LLAMA SCOPE: EXTRACTING MILLIONS OF FEATURES FROM LLAMA-3.1-8B WITH SPARSE AUTOENCODERS

Zhengfu He[1,2] **Wentao Shu**[1] **Xuyang Ge**[1]
Lingjie Chen[1] **Junxuan Wang**[1] **Yunhua Zhou**[1,3] **Frances Liu**
Qipeng Guo[1,2,3] **Xuanjing Huang**[1] **Zuxuan Wu**[1] **Yu-Gang Jiang**[1] **Xipeng Qiu**[1,2]

[1]Fudan University
[2]Shanghai Innovation Institute
[3]Shanghai Artificial Intelligence Laboratory

{zfhe19, xpqiu}@fudan.edu.cn

ABSTRACT

Sparse Autoencoders (SAEs) have emerged as a powerful unsupervised method for extracting sparse representations from language models, yet scalable training remains a significant challenge. We introduce a suite of 256 SAEs, trained on each layer and sublayer of the Llama-3.1-8B-Base model, with 32K and 128K features. Modifications to a state-of-the-art SAE variant, Top-K SAEs, are evaluated across multiple dimensions. In particular, we assess the generalizability of SAEs trained on base models to longer contexts and fine-tuned models. Additionally, we analyze the geometry of learned SAE latents, confirming that *feature splitting* enables the discovery of new features. The Llama Scope SAE checkpoints are publicly available at https://huggingface.co/fnlp/Llama-Scope, alongside our scalable training, interpretation, and visualization tools at https://github.com/OpenMOSS/Language-Model-SAEs. These contributions aim to advance the open-source Sparse Autoencoder ecosystem and support mechanistic interpretability research by reducing the need for redundant SAE training.

1 INTRODUCTION

Mechanistic interpretability has long grappled with the challenge of identifying interpretable primitives within language models. Despite this, researchers have demonstrated that the structure of network representations tends to be linear, sparse, and decomposable (Mikolov et al., 2013; Olah et al., 2020; Elhage et al., 2022b; Gurnee et al., 2023). Ideally, the model's components, such as neurons and attention heads, would correspond directly to interpretable features of the input. However, due to superposition (Arora et al., 2018; Elhage et al., 2022b) and the misalignment of linear features with neuron bases (Elhage et al., 2023), this is not typically the case.

Sparse Autoencoders (SAEs) (Bricken et al., 2023; Huben et al., 2024; Templeton et al., 2024b) offer a promising approach to addressing superposition. The features extracted by SAEs exhibit high monosemanticity and causal relevance, allowing them to capture significantly more features than neuron-based methods in pretrained Transformer models(Olah et al., 2020; Bills et al., 2023; Huben et al., 2024). This technique aids in identifying the latent variables within neural networks, providing anchor points for reverse engineering. These features may also prove useful in addressing model hallucination and mitigating safety-relevant behaviors.

Despite these advances, research in this area remains somewhat disconnected from industrial-scale language models. While recent efforts have begun training and investigating SAEs on models exceeding 8 billion parameters (Templeton et al., 2024b; Engels et al., 2024; Gao et al., 2024; Lieberum et al., 2024), rigorous interpretability research on such large models is still in its early stages. Com-

prehensive analyses often require SAEs trained across multiple sites (e.g., for SAE-based circuit analysis (He et al., 2024; Marks et al., 2024; Ge et al., 2024)) or trained to handle multiple feature sizes (e.g., for feature splitting (Bricken et al., 2023) and identifying ultra-rare features (Templeton et al., 2024b)).

To address these challenges, we introduce Llama Scope, a suite of 256 Sparse Autoencoders trained on Llama-3.1-8B, a widely used open-source large language model. Llama Scope aims to facilitate research in mechanistic interpretability by providing ready-to-use, open-source SAE models, reducing the need for extensive retraining. We believe that this open-source ecosystem will serve as a *common language* for researchers to communicate and share insights across the field.

The main contributions of this work are highlighted as follows:

Decomposing Every Activation Space of Llama-3.1-8B-Base. We train SAEs on every sublayer, namely post-MLP residual stream, attention output, MLP output, and Transcoders, for all 32 layers with 32K and 128K feature widths (Section 3.2). This results in 256 SAEs in total. Such a comprehensive suite of SAEs trained on industrial-scale language models opens up new possibilities for both interpretability and LLM research. Following Lieberum et al. (2024), we list a number of exciting research directions that can be pursued with this suite in Section 5.

Improved TopK SAEs. We make several modifications to the Top-K SAEs, including incorporating the 2-norm of the decoder columns into the TopK computation, post-processing TopK SAEs to JumpReLU variants, and introducing a K-annealing training schedule (Section 2.4).

Comprehensive Evaluation. Llama Scope SAEs are evaluated using a range of metrics, including canonical metrics like sparsity-fidelity trade-off, latent firing frequency, and feature interpretability. We also assess the generalizability of SAEs trained on base models to longer contexts and fine-tuned models and analyze the geometry of learned SAE latents (Section 4).

Disk-IO Friendly Training Infrastructure. A mixed parallelism approach is employed to train SAEs with a large number of features, which significantly reduces the memory bottleneck of SAE training (Appendix A).

2 Conceptual and Technical Background

2.1 Attacking Superposition with Sparse Autoencoders

The Sparse Autoencoder approach is motivated by the superposition hypothesis (Arora et al., 2018; Elhage et al., 2022b), which posits that the representations learned by neural networks are composed of independent features with the following properties:

- **Decomposability**: High-dimensional representations can be expressed as a combination of independent, interpretable features.
- **Linearity**: Features are represented linearly, meaning the direction of each feature indicates the presence of a specific concept, while its magnitude reflects the concept's strength.
- **Sparsity**: The decomposition is sparse, with only a few features active at any given time.
- **Overcompleteness**: The number of underlying features exceeds the dimensionality of the representation.

This hypothesis combines several well-established concepts from the literature. For instance, in Olshausen & Field (1996), it has been shown that images can be sparsely represented by an overcomplete basis set of Gabor functions. Similarly, Mikolov et al. (2013) showed that word embeddings exhibit a degree of linear structure, exemplified by relationships such as $v_{\text{king}} - v_{\text{queen}} \approx v_{\text{man}} - v_{\text{woman}}$. Linear probing (Alain & Bengio, 2017; Gurnee et al., 2023) also reflects this hypothesis, as it assumes the linearity of features in neural networks.

Despite some modifications (Engels et al., 2024) and counterexamples (Csordás et al., 2024), the superposition hypothesis remains a valuable framework of how individual features are represented

in neural networks. To overcome the superposition problem, the field of mechanistic interpretability has been actively developing methods to extract sparse and linear features from neural networks (Elhage et al., 2022a; Bricken et al., 2023), with Sparse Autoencoders being one of the most prominent approaches.

2.2 Vanilla Sparse Autoencoders

Sparse Autoencoders are designed to reverse the effects of superposition by extracting features that are sparse, linear, and decomposable.

A Sparse Autoencoder typically consists of a single hidden layer. The input $\mathbf{x}$ is linearly mapped to a hidden layer $f(\mathbf{x})$ followed by a non-linear activation function.

$$f(\mathbf{x}) = \text{ReLU}(W^{enc}\mathbf{x} + b^{enc}), \tag{1}$$

Where $W^{enc} \in \mathbb{R}^{F \times D}$ is the weight matrix and $b^{enc} \in \mathbb{R}^F$ is the bias vector. The hidden layer is then linearly mapped back to the input space to reconstruct the input $\hat{\mathbf{x}}$ with $W^{dec} \in \mathbb{R}^{D \times F}$ and a bias term $b^{dec} \in \mathbb{R}^D$:

$$\hat{\mathbf{x}} = W^{dec} f(\mathbf{x}) + b^{dec}. \tag{2}$$

To ensure that the extracted features are sparse, a sparsity constraint is applied to the hidden layer. A common approach is L1 regularization, which encourages fewer active features while minimizing reconstruction error:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \mathcal{L}_{\text{Sparsity}} = \|x - \hat{x}\|_2 + \lambda \sum_{i=1}^{F} \|f_i(\mathbf{x})\|_1, \tag{3}$$

Conceptually, the encoder decides which features are activated and to what degree, while the decoder maps these features back to the input space. The decoder columns $W^{dec}_{:,i}$ form an overcomplete basis for the latent space, where each column's direction corresponds to a specific feature.

There are two primary ways to interpret the features extracted by SAEs. The *encoder view* focuses on feature activations, identifying which inputs activate particular features and to what extent. The *decoder view*, by contrast, interprets the basis formed by the decoder columns, revealing the geometric structure of the latent space. This perspective provides insight into how neural networks encode complex information in a compressed form.

2.3 Top-K Sparse Autoencoders

We follow the approach of Gao et al. (2024), which introduced Top-K Sparse Autoencoders (SAEs) as an improvement over vanilla SAEs (Section 4.2), demonstrating better performance on canonical metrics such as the sparsity-fidelity trade-off (Gao et al., 2024), as well as on novel evaluation metrics proposed by Templeton et al. (2024a).

Top-K SAEs enforce sparsity by selecting only the K most active features, $f_i(\mathbf{x})$, for reconstruction, setting the remaining features to zero. The hidden layer of the SAE is thus defined as:

$$f_i(\mathbf{x}) = \text{TopK}(\text{ReLU}(W^{enc}_{i,:}\mathbf{x} + b^{enc}_i)), i \in \{1, 2, \ldots, F\}, \tag{4}$$

The units in the hidden layer, $f_i(\mathbf{x})$, are referred to as *features*, representing the magnitude of the corresponding latent interpretability primitives.

As in the vanilla SAE, the hidden layer is then mapped back to the input space using a decoder matrix as in Equation 2. In addition, the decoder columns $W^{dec}_{:,i}$ are set to unit 2-norm after each minibatch to prevent the network from shrinking feature activations and compensating by increasing the norm of the corresponding decoder columns.

The training objective is to minimize the reconstruction error, which is simply the mean squared error: $\mathcal{L} = \mathcal{L}_{\text{MSE}} = \|x - \hat{x}\|_2$.

2.4 Modifications to Top-K SAEs

We introduce several modifications to the Top-K SAEs. Notably, we do not apply any auxiliary loss to *revive* dead features, as this is rarely an issue in our SAEs.

Incorporating the 2-norm of Decoder Columns in Top-K Computation. One key modification involves incorporating the 2-norm of the decoder columns into the Top-K computation to avoid normalizing decoder columns $W^{dec}_{:,i}$ to unit 2-norm after each minibatch, which adopts the same motivation as Templeton et al. (2024b). Specifically, we redefine the activation of a feature, $f_i(\mathbf{x})$, as the product of the feature's activation and the 2-norm of the corresponding decoder column, $\|W^{dec}_{:,i}\|_2$. The hidden layer is thus computed as:

$$\begin{aligned} \text{Mask} &= \text{TopK}\left(\text{ReLU}(W^{enc}\mathbf{x} + b^{enc}) \cdot \begin{bmatrix} \|W^{dec}_{:,1}\|_2 \\ \|W^{dec}_{:,2}\|_2 \\ \vdots \\ \|W^{dec}_{:,F}\|_2 \end{bmatrix}\right), \\ f_i(\mathbf{x}) &= \text{Mask}_i \cdot (\text{ReLU}(W^{enc}_{i,:}\mathbf{x} + b^{enc}_i)), i \in \{1, 2, \ldots, F\}. \end{aligned} \tag{5}$$

This modification also avoids the need to prune gradients parallel to the decoder columns in the Adam optimizer, which was previously required to prevent suboptimal training outcomes (Bricken et al., 2023). We conjecture that pruning gradients in this manner introduces noise into the Adam momentum, leading to less efficient SAE training. By incorporating the 2-norm into the Top-K computation, we achieve the same sparsity enforcement without the need for additional pruning.

It is important to note that during training, the 2-norm of the decoder columns is not fixed to 1, which can conflict with its interpretation as the direction of the feature (Section 2.2). We address this issue at inference time, as discussed in Section 3.3.4.

Post Processing TopK SAEs to JumpReLU Variants. Another modification is applied post-training. We introduce a threshold, θ, to ensure that, on average, K features are activated across the training distribution, rather than exactly K features being activated for every input (Templeton et al., 2024a). Features with activations exceeding this threshold retain their values, while those below are set to zero. This approach actually mirrors the JumpReLU activation function (Erichson et al., 2020; Rajamanoharan et al., 2024) with its threshold set to θ. Further details can be found in Section 3.3.4.

This variant combines the advantages of both Top-K and JumpReLU activations. Training with a predetermined L0 is more intuitive, as it allows researchers to set a desired L0 and observe the reconstruction loss during convergence. Additionally, thresholding during inference decouples the activation of features, meaning each feature is evaluated independently. This avoids scenarios where a feature remains inactive merely because other features are more strongly activated.

K-Annealing Training Schedule. In the early stages of training, we find it beneficial to gradually reduce the number of activated features from D to K over the first 10% of training steps. This K-annealing schedule improves convergence by allowing the model to adjust more smoothly to sparse activations. We refer readers to Section 3.3.2 for details.

3 Training Llama Scope SAEs

Table 1 provides a broad overview of Llama Scope SAE suite, along with a comparison to recent work on training Sparse Autoencoders on models with more than 8 billion parameters (Templeton et al., 2024b; Gao et al., 2024; Lieberum et al., 2024).

	Llama Scope	Scaling Monosemanticity	GPT-4 SAE	Gemma Scope
Models	Llama-3.1 8B (Open Source)	Claude-3.0 Sonnet (Proprietary)	GPT-4 (Proprietary)	Gemma-2 2B & 9B (Open Source)
SAE Training Data	SlimPajama	Proprietary	Proprietary	Proprietary, Sampled from Mesnard et al. (2024)
SAE Position (Layer)	Every Layer	The Middle Layer	$\frac{5}{6}$ Late Layer	Every Layer
SAE Position (Site)	R, A, M, TC	R	R	R, A, M, TC
SAE Width (# Features)	32K, 128K	1M, 4M, 34M	128K, 1M, 16M	16K, 64K, 128K, 256K - 1M (Partial)
SAE Width (Expansion Factor)	8x, 32x	Proprietary	Proprietary	4.6x, 7.1x, 28.5x, 36.6x
Activation Function	TopK-ReLU	ReLU	TopK-ReLU	JumpReLU

Table 1: An overview of existing work training SAEs on Language Models with more than 8B parameters.

3.1 Collecting Activations

Llama Scope SAEs are trained on activations generated from Llama-3.1-8B using text data sampled from SlimPajama (Soboleva et al., 2023) with the proportions of each subset (e.g., Commoncrawl, C4, Github, Books, Arxiv, StackExchange) preserved. All activations and computations are performed using bfloat16 precision to optimize memory usage without sacrificing accuracy, as also demonstrated in Lieberum et al. (2024) where bfloat16 and float32 activations are shown to have similar performance.

Each document is truncated to 1024 tokens, and a <beginoftext> token is prepended to the start of every document. During training, the activations of the <beginoftext> and <endoftext> tokens are excluded from use.

It has been shown that the activations of different sequence positions of the same document tend to be highly correlated (Bricken et al., 2023), potentially reducing the diversity of the training data. To address this, randomness is introduced by employing a buffer shuffling strategy: the buffer is refilled and shuffled at regular intervals (Appendix A.1).

3.2 Training Position

We train SAEs at each layer and sublayer of Llama-3.1-8B. Specifically, there are four potential training positions:

Llama-3.1-8B consists of 32 layers, resulting in 128 possible training positions when considering all four options. For each training position, SAEs are trained with 8x (32K features) and 32x (128K features) the dimension of the hidden size of Llama-3.1-8B.

Throughout this paper, we adopt the naming convention introduced by He et al. (2024) to label the SAEs as `L[Layer][Position]-[Expansion]x-[TopK,Vanilla]`. For example, a Top-K SAE trained on the post-MLP residual stream of layer 1 in Llama-3.1-8B, with an 8x expansion of the hidden size is named as `L1R-8x-TopK`.

3.3 Training Details

3.3.1 Initialization

The decoder columns $W^{dec}_{:,i}$ are initialized with Kaiming uniform (He et al., 2016) and normalized to have 2-norm of $\sqrt{\frac{2D}{F}} = \sqrt{\frac{2}{\text{expansion factor}}}$. The encoder weights W^{enc} are initially set to the

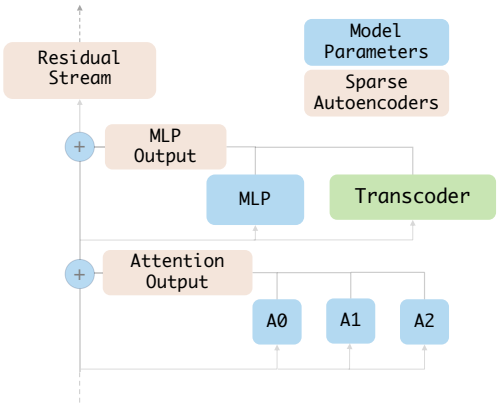

Figure 1: Four potential training positions in one Transformer Block.

- **Post-MLP Residual Stream (R):** The residual stream after each Transformer block, which is the overall result of computations before this layer.
- **Attention Output (A):** The output of each attention layer.
- **MLP Output (M):** The output of each MLP layer.
- **Transcoder (TC):** A sparse approximation of the MLP block designed for circuit analysis (Ge et al., 2024; Dunefsky et al., 2024). It differ from SAEs in that it takes in the layer-normalized residual stream as input and attempts to predict MLP output.

transpose of the decoder weights W^{dec}, though their parameters are untied after initialization. Both the encoder bias b^{enc} and decoder bias b^{dec} are initialized to zero.

This initialization strategy allows the SAE to start with a near-zero reconstruction loss, a widely observed benefit in SAE training (Templeton et al., 2024b; Gao et al., 2024; Lieberum et al., 2024).

For transcoders, the encoder weights are not initialized as the transpose of the decoder weights, as the input and output follow different distributions.

3.3.2 Optimization

We train the SAEs using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The learning rate is set to 8e-4 for all SAEs, with a warm-up from 0 to the target rate over the first 10K steps, followed by a linear decay to 0 during the final 20% of training steps[1].

We also observed that fixing the number of activated features to K throughout training can result in some features not being activated until halfway of the training process. To address this, we gradually reduce the number of activated features from D to K during the first 10% of training steps. This approach, similar to L1 coefficient warmup in L1 regularization (Templeton et al., 2024b), ensures more even feature activation throughout the training.

3.3.3 Input and Output Normalization

The 2-norm of the input $\|\mathbf{x}\|_2$ varies across different positions. To ensure consistent hyperparameters, we normalize the input $\mathbf{x}$ to have a 2-norm of $\sqrt{D}$ before passing it into the SAE, and the reconstruction loss is computed with the same normalization. For Transcoders, we normalize both the input and output to have a 2-norm of $\sqrt{D}$.

Formally, the input normalization is performed as follows:

$$\mathbf{x}_{\text{in}} \leftarrow \mathbf{x}_{\text{in}} \cdot S_{in},$$
$$\mathbf{x}_{\text{out}} \leftarrow \mathbf{x}_{\text{out}} \cdot S_{out},$$

where S_{in} and S_{out} are the scalar normalization factors defined as $S_{in} = \sqrt{D}/\mathbb{E}(||\mathbf{x}_{\text{in}}||_2)$ and $S_{out} = \sqrt{D}/\mathbb{E}(||\mathbf{x}_{\text{out}}||_2)$. These two factors differ only when training transcoders.

3.3.4 Post-Training Processing

After training, we rescale the learned SAE weights to account for the input and output normalization that was applied during training. This is done using the following transformations:

[1]We find that learning rate decay significantly benefits SAE training.

$$
\begin{aligned}
W_{:,i}^{dec} &\leftarrow \frac{W_{:,i}^{dec}}{S_{out}} \cdot S_{in}, \\
b^{dec} &\leftarrow \frac{b^{dec}}{S_{out}}, \\
b^{enc} &\leftarrow \frac{b^{enc}}{S_{in}}.
\end{aligned}
$$

For standard SAEs, where $S_{in} = S_{out}$, the decoder weights remain unchanged.

To simplify analysis, we further adjust the decoder columns $W_{:,i}^{dec}$ to have unit 2-norm. This ensures that each column indicates only the direction of the corresponding feature, not its strength. The encoder weights and bias are rescaled accordingly:

$$
\begin{aligned}
W_{:,i}^{dec} &\leftarrow \frac{W_{:,i}^{dec}}{\|W_{:,i}^{dec}\|_2}, \\
W_{i,:}^{enc} &\leftarrow \|W_{:,i}^{dec}\|_2 \cdot W_{i,:}^{enc}, \\
b_i^{enc} &\leftarrow \|W_{:,i}^{dec}\|_2 \cdot b_i^{enc}
\end{aligned}
$$

After these two post-processing steps, the SAE operates on the original input and attempts to reconstruct the original output. The decoder columns, now with unit 2-norm, represent the direction of each feature, while the hidden layer encodes the magnitude of the corresponding feature.

4 Evaluation

We evaluate all our SAEs after post-training processing (Section 3.3.4), focusing primarily on two key aspects: how our SAEs advance the Pareto frontier of L0 sparsity and MSE efficiency (Section 4.2), and the interpretability of the features extracted with both automated and manual analysis (Section 4.3).

While our main focus is on these two areas, there are several other potential evaluation approaches. These include assessing the causal relevance of features to one another, as explored in circuit analysis (He et al., 2024; Ge et al., 2024; Dunefsky et al., 2024), conducting fine-grained causal analysis with counterfactuals (Huang et al., 2024), and performing automated prompt-based evaluations across a wide range of tasks (Templeton et al., 2024a). We plan to explore these and emerging methods as the field of SAE evaluation continues to develop.

Additionally, we investigate whether the features learned by state-of-the-art SAE variants differ from those learned by vanilla SAEs (Section 4.6). We also assess the out-of-distribution generalization of Llama Scope SAEs across sequence length and to instruct-finetuned models (Section 4.5).

4.1 Data and Metrics

We evaluate the SAEs using three primary metrics: L0-norm of SAE latents, explained variance, and Delta LM loss. These metrics assess sparsity, reconstruction quality and the impact of the SAEs on the language model's performance, respectively.

Let $\hat{x}$ be the SAE's reconstruction of the input x, derived from Equation 2 and 5. The explained variance (EV) measures the proportion of variance in the input captured by the SAE and is defined as:

$$
\text{EV} = 1 - \frac{\|x - \hat{x}\|_2^2}{\|x\|_2^2}. \tag{6}
$$

In addition to reconstruction quality, we measure the impact of the SAE on the language model with Delta LM loss. This metric is defined as the difference between the original language model loss

and the loss when the SAE is inserted at the corresponding position. Unlike explained variance and reconstruction MSE, which focus solely on the accuracy of the SAE's reconstruction, Delta LM loss reflects the effect of the SAE on overall language model performance.

All evaluations are performed over 50 million tokens of held-out text data, sampled from the same distribution as the training data.

4.2 Sparsity-Fidelity Pareto Efficiency

TopK SAEs Outperform Vanilla SAEs. As shown in Figure 2, TopK SAEs consistently outperform vanilla SAEs in both L0 sparsity and MSE efficiency when trained on each quartile layer's post-MLP residual stream. Holding all other hyperparameters constant, TopK SAEs reduce L0 sparsity from around 150 to 50, while maintaining or improving both explained variance and Delta LM loss. This improvement is likely due to TopK SAEs (1) mitigating the feature shrinkage issue, and (2) removing weakly firing features.

Wider SAEs Achieve Better Pareto Efficiency. Figure 2 also shows that wider SAEs outperform narrower ones in reconstruction while maintaining the same L0 sparsity. This suggests that, based on these high-level metrics, there is a positive correlation between the number of features and the overall quality of the SAEs. However, it is possible that wider SAEs simply learn more frequent compositions of existing features rather than discovering entirely new ones (Anders et al., 2024). We provide a counterexample to this in Section 4.6.

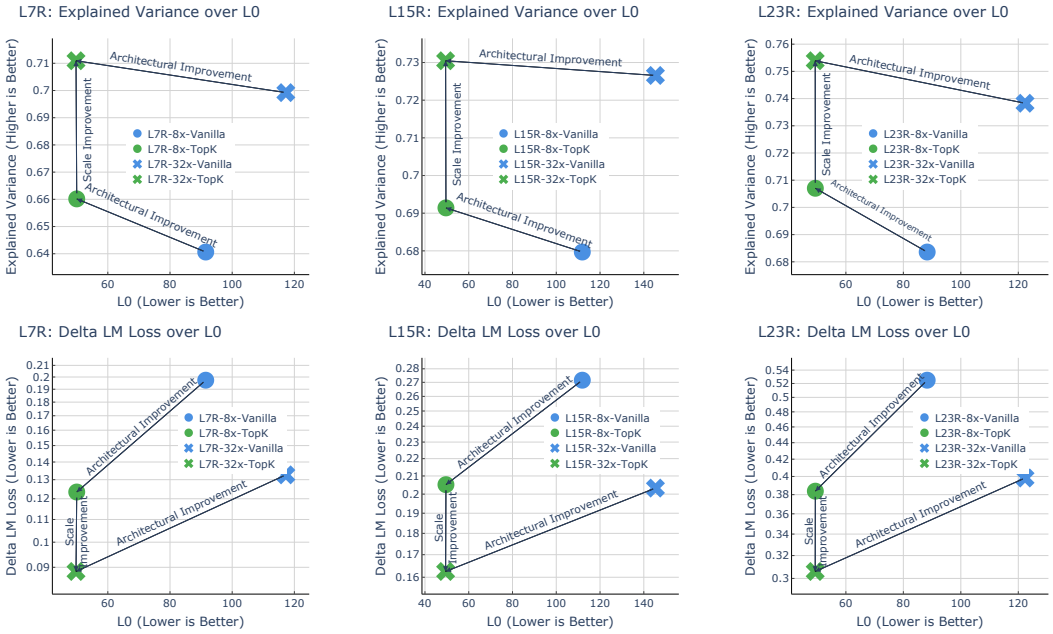

Figure 2: Explained Variance (upper) and Delta LM loss (lower) over L0 sparsity for SAEs trained on L7R, L15R and L23R.

Overall Assessment. Evaluation results for all 256 SAEs are illustrated in Figure 10 in Appendix B. Across all positions, layers and widths, TopK SAEs achieve consistently match or slightly exceed the reconstruction quality of vanilla SAEs, while achieving significantly better L0 sparsity. Wider SAEs show lower Delta LM loss and higher explained variance, all while maintaining the same L0 sparsity.

Residual stream SAEs (LXR-8x and LXR-32x) generally outperform those trained on other positions in terms of the aforementioned 3 metrics. We conjecture this is due to Cross Layer Superposition (Olah & Jermyn, 2024) where the output of each model component only form a small fraction to certain features, potentially introducing noise into the SAE training process.

4.3 Interpretability of Features

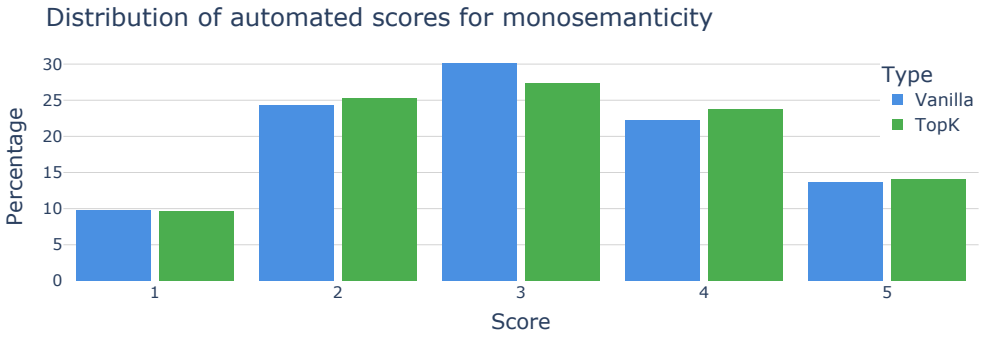

Figure 3: Automatically labeled monosemanticity scores of L15R-8x SAE features.

We assess the interpretability of the features extracted by SAEs using automated analysis, following the approach of Bills et al. (2023); Cunningham & Conerly (2024). For each feature, the 20 most activating contexts are sent to GPT-4o to generate a description of the feature. GPT-4o also scores each feature's monosemanticity, from 1 (Not comprehensible) to 5 (Clear and consistent pattern), using the rubric adapted from Cunningham & Conerly (2024):

- **5**: Clear pattern with no deviating examples;
- **4**: Clear pattern with one or two deviating examples;
- **3**: Clear overall pattern but quite a few examples not fitting that pattern;
- **2**: Broad consistent theme but lacking structure;
- **1**: No discernible pattern.

We randomly selected 128 features from the L7R-8x, L15R-8x, and L23R-8x SAEs, for both TopK and Vanilla SAEs, resulting in a total of 768 features. The monosemanticity results, shown in Figure 3, indicate no significant difference between TopK and Vanilla SAEs.

Additionally, we performed manual analysis on a subset of these features. The manual evaluation is consistent with the automated analysis, finding that about 10% of the features are not interpretable (monosemanticity score of 1). We did not further distinguish between scores of 2-5, as it is difficult to draw clear distinctions between these levels of interpretability.

It's important to note that our analysis focuses on top activations, which may not assess each feature's behavior on low-activating samples. It has been observed that lower activating samples tend to be less relevant to the interpretation (Bricken et al., 2023) for vanilla SAE features. Both TopK and JumpReLU SAEs are designed to remove weakly firing instances and prioritize features with higher activation levels, based on a threshold or rank-based thresholding. This may not reflected in the monosemanticity scores, which is a limitation of our current evaluation.

4.4 Activation Frequency

The firing frequency of latent features in SAEs is a crucial metric for validating the correctness of the training process and diagnosing potential issues early. If too many features fire frequently, they are likely to be uninterpretable. Conversely, if features rarely fire across a large number of tokens, it could indicate that the sparsity constraints are too strict.

We empirically find that a satisfying proportion of inactive features (those that do not fire once over 100,000 tokens) is around 10%. This includes both ultra-rare features and potential training failures. For larger SAEs, this threshold may be higher, but we use it to guide the training of Llama Scope SAEs. Additionally, we aim to keep ultra-active features (those firing with a frequency greater than 0.1) below 2% of all features.

We show the firing frequency of L7R-8x, L15R-8x and L23R-8x TopK SAEs in Figure 4. The firing frequency distribution of wider SAEs tend to *left-shift* towards lower frequency compared to

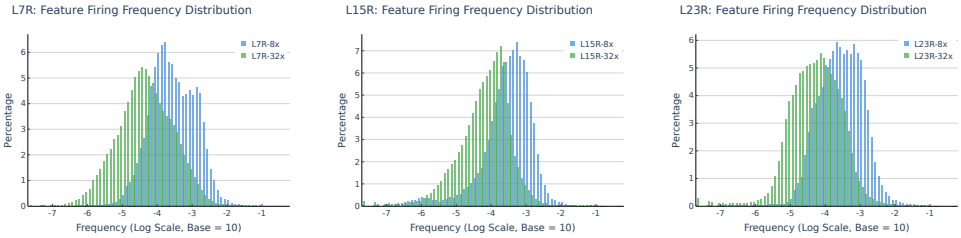

Figure 4: Firing frequency of L7R-8x, L15R-8x and L23R-8x TopK SAEs.

narrower ones. This is expected, as wider SAEs have more features whereas top-k sparsity constraint is fixed at 50 for both of them.

4.5 Out-of-Distribution Generlization

Since SAE training is resource-intensive, it's important that the models generalize beyond the training distribution. We evaluate the out-of-distribution generalization of our SAEs in two ways: on longer contexts and on instruction-finetuned models.

4.5.1 Across Sequence Length

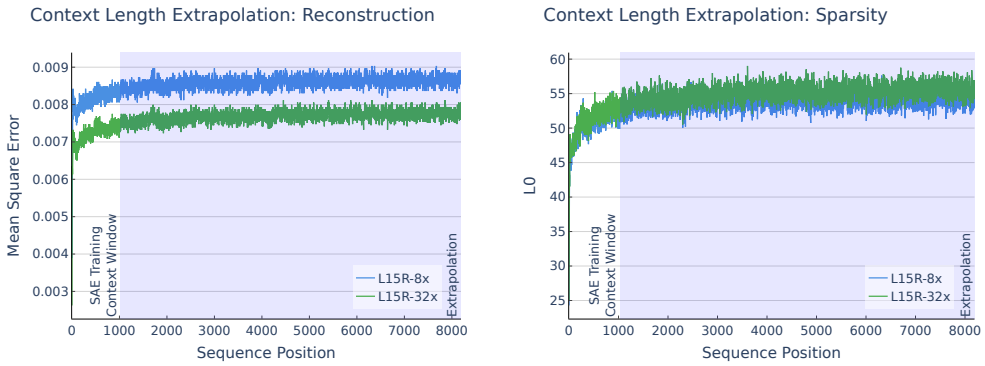

Figure 5: SAE performance on long context data, measured by MSE and L0 sparsity.

We selected all documents with more than 8192 tokens from L-Eval (An et al., 2024) and truncated them to 8192 tokens, resulting in a dataset of 166 documents and 1.3 million tokens. The L0 sparsity and MSE of Llama Scope SAEs on this dataset are shown in Figure 5.

Since our SAEs are trained on 1024-token sequences (Section 3.1), a slight decrease in performance with longer sequences is expected. However, the degradation converges at around 8192 tokens. The average reconstruction loss in the last 1024 tokens is 0.0086, which is a 12% increase compared to the training data. The L0 sparsity increases from 50 to 55.

Although we have not fine-tuned our SAEs on longer sequences, we believe they can be further optimized for long-context data with minimal fine-tuning, which we plan to explore in future work.

4.5.2 To Instruction-Finetuned Models

When discussing the generalization of SAEs to instruction-finetuned models, there are three key dimensions to consider: the language model, SAE training data, and downstream tasks.

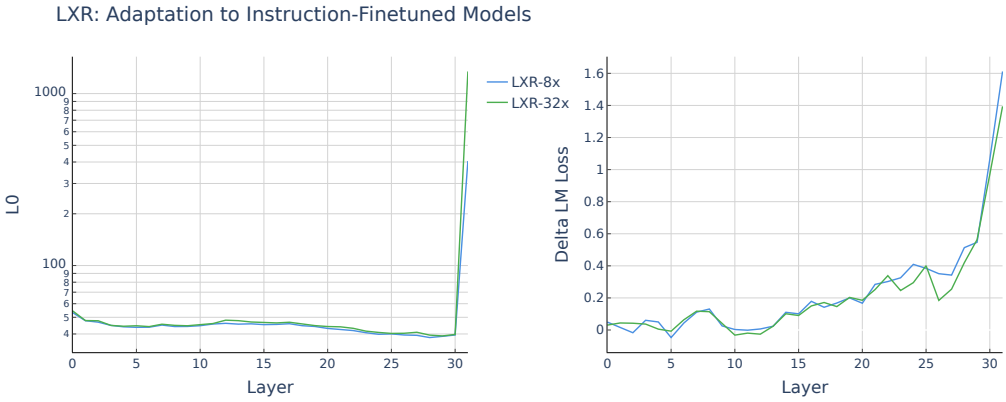

Figure 6: L0 sparsity and Delta LM loss of L15R-8x TopK SAEs on instruction-finetuned models.

In Templeton et al. (2024b), SAEs trained on pretraining data with an instruction model generalized well to downstream instruction-guided tasks[2]. This can be described as *base SAE training data generalizing to chat tasks* while keeping the model fixed to instruction-finetuned models.

Our setup is similar to that of Kissane et al. (2024), where *base SAE training data and a base language model are generalized to chat models and chat data*. Specifically, we use a subset of the Wildchat dataset (Zhao et al., 2024) with 4K chat histories. These are fed into Llama-3.1-8B-Instruct to generate activations, and we evaluate 32 LXR-8x-TopK and 32 LXR-32x-TopK SAEs on this dataset 6.

With the exception of L31R-8x and L31R-32x (trained on residual stream activations just before the final layernorm), no significant degradation in Delta LM loss or increase in L0 sparsity is observed across the other SAEs[3]. This indicates that our SAEs generalize well to instruction-finetuned models.

For example, replacing L15R activations with those reconstructed by L15R-32x-TopK SAEs in the base model increases the language modeling loss by 0.162, whereas the increase is only 0.090 for instruction-finetuned models, all while maintaining the same L0 sparsity. The smaller Delta LM loss in Llama-3.1-8B-Instruct does not necessarily mean that the SAEs are better suited for instruction-finetuned models, but rather that the downstream tasks are more robust to the SAE's perturbations.

4.6 Feature Geometry

To better understand the behavior of SAEs across different architectures and sizes, we analyze their feature geometry, following the approach of Bricken et al. (2023); Templeton et al. (2024b). Specifically, we examine the cosine similarity between decoder columns $W^{dec}_{:,i}$ to identify adjacent features in the activation space (the *decoder* view). For this analysis, we selected a feature in L15R-8x-TopK that activates in contexts related to significant historical events, particularly those harmful to humanity. We then found the six most similar features in both L15R-8x-Vanilla and L15R-8x-TopK, and the 24 most similar features in L15R-32x-TopK.

It is important to note that the cosine similarity between two features is not a direct measure of their semantic similarity. However, we do find that features with high cosine similarity tend to have similar interpretations, as shown in the following section.

The Threats-to-Humanity Cluster. Figure 7 shows the 2D UMAP projection of the decoder columns for these 36 features. Using GPT-4o to describe their top activating samples (the *encoder* view), we found a coherent theme across all features: they are related to events like wars, pandemics, natural disasters, and other harmful occurrences.

[2]Interestingly, they also showed generalization to image data.

[3]The exceptions in the near-output space may offer interesting insights, though we do not explore them further here.

Figure 7: Feature geometry of L15R-8x-Vanilla, L15R-8x-TopK and L15R-32x-TopK SAEs.

UMAP, which preserves the local structure of data, also reveals more detailed local clusters. For instance, the top-left region of Figure 7 contains features associated with large-scale wars, while other regions correspond to themes such as *climate change*, *pandemics*, and *financial crises*.

How Close Enough are Features to Be Neighbors? One challenge in identifying *feature neighbors* is the lack of a principled threshold for determining whether two features are close enough to be considered neighbors. To address this, we provide a baseline based on the Johnson-Lindenstrauss lemma's inner product version. If we randomly project F *true* features to the D-dimensional hidden space, the probability of finding a pair (out of $\binom{F}{2}$ pairs) that have cosine similarity larger than $\epsilon = \sqrt{\frac{12 \ln F}{D}}$ is approximately $2/F$.[4]

For 8x SAEs, $\epsilon = 0.174$, and for 32x SAEs, $\epsilon = 0.186$. We empirically validate this by randomly mapping $F = 8 \times 4096$ (or 32×4096) features into a $D = 4096$-dimensional space, where we find the maximum pairwise cosine similarity to be 0.09 (0.10). In comparison, the farthest cosine

[4]Proof can be found at https://home.ttic.edu/~gregory/courses/LargeScaleLearning/lectures/jl.pdf. The small difference between inner product and cosine similarity is neglected for estimation purposes.

similarity distance between the 36 features in Figure 7 is 0.28, suggesting that these features exhibit non-trivial similarity.

Wider SAEs Do Learn New Features. In this case, features from L15R-32x-TopK are not merely linear combinations of features from L15R-8x-TopK. For example, there is a distinct *Brexit* feature in L15R-32x-TopK that activates exclusively on this topic, whereas the most similar feature in the smaller SAEs is a more general *Historical Movements* feature. This suggests that wider SAEs are capable of learning entirely new features, not just frequent compositions of existing ones.

TopK SAEs Learn Similar Features to Vanilla SAEs. We also observe that the features learned by TopK SAEs closely resemble those in Vanilla SAEs, as measured by cosine similarity of the decoder columns. This is visually supported by the fact that the blue dots (L15R-8x-Vanilla) and green dots (L15R-8x-TopK) are intermixed in Figure 7 and share a common theme.

For a broader analysis, we computed the cosine similarity between all features in L15R-8x-TopK and their most similar counterparts in L15R-8x-Vanilla. Figure 8 shows the cumulative distribution of these similarity scores. The similarity between TopK and Vanilla SAEs is significantly higher than a random baseline (where the max cosine similarity between two random SAEs of the same size is computed). This indicates that TopK and Vanilla SAEs share a universal feature geometry.

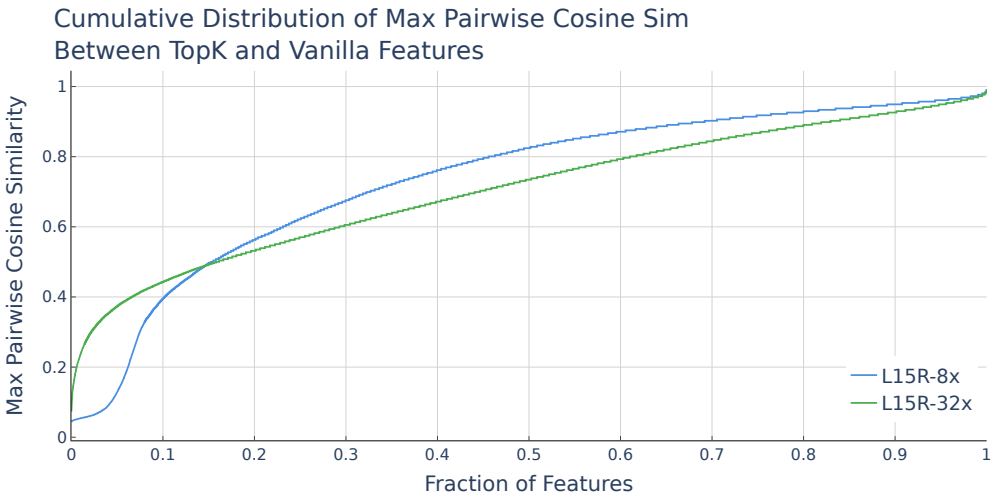

Figure 8: Feature geometry of L15R-8x-Vanilla, L15R-8x-TopK and L15R-32x-TopK SAEs.

5 Related And Future Work

Scaling up Sparse Autoencoders. Building on recent success in scaling SAEs to large language models (Templeton et al., 2024b; Lieberum et al., 2024; Gao et al., 2024), further scaling is likely to continue with larger models. The development of customized GPU acceleration for SAEs (Gao et al., 2024) holds great potential for extracting more complex concepts from larger models (Kaplan et al., 2020; Hoffmann et al., 2022). Additionally, structured sparsity in Mixture-of-Expert (MoE) SAEs, as mentioned in Gao et al. (2024) and Sharkey et al. (2024), has shown promise for efficient SAE training (Mudide et al., 2024). This approach could significantly reduce computational costs while maintaining performance, making it a promising avenue for future exploration.

Extending Neuron-Level Analysis to SAE Features. SAE features share similarities with MLP neurons, which have been widely studied for their role in knowledge representation (Dai et al., 2022), universality (Gurnee et al., 2023; 2024), and multilingual capabilities (Tang et al., 2024). Insights from these studies could be applied to SAE features to deepen our understanding of language models. Recent work (Wang et al., 2024) suggests that universality across model architectures can be reflected more accurately by SAE features than by MLP neurons.

Revealing a More Interpretable Latent Space. Understanding the activation space of language models is a central challenge in interpretability research. We believe techniques used in language model activation spaces can be applied to SAEs' latent spaces. For instance, linear probes and decision tree classifiers have been successfully used to predict harmful content from SAE features (Bricken et al., 2024). Additionally, similarity metrics like Canonical Correlation Analysis (CCA) have been shown to provide valuable insights into SAE latent spaces (Lan et al., 2024).

6 Conclusion

In this work, we introduced Llama Scope SAEs, a series of TopK Sparse Autoencoders trained on the Llama-3.1-8B-Base model. The insights and lessons learned from this work provide a valuable foundation for future improvements in SAE training. As mechanistic interpretability remains an open field with many unexplored ideas, we hope that Llama Scope, alongside other open-source SAEs, will help researchers save time and effort in their investigations.

Both language models and SAEs are evolving rapidly, and we see this as a long-term research direction. We are excited to contribute to the growing body of open-source SAEs and to continue pushing the epistemic frontier of SAEs in the future.

REFERENCES

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJ4-rAVtl.

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 14388–14411. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.776. URL https://doi.org/10.18653/v1/2024.acl-long.776.

Evan Anders, Clement Neo, Jason Hoelscher-Obermaier, and Jessica N. Howard. Sparse autoencoders find composed features in small toy models. https://www.lesswrong.com/posts/a5wwqza2cY3W7L9cj/sparse-autoencoders-find-composed-features-in-small-toy, 2024.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Trans. Assoc. Comput. Linguistics*, 6:483–495, 2018. doi: 10.1162/TACL_A_00034. URL https://doi.org/10.1162/tacl_a_00034.

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.

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.

Trenton Bricken, Jonathan Marcus, Siddharth Mishra-Sharma, Meg Tong, Ethan Perez, Mrinank Sharma, Kelley Rivoire, Thomas Henighan, and Adam Jermyn. Using dictionary learning features as classifiers. *Transformer Circuits Thread*, 2024. https://transformer-circuits.pub/2024/features-as-classifiers/index.html.

Róbert Csordás, Christopher Potts, Christopher D Manning, and Atticus Geiger. Recurrent neural networks learn to store and generate sequences using non-linear representations. *arXiv preprint arXiv:2408.10920*, 2024.

Hoagy Cunningham and Tom Conerly. Circuits updates - june 2024. *Transformer Circuits Thread*, 2024. https://transformer-circuits.pub/2024/june-update/index.html.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 8493–8502. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.581. URL https://doi.org/10.18653/v1/2022.acl-long.581.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable LLM feature circuits. *CoRR*, abs/2406.11944, 2024. doi: 10.48550/ARXIV.2406.11944. URL https://doi.org/10.48550/arXiv.2406.11944.

Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer ElShowk, Nicholas Joseph, Nova DasSarma, Ben Mann, Danny Hernandez, Amanda Askell, Kamal Ndousse, Andy Jones, Dawn Drain, Anna Chen, Yuntao Bai, Deep Ganguli, Liane Lovitt, Zac Hatfield-Dodds, Jackson Kernion, Tom Conerly, Shauna Kravec, Stanislav Fort, Saurav Kadavath, Josh Jacobson, Eli Tran-Johnson, Jared Kaplan, Jack Clark, Tom Brown, Sam McCandlish, Dario Amodei, and Christopher Olah. Softmax linear units. *Transformer Circuits Thread*, 2022a. https://transformer-circuits.pub/2022/solu/index.html.

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*, 2022b. https://transformer-circuits.pub/2022/toy_model/index.html.

Nelson Elhage, Robert Lasenby, and Christopher Olah. Privileged bases in the transformer residual stream. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/privileged-basis/index.html.

Joshua Engels, Isaac Liao, Eric J. Michaud, Wes Gurnee, and Max Tegmark. Not all language model features are linear. *CoRR*, abs/2405.14860, 2024. doi: 10.48550/ARXIV.2405.14860. URL https://doi.org/10.48550/arXiv.2405.14860.

N. Benjamin Erichson, Zhewei Yao, and Michael W. Mahoney. Jumprelu: A retrofit defense strategy for adversarial attacks. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred (eds.), *Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2020, Valletta, Malta, February 22-24, 2020*, pp. 103–114. SCITEPRESS, 2020. doi: 10.5220/0009316401030114. URL https://doi.org/10.5220/0009316401030114.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *CoRR*, abs/2406.04093, 2024. doi: 10.48550/ARXIV.2406.04093. URL https://doi.org/10.48550/arXiv.2406.04093.

Xuyang Ge, Fukang Zhu, Wentao Shu, Junxuan Wang, Zhengfu He, and Xipeng Qiu. Automatically identifying local and global circuits with linear computation graphs. *CoRR*, abs/2405.13868, 2024. doi: 10.48550/ARXIV.2405.13868. URL https://doi.org/10.48550/arXiv.2405.13868.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=JYs1R9IMJr.

Wes Gurnee, Theo Horsley, Zifan Carl Guo, Tara Rezaei Kheirkhah, Qinyi Sun, Will Hathaway, Neel Nanda, and Dimitris Bertsimas. Universal neurons in GPT2 language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=ZeI104QZ8I.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Zhengfu He, Xuyang Ge, Qiong Tang, Tianxiang Sun, Qinyuan Cheng, and Xipeng Qiu. Dictionary learning improves patch-free circuit discovery in mechanistic interpretability: A case study on othello-gpt. *CoRR*, abs/2402.12201, 2024. doi: 10.48550/ARXIV.2402.12201. URL https://doi.org/10.48550/arXiv.2402.12201.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. doi: 10.48550/ARXIV.2203.15556. URL https://doi.org/10.48550/arXiv.2203.15556.

Jing Huang, Zhengxuan Wu, Christopher Potts, Mor Geva, and Atticus Geiger. RAVEL: evaluating interpretability methods on disentangling language model representations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 8669–8687. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.470. URL https://doi.org/10.18653/v1/2024.acl-long.470.

Robert Huben, Hoagy Cunningham, Logan Riggs, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Connor Kissane, Robert Krzyzanowski, Arthur Conmy, and Neel Nanda. Saes (usually) transfer between base and chat models. Alignment Forum, 2024. URL https://www.alignmentforum.org/posts/fmwk6qxrpW8d4jvbd/saes-usually-transfer-between-base-and-chat-models.

Michael Lan, Philip Torr, Austin Meek, Ashkan Khakzar, David Krueger, and Fazl Barez. Sparse autoencoders reveal universal feature spaces across large language models. *arXiv preprint arXiv:2410.06981*, 2024.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

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. *CoRR*, abs/2403.19647, 2024. doi: 10.48550/ARXIV.2403.19647. URL https://doi.org/10.48550/arXiv.2403.19647.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295, 2024. doi: 10.48550/ARXIV.2403.08295. URL https://doi.org/10.48550/arXiv.2403.08295.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun (eds.), *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

Anish Mudide, Joshua Engels, Eric J Michaud, Max Tegmark, and Christian Schroeder de Witt. Efficient dictionary learning with switch sparse autoencoders. *arXiv preprint arXiv:2410.08201*, 2024.

Chris Olah and Adam Jermyn. Circuits updates - july 2024. *Transformer Circuits Thread*, 2024. https://transformer-circuits.pub/2024/july-update/index.html.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Bruno A. Olshausen and David J. Field. Learning efficient linear codes for natural images: the roles of sparseness, overcompleteness, and statistical independence. In Bernice E. Rogowitz and Jan P. Allebach (eds.), *Human Vision and Electronic Imaging, San Jose, CA, USA, January 28, 1996*, volume 2657 of *SPIE Proceedings*, pp. 132–138. SPIE, 1996. doi: 10.1117/12.238708. URL https://doi.org/10.1117/12.238708.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *CoRR*, abs/2407.14435, 2024. doi: 10.48550/ARXIV.2407.14435. URL https://doi.org/10.48550/arXiv.2407.14435.

Lee Sharkey, Lucius Bushnaq, Dan Braun, Stