic interpretability, SAEs are used to understand internal model behavior by extracting human interpretable features from models and then projecting those activations onto a sparse, higher dimensional latent space. A limitation with this approach is that using these

[*]zgoel3@gatech.edu
[†]mlu316@gatech.edu
[‡]tliu479@gatech.edu

features as input for circuit analysis is often not scalable and results produced by SAEs are also not yet provably effective at disentangling local and global behavior [11].

2.1 Overview of Circuit Analysis

Deep learning models produce their outputs using a series of transformations distributed across many computational units called artificial "neurons". Mechanistic interpretability seeks to understand the black box of machine learning models by extracting features of the machine learning model and the processes by which these features interact, called circuits.

A big problem in circuit analysis is that the features are often hard to identify, due to the polysemanticity of artificial neurons. Polysemantic neurons are neurons that appear to be used by the model to represent more than one independent concept [11]. Therefore, a key first step to circuit analysis is often to extract features in a way that represents a single concept.

2.2 Sparse-Autoencoders and Transcoders

Sparse-autoencoders and transcoders are two types of sparse coding models. Unlike the normal machine learning models we use with a dense output layer, sparse coding models ae designed to decompose model activates into sparsely active components. Rather than using raw neurons, these components are then used as the features for our circuit analysis. Sparse Autoencoders (SAEs) are neural networks designed to decompose model activations into sparse, interpretable features. They are particularly useful in addressing superposition, where multiple concepts are entangled within a neural network's activation space. An SAE consists of an encoder, which maps input activations to a higher-dimensional, sparse latent space, and a decoder which reconstructs the original input from the sparse representation.

Let the input activation be denoted as $x \in \mathbb{R}^d$. The encoder is parameterized by a weight matrix $W_e \in \mathbb{R}^{k \times d}$ and a bias vector $b_e \in \mathbb{R}^k$. The decoder is similarly parameterized by a weight matrix $W_d \in \mathbb{R}^{d \times k}$ and bias $b_d \in \mathbb{R}^d$. An activation function f, typically ReLU or TopK, is applied after the encoding step.

The encoding step is given by:

$$z = f(W_e x + b_e)$$

The decoding step reconstructs the input as:

$$\hat{x} = W_d z + b_d$$

The overall loss function combines reconstruction accuracy with a sparsity penalty on the latent representation:

$$\mathcal{L} = \|x - \hat{x}\|^2 + \lambda \cdot \|z\|_1$$

Here, $\|x - \hat{x}\|^2$ represents the reconstruction loss, $\|z\|_1$ is the L1 sparsity penalty encouraging sparse activations in z, and λ is a hyperparameter that controls the trade-off between reconstruction quality and sparsity.

To explicitly enforce sparsity, the SAE may use the TopK activation function, defined as:

$$\text{TopK}(z)_i = \begin{cases} z_i, & \text{if } z_i \text{ is among the top } K \text{ activations} \\ 0, & \text{otherwise} \end{cases}$$

The goal of training SAEs in this context is to achieve monosemanticity, in which each feature in z corresponds to a specific, interpretable concept (e.g., a token, syntactic role, or logical structure). By reconstructing original activations from sparse latent codes, SAEs isolate independent features and make neural activations easier to interpret [11].

Transcoders are a learned approximation to a transformer's MLP sublayer. They are designed to both closely match the original MLP's input-output mapping and remain interpretable. A transcoder replaces the MLP sublayer, which maps an input vector $x \in \mathbb{R}^{d_{\text{model}}}$ to an output vector $y \in \mathbb{R}^{d_{\text{model}}}$, with a wide, one-hidden-layer ReLU MLP:

$$\text{TC}(x) \;=\; W_{\text{dec}}\,\text{ReLU}(W_{\text{enc}}\,x \;+\; b_{\text{enc}}) \;+\; b_{\text{dec}},$$

where $W_{\text{enc}} \in \mathbb{R}^{d_{\text{features}} \times d_{\text{model}}}$ and $W_{\text{dec}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{features}}}$ define how the hidden feature vectors are extracted from the input and then recombined. In most implementations, d_{features} is much larger than d_{model}, so each learned feature can fire relatively sparsely via the ReLU. The transcoder is trained by adding an L_1 penalty on hidden activations [4]:

$$\mathcal{L}_{\text{TC}}(x) \;=\; \big\|\text{MLP}(x) \;-\; \text{TC}(x)\big\|_2^2 \;+\; \lambda_1 \Big\|\text{ReLU}\big(W_{\text{enc}}\, x \;+\; b_{\text{enc}}\big)\Big\|_1.$$

While SAEs can reveal features that appear coherent, they have limitations when trying to interpret how those features move through an MLP layer. First, any given SAE feature tends to be a linear combination of many neurons, and each neuron is subject to the MLP's nonlinearities. This makes it hard to track how one feature in the current layer contributes to another feature in the next layer without resorting to single-input interventions or gradient-based scores, which capture only input-specific effects. Second, it is difficult to make general statements about "for all inputs, if feature A is highly active, it contributes in the following way to feature B." SAE-based approaches typically do not cleanly separate input dependence (whether a feature activates on a given token) from input-invariant structure (how the feature, once activated, flows forward).

In contrast, a transcoder is directly trained to approximate the MLP's outputs ([4], [5]). That means we can look at its factorization:

$$\text{Attribution from feature } i \text{ to feature } j \;=\; \Big(\text{ReLU}(W_{\text{enc},i} \cdot x)\Big)\,\Big(W_{\text{dec},i} \;\cdot\; W_{\text{enc},j}\Big).$$

The first term, $\text{ReLU}(W_{\text{enc},i} \cdot x)$, depends on the input x (i.e. whether feature i actually activates). The second term, $\big(W_{\text{dec},i} \cdot W_{\text{enc},j}\big)$, is a dot product of the decoder vector of feature i with the encoder vector of feature j. Crucially, this second term is constant across all inputs. This allows us to create a computational graph" of which features connect to which for all inputs. The result is a far cleaner method of attributing how an MLP sublayer's activation flows from earlier features to later features. In other words, transcoders provide a more direct pathway to analyzing entire circuits that pass through MLP layers.

2.3 SKIP-TRANSCODERS

Skip transcoders are an architectural refinement of transcoders, designed to improve faithfulness without sacrificing interpretability. A standard transcoder approximates an MLP sublayer by learning a sparse, wide ReLU MLP that maps from the MLP input to its output. Skip transcoders augment this with an affine skip connection from the input to the output, yielding the functional form:

$$f(x) = W_2 \cdot \text{TopK}(W_1 x + b_1) + W_{\text{skip}} x + b_2$$

where $W_{\text{skip}} x$ provides a linear bypass path.

This addition enables the sparse component to specialize in modeling nonlinear, semantically meaningful residuals, while the skip path captures the dominant linear behavior. Empirically, skip transcoders consistently outperform both standard transcoders and sparse autoencoders (SAEs) on the tradeoff between reconstruction fidelity (measured by cross-entropy loss increase) and feature interpretability (measured by automated detection and fuzzing scores). Furthermore, skip transcoders achieve higher interpretability scores and lower loss degradation when patched into language models, and generalize better across activation distributions.

[10].

2.4 ANTHROPIC VARIATIONS ON TRANSCODERS

The Anthropic team implemented two variations of transcoders: Per-Layer Transcoders (PLTs) and Cross-Layer Transcoders (CLTs), applying the transcoder architecture at every MLP layer in two transformer models (an 18-layer transformer and Claude 3.5 Haiku) [1]. In PLTs, each layer ℓ has a distinct encoder-decoder pair $\left(W_{\text{enc}}^{(\ell)}, W_{\text{dec}}^{(\ell)}\right)$ used to reconstruct the next layer's MLP output $y_{\ell+1}$ from the current layer's input x_ℓ:

$$\hat{y}_{\ell+1} = W_{\text{dec}}^{(\ell)}\, \text{ReLU}\left(W_{\text{enc}}^{(\ell)} x_\ell + b_{\text{enc}}^{(\ell)}\right) + b_{\text{dec}}^{(\ell)}.$$

The PLT objective minimizes a sum of squared reconstruction losses across layers, with an additional sparsity penalty applied to the tanh-transformed encoder outputs [1]:

$$\mathcal{L}_{\text{PLT}} = \sum_{\ell=1}^{L-1} \|\hat{y}_{\ell+1} - y_{\ell+1}\|_2^2 + \lambda \sum_{\ell=1}^{L-1} \left\| \tanh(W_{\text{enc}}^{(\ell)} x_\ell + b_{\text{enc}}^{(\ell)}) \right\|_1 .$$

By contrast, CLTs use a single shared encoder-decoder pair $(W_{\text{enc}}, W_{\text{dec}})$ across all layers. For each layer ℓ, the layer's MLP input x_ℓ is passed through the transcoder [1]:

$$\hat{y}_\ell = W_{\text{dec}} \, \text{ReLU}(W_{\text{enc}} x_\ell + b_{\text{enc}}) + b_{\text{dec}}.$$

The CLT loss is computed as the sum of squared reconstruction errors across layers, along with an ℓ_1 sparsity penalty on the tanh-transformed encoder activations [1]:

$$\mathcal{L}_{\text{CLT}} = \sum_{\ell=1}^{L} \|\hat{y}_\ell - y_\ell\|_2^2 + \lambda \sum_{\ell=1}^{L} \|\tanh(W_{\text{enc}} x_\ell + b_{\text{enc}})\|_1 .$$

To construct a replacement model using either transcoder approach, the decoded feature outputs from all layers are accumulated [1]. Specifically, the output at layer ℓ is replaced by the sum of decoder output from layers 1 through ℓ:

$$\tilde{y}_\ell = \sum_{k=1}^{\ell} W_{\text{dec}} \, \text{ReLU}(W_{\text{enc}} x_k + b_{\text{enc}}) + b_{\text{dec}},$$

They then compared these approaches by creating their respective replacement models and comparing their output distributions to those of the original model. They ultimately found that the CLT approach performed better with respect to Top-1 Error Rate and KL Divergence.

2.5 Matryoshka SAEs and its Variants

Matryoshka SAEs aim to solve the problems of feature absorption and fragmentation. SAEs break down neural network internals into more interpretable latents, which may correspond to actual "features" the model uses when analyzing an input. As SAE size increases, these latents become more fine grained, and an abstract concept represented by a latent might develop "holes", where another latent assumes responsibility for that part of that particular concept. Hence the term "feature absorption," as one latent essentially "absorbs" part of another. Furthermore, as latents become more granular as a result of the increasing SAE size, fragmentation occurs, wherein abstract features disappear and "fragment" into many more specific, smaller latents. This represents a tradeoff: more fragmented latents may allow for a more finegrained analysis of model internals, but may also be less interpretable than a smaller number of more abstract features.

Matryoshka Sparse Autoencoders (Mat-SAEs) are a recently introduced variant of sparse autoencoders designed to learn features at multiple levels of abstraction [3]. The key idea is to partition the latent space into nested groups of increasing size and train each group to reconstruct the input independently. Earlier latents are reused across more loss terms, encouraging them to capture broad, generalizable features, while later latents specialize in fine-grained or residual information.

Formally, let $f(x) = \text{BatchTopK}(W_{\text{enc}} x + b_{\text{enc}})$ denote the sparse latent encoding of input x [3]. Define a set of prefix sizes $\mathcal{P} = \{p_1, \ldots, p_K\}$, each specifying a truncated subset of the latent vector. For each prefix p_k, the reconstruction is computed as:

$$\hat{x}^{(p_k)} = W_{\text{dec}}^{(p_k)} f(x)_{1:p_k} + b_{\text{dec}}^{(p_k)},$$

where $W_{\text{dec}}^{(p_k)}$ selects only the first p_k decoder vectors. The full Matryoshka-SAE loss is then:

$$\mathcal{L}_{\text{Mat-SAE}} = \sum_{p_k \in \mathcal{P}} \left\| \hat{x}^{(p_k)} - x \right\|_2^2 + \lambda \sum_{p_k \in \mathcal{P}} \|f(x)_{1:p_k}\|_1 .$$

This loss enforces a coarse-to-fine structure on the latent space, promoting disentanglement and reusability of early features across multiple levels of detail [3].

COMPARISON OF MAT-SAE IMPLEMENTATIONS

Two implementations of Mat-SAEs were developed independently in parallel by Nabeshima and Bussmann et al. ([8], [3]). The main differences are listed below.

- **Prefix Sampling:** Nabeshima samples latent prefixes from a truncated Pareto distribution, applying stochastic nesting pressure across a broad range of dimensionalities. Bussmann et al. use fixed group sizes (e.g., halves or quarters of the latent dictionary), which may offer more stable training.
- **Latent Activation:** Bussmann et al. use `BatchTopK` for activation sparsity, while Nabeshima may rely on standard top-k sparsity or a variant thereof.
- **Latent Ordering:** Bussmann et al. rely on natural ordering from training; Nabeshima explores explicit latent reordering strategies to improve interpretability.

As of the time of this writing, there has not been any structured work that benchmarks these two implementations, so a minor goal of this project will be to integrate the authors' original open-source implementations into our project and benchmark these two implementations using our criteria.

2.6 CONSTRUCTING ATTRIBUTION GRAPHS FROM TRANSCODER WEIGHTS

To analyze how sparse features influence each other across layers in the transformer, the Anthropic team constructs attribution based on the parameters of trained transcoders. These graphs serve as a structural abstraction of the flow of information between features, enabling researchers to trace and interpret the model's internal circuits.

Let $f_i(x) = \mathrm{ReLU}(W_{\mathrm{enc},i} \cdot x + b_{\mathrm{enc},i})$ be the i-th hidden feature activation, where $W_{\mathrm{enc},i}$ is the i-th row of W_{enc} and $W_{\mathrm{dec},i}$ is the i-th column of W_{dec}.

To define the influence of one feature on another, the authors construct an attribution matrix $A \in \mathbb{R}^{d_{\mathrm{features}} \times d_{\mathrm{features}}}$, where each element $A_{i \to j}$ represents the strength of directed influence from feature f_i to feature f_j. The core formulation is:

$$A_{i \to j} := W_{\mathrm{dec},i}^{\top} W_{\mathrm{enc},j}.$$

This quantity can be interpreted geometrically as the alignment between the output vector generated by feature i and the input direction that activates feature j. A high dot product indicates that the output from feature i lies in the direction that would tend to activate feature j at the next layer, suggesting a directed computational pathway.

Crucially, this formulation is input-independent. However, it also admits a differential interpretation. Specifically, let x be the input to a transcoder at layer ℓ, and let $z = \mathrm{ReLU}(W_{\mathrm{enc}} x + b_{\mathrm{enc}})$ be the sparse feature vector. Then the Jacobian of the decoder output with respect to x is given by:

$$\frac{\partial \mathrm{TC}(x)}{\partial x} = W_{\mathrm{dec}} \cdot \mathrm{diag}(\mathbf{1}_{z>0}) \cdot W_{\mathrm{enc}},$$

where $\mathrm{diag}(\mathbf{1}_{z>0})$ is a diagonal matrix indicating which features are active (i.e., passed the ReLU threshold).

Now suppose we fix feature f_i to be active, i.e., $z_i > 0$. The partial derivative of the output with respect to x contributed by this feature is:

$$\frac{\partial \mathrm{TC}(x)}{\partial x}\Big|_{f_i} = W_{\mathrm{dec},i} \cdot W_{\mathrm{enc},i}.$$

Similarly, the influence of this output on the activation of another feature f_j in the next layer is proportional to:

$$\frac{\partial f_j(x')}{\partial x'} \cdot W_{\mathrm{dec},i} = \mathbf{1}_{f_j>0} \cdot W_{\mathrm{enc},j} \cdot W_{\mathrm{dec},i},$$

which is equivalent to the dot product $W_{\mathrm{dec},i}^{\top} W_{\mathrm{enc},j}$ when f_j is active. Thus, the attribution matrix A can be interpreted as a first-order approximation to inter-feature influence under the assumption that both features are active.

These theoretical insights justify the construction of the attribution graph as a weighted directed graph $G = (V, E)$, where each node corresponds to a feature and each edge weight is given by $A_{i \to j}$. Edges can be pruned by applying a magnitude threshold or top-k filter to focus analysis on the strongest influences. This graph structure enables interpretable circuit tracing, identifying hubs, chains, and feedback motifs in the flow of computation through the model. [1]

3 Project Goals

While our overarching goal of this project is to create a visualization tool more specific to Matryoshka SAE techniques, we also aim to benchmark other existing transcoder-based interpretability methods since there are no previous work that completes this goal. To do so, we summarize the main project milestones as follows:

- **Goal 1.** Build on top of Anthropic's existing **attribute-graphs-frontend** visualization tool to integrate existing Matryoshka SAE techniques by Bussman and Nabeshima ([3], [8]).
- **Goal 2.** Benchmark existing transcoder-based and Matryoshka SAE techniques, which would likely result in a novel survey paper.
- **Goal 3.** Based on the existing work on Matryoshka SAEs and cross-layer transcoders, we came up with a new Matryoshka-based interpretability technique called Matryoshka-CLT, which we will elaborate further in section 4.3. We will implement this technique and benchmark this against the prior existing techniques.
- **Goal 4.** Finally, we may want to add additional features that would improve the user experience of the visualization tool. Some extra features that we may want to add are allowing users the ability to break down abstract features into more fine-grained features, such as by allowing the user to pick prefix length of their prompt.

4 Methods

Recent advances in mechanistic interpretability have highlighted the need for scalable, human-centered visualization techniques. We propose a visualization method that extend current graph-based representations with interactive and accessible features.

4.1 Creating Specialized Visualization Tool for Matryoshka SAEs

A visualization web application of a responsive, interpretable interface that supports deep, real-time exploration of learned representations in Matryoshka-style models.

SAEs node attribution graph will be visualized, and activation trace of the nodes will be mapped in the Circuits-style interface, with showing input and output features.

4.2 Integrating Existing Matryoshka SAE Implementations

We will integrate existing Matryoshka SAE implementations as mentioned previously into our project. These implementations were given in the open-source repositories of Nabeshima and Bussman ([3], [8]). We will first ensure that we have used the correct implementations by reproducing results presented in the authors' papers and then benchmark these two implementations against one another using metrics which will be explained in the following sections.

4.3 Matryoshka-CLT: A Nested Variant of Cross-Layer Transcoders

We propose a Matryoshka-style modification of the Cross-Layer Transcoder (CLT) loss to encourage a hierarchical organization of the latent space, where earlier latent dimensions capture generalizable features shared across layers, and later dimensions specialize in residual or layer-specific information.

Recall that in the standard CLT, a shared encoder-decoder pair $(W_{\text{enc}}, W_{\text{dec}})$ is used across all transformer layers $\ell = 1, \ldots, L$:

$$f(x_\ell) = W_{\text{enc}} x_\ell + b_{\text{enc}}, \quad \hat{y}_\ell = W_{\text{dec}} \operatorname{ReLU}(f(x_\ell)) + b_{\text{dec}}.$$

The CLT loss is given by:

$$\mathcal{L}_{\text{CLT}} = \sum_{\ell=1}^{L} \|\hat{y}_\ell - y_\ell\|_2^2 + \lambda \sum_{\ell=1}^{L} \|\tanh(f(x_\ell))\|_1 .$$

To encourage nested information encoding, we define a set of prefix sizes $\mathcal{P} = \{p_1, \ldots, p_K\}$, where each p_k denotes a truncated latent dimensionality. For each prefix p_k, we compute:

$$\hat{y}_\ell^{(p_k)} = W_{\text{dec}}^{(p_k)} \operatorname{ReLU}(f(x_\ell)_{1:p_k}) + b_{\text{dec}}^{(p_k)},$$

where $f(x_\ell)_{1:p_k}$ denotes the first p_k dimensions of the encoder output, and $W_{\text{dec}}^{(p_k)}$ is a decoder specific to that prefix. The corresponding **Matryoshka-CLT loss** is:

$$\mathcal{L}_{\text{Mat-CLT}} = \sum_{\ell=1}^{L} \sum_{p_k \in \mathcal{P}} \left\| \hat{y}_\ell^{(p_k)} - y_\ell \right\|_2^2 + \lambda \sum_{\ell=1}^{L} \sum_{p_k \in \mathcal{P}} \|\tanh(f(x_\ell)_{1:p_k})\|_1 .$$

Although only the reconstruction from the full latent space is ultimately used in the replacement model, we train decoders for each prefix size to directly supervise intermediate subsets of the encoder. This ensures that early latents retain meaningful reconstruction fidelity, which is critical for learning abstract, generalizable features.

We plan to experiment with different strategies for generating prefix sizes, including fixed groupings (e.g., linear or geometric spacing), random sampling from distributions such as the truncated Pareto, and adaptive schemes based on entropy or gradient signal concentration. These design choices may affect the degree to which the encoder learns coarse-to-fine representations and influence both training stability and interpretability. Furthermore, it is relatively straightforward to extrapolate Nabeshima's delta trick for efficiently calculating loss to Matryoshka-CLT loss calculations if prefixes are drawn from some distribution.

4.4 Benchmarking Interpretability Techniques

In our project we will benchmark various existing transcoder-based and Matryoshka SAE approaches to interpretability. Also, after we implemented our technique Matryoshka-CLTs, we will benchmark our technique against the aforementioned existing techniques. It should be noted that evaluation metrics for benchmarking SAE implementations can naturally be extended to evaluate transcoder implementations and vice versa. Additionally, we will also utilize both quantitative and qualitative metrics, since our tool should not only provide mathematical insights but is also aimed to be user-friendly.

4.4.1 Benchmarking Matryoshka SAEs Qualitatively

We will benchmark existing Matryoshka SAEs qualitatively using the method proposed by Nabeshima. In Nabeshima's toy model experiment, the input data consists of vectors composed of 16 orthogonal ground-truth features [8]. The Matryoshka SAE is trained to reconstruct these vectors using sampled latent prefixes. After training, Nabeshima inspects the learned features by comparing the decoder vectors to the original ground-truth directions. A key question is whether individual latents learn independent atomic features (such as "red" or "circle") or instead specialize in memorizing composite combinations (such as "red circle"). The expectation is that under Matryoshka training, earlier-indexed latents should learn simpler and more reusable features, as they are used in reconstructions for a wider range of prefixes.

To answer this question, Nabeshima trained their Matryoshka SAE on residual stream activations from the GPT-2-Small model using OpenWebText [8]. Reconstructions are then computed using different prefix lengths, and the fidelity of each prefix is evaluated by measuring the qualitative accuracy of reconstructions. Smaller prefixes are expected to capture coarse structure, while later latents refine the reconstruction. Nabeshima includes visualizations that show how the quality of the reconstructed activations improves as more latents are included, validating that earlier latents encode general information and later latents add specificity.

4.4.2 BENCHMARKING MATRYOSHKA SAEs QUANTITATIVELY

To quantify reconstruction quality, both Nabeshima and Bussmann et al. plot the mean squared error $\text{MSE}(p)$ as a function of prefix length p ([3], [8]). The plot shows that Matryoshka SAEs achieve significantly better reconstruction performance at small prefix sizes than standard SAEs or models with randomly ordered latents.

Although Matryoshka SAEs slightly underperform standard BatchTopK SAEs in terms of total reconstruction error, Bussmann et al. find that they perform comparably on downstream cross-entropy loss for next-token prediction [3]. This suggests that despite reduced reconstruction fidelity, the features learned by Matryoshka SAEs retain meaningful semantic information used by the language model.

To evaluate latent quality more directly, they measure feature absorption using the first-letter classification task from SAEBench [7], showing that Matryoshka SAEs exhibit dramatically lower absorption rates than standard SAEs—remaining as low as 0.03 compared to over 0.29 in BatchTopK models at large dictionary sizes [3]. They also evaluate feature splitting, finding that Matryoshka SAEs tend to represent conceptual features in a more unified way, whereas BatchTopK SAEs often fragment these across multiple latents.

Further, they assess latent composition using meta-SAEs trained to reconstruct the decoder matrices of the original SAEs [3]. Lower reconstruction quality from meta-SAEs indicates that Matryoshka latents are more disentangled.

Finally, they use a suite of sparse probing metrics from SAEBench [7]. While Matryoshka SAEs show mixed results on k-sparse probing, they outperform BatchTopK SAEs on Targeted Probe Perturbation (TPP) and Spurious Correlation Removal (SCR), indicating improved modularity and causal disentanglement of features. An automated interpretability benchmark using LLM-based feature explanation and scoring finds a slight advantage for early Matryoshka subgroups in terms of interpretability, though this benefit does not extend to later latent groups. These results together suggest that Matryoshka SAEs achieve stronger structural disentanglement at the cost of slightly worse raw reconstruction.

4.5 BENCHMARKING TRANSCODER IMPLEMENTATIONS QUANTITATIVELY

To evaluate the quality of their transcoder architectures, the Anthropic team employed a series of quantitative metrics that reflect both functional fidelity and representational efficiency. In addition to standard measures such as top-1 accuracy—used to verify whether the token predicted by the transcoder-altered model matches that of the original—they computed KL divergence between the full logit distributions of the two models.

Additionally, they assessed structural similarity through logit correlation, calculating the Pearson correlation between original and reconstructed logit vectors. This metric is less sensitive to softmax-induced top-token dominance and instead captures alignment in the overall shape of the output space. To measure how well the transcoders replicated internal computations, they computed mean squared error (MSE) between the MLP output vectors of the original model and those generated by the transcoder.

They also evaluated the sparsity and consistency of the learned latent features. The ℓ_1 norm of tanh-transformed encoder outputs served as a direct proxy for feature sparsity, which is essential for interpretability. The use of a $\tanh$ activation prior to regularization ensured that the sparsity loss did not dominate gradients when activations became large, a common problem with ReLU or raw pre-activation penalties. Furthermore, the frequency with which each latent feature activated across the dataset was measured to ensure that a small number of features were not overused, nor that features were so specific that they only activated for a narrow set of inputs.

To assess feature importance more directly, the authors conducted ablation experiments by ranking latent features according to their global decoder weights. These weights, which quantify how much a feature contributes to the model's final output logits, were used to iteratively ablate features and track the resulting drop in performance. The observation that a small number of high-weight features accounted for a disproportionate share of predictive capacity lent support to the idea that the transcoders had discovered a compact and meaningful feature basis.

4.6 BENCHMARKING TRANSCODER IMPLEMENTATIONS QUALITATIVELY

In addition to quantitative metrics, the Anthropic team developed a suite of qualitative interpretability analyses to probe the semantic coherence and causal structure of the learned feature space. One major axis of evaluation was the extent to which individual latent features were monosemantic. This was tested using prompt design and activation maximization techniques, where specific inputs were engineered to selectively activate individual features. If a feature could be reliably elicited by a semantically consistent class of inputs, it was considered monosemantic.

To trace how features influence one another across layers, the authors constructed circuit attribution graphs as described in section 2.6. In these graphs, nodes correspond to latent features and directed edges represent functional influence from one feature to another. This influence was defined as the dot product between the decoder vector of a source feature and the encoder vector of a destination feature. Because this value is independent of any particular input, it allowed them to identify input-invariant structural connections between features. The resulting graphs enabled them to visualize computational pathways and identify causal substructures, such as fan-in, fan-out, or symmetric feature relationships.

Feature-to-token attribution analyses were also conducted to determine how specific features contributed to output token probabilities. By decomposing the logit generation process via the shared decoder and output projection matrix, they could trace the influence of individual features on specific vocabulary logits. This grounded the abstract latent features in observable language behavior and provided an avenue for human verification.

Perhaps the most ambitious qualitative evaluation came from constructing the replacement model mentioned earlier, in which every MLP sublayer was removed and replaced with its corresponding transcoder [1]. This model was evaluated using the same quantitative metrics as before, but its construction also served as a proof of concept for the completeness and expressiveness of the learned features. Furthermore, the cumulative feature accumulation strategy described earlier composition proved beneficial in reconstructing the contextual richness of the original model, showing that the learned feature space supported not just local approximations, but a form of global compositionality as well.

5 COMPUTATIONAL COST

Training a 2-billion-parameter transformer model on a dataset with 2M features and 1B training tokens is estimated to require 3.8×10^{20} floating-point operations (FLOPs) [1]. With training 3 CLTs, generate attribution graphs for interpretability, validation, and pruning, the estimated load would be 1.5FLOPs.

Considering the use of 8 NVIDIA A100 80GB GPUs in parallel, with 312 TFLOPs/sec per device [9] under FP16 training and 85% efficiency, the total training time is approximately 50 hours. At an average cloud computing cost of $4 per GPU-hour, the total estimated cost for the training run is approximately $2,400. In the deployment phase, the inference back end and the hosing front end are estimated to cost up to $1400 per month.

Table 1: Computational Cost Estimation

Category	**Component Description**	**Est. Cost ($)**
Training Phase (One-Time Cost)		
Transcoders	GPU usage to train on 1B tokens (8×A100, 85% efficiency)	1,600
CLT	Cross-layer Transcoders	256
Attribution Graphs	Describe the steps a model outputs	344
Validation	In-training evaluation	88
Pruning	Lightweight retraining	112
Total Training Cost		**2,400**
Deployment Phase (Ongoing Monthly Cost)		
Inference Backend	GPU server for real-time inference API	400–1,000
Front end Hosting	Deployment on Vercel or Netlify	0–20
API Gateway	Request routing and load balancing	10–50
Monitoring	Logs, alerts, metrics tracking	20–200
Cloud Storage	Storage for artifacts or visualizations	10–100
Domain	Custom domain and SSL certificate	10–30
Total Deployment (Monthly)		**450–1,400**

6 EXPECTED RESULTS

We expect our framework to yield several important outcomes for the field of mechanistic interpretability.

First of all, we expect our implementation of Matryoshka SAEs and CLTs will yield a responsive, interpretable interface website application. Website includes SAE token-level attribution graph, input and output features, graph tracing flow chart, top activations list.

Interactive synchronization in multiple view will be implemented. For example, clicking on a node within SEAs attribution graph will lead to updates input and output features list and node tracing flow chart.

The website includes an interactive fronted, and a backend providing precomputed SAEs and CLTs. In frontend, React for modular UI componets, with D3.js and Three.js used for attribution graph. React hooks will be enable synchronized real-time updates. Zooming and panning will be handled with WebGL. Lightweight backend will be implemented by FlaskAPI.

7 SCHEDULE OF WORK

7.1 STAGE 1

- **Objective**: Literature Review. Key questions that we want to address: what is circuit analysis? what are the differences between SAEs, Transcoders, and Skip Trasncoders? How will we evaluate/validate our work? How will we benchmark our transcoders?
- **Duration**: Week 1

7.2 STAGE 2

- **Objective**: Build on the existing open-source backend logic to compute attribution graphs from trained SAEs, transcoders. exploration.
- **Duration**: Week 2-4

7.3 STAGE 3

- **Objective**: Begin designing the visualization pipeline for circuit exploration. This step is partially dependent On stage 2 model outputs.
- **Duration**: Week 3-6. Will probably begin after stage 2 starts but might occur concurrently.

7.4 STAGE 4

- **Objective**: Build the browser-based interface, including interactive graph visualizations, toggle tools, and prompt comparison views. This will be concurrent with Stages 3 and 5, and partially dependent on output format from Stage 3.
- **Duration**: Week 5-9

7.5 STAGE 5

- **Objective**: Validate the quality and coherence of generated computational graphs. Replicate known circuits (e.g., induction heads) and ensure accuracy across prompts. This will require functional attribution graph generation (Stage 3) and SAE outputs (Stage 2).
- **Duration**: Week 6-9

7.6 STAGE 6

- **Objective**: Run final benchmarks comparing SAEs, transcoders. Conduct case studies and prepare materials for open-source release.
- **Duration**: Week 10-12

REFERENCES

[1] Emmanuel Ameisen et al. "Circuit Tracing: Revealing Computational Graphs in Language Models". In: *Transformer Circuits Thread* (2025). URL: https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

[2] *attribution-graphs-frontend*. URL: https://github.com/anthropics/attribution-graphs-frontend.

[3] Bart Bussmann, Patrick Leask, and Neel Nanda. *Learning Multi-Level Features with Matryoshka SAEs*. Dec. 2024. URL: https://www.lesswrong.com/posts/rKM9b6B2LqwSB5ToN/learning-multi-level-features-with-matryoshka-saes.

[4] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. "Transcoders Find Interpretable LLM Feature Circuits". In: *Advances in Neural Information Processing Systems* (2024). URL: https://github.com/jacobdunefsky/transcoder_circuits.

[5] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. *Attribution Graphs via Transcoders*. Transformer Circuits Thread. 2025. URL: https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

[6] *EleutherAI/clt-training: Sparsify transformers with cross-layer transcoders*. URL: https://github.com/EleutherAI/clt-training.

[7] Adam Karvonen et al. *SAEBench: A Comprehensive Benchmark for Sparse Autoencoders in Language Model Interpretability*. 2025. arXiv: 2503.09532 [cs.LG]. URL: https://arxiv.org/abs/2503.09532.

[8] Noa Nabeshima. *Matryoshka Sparse Autoencoders*. Dec. 2024. URL: https://www.lesswrong.com/posts/zbebxYCqsryPALh8C/matryoshka-sparse-autoencoders#fn-gTupTd3B3GQegCM2j-4.

[9] *NVIDIA A100 Tensor Core GPU*. 2023. URL: https://www.nvidia.com/en-us/data-center/a100/.

[10] Gonçalo Paulo, Stepan Shabalin, and Nora Belrose. *Transcoders Beat Sparse Autoencoders for Interpretability*. 2025. arXiv: 2501.18823 [cs.LG]. URL: https://arxiv.org/abs/2501.18823.

[11] Adly Templeton et al. *Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet*. May 2024. URL: https://transformer-circuits.pub/2024/scaling-monosemanticity/.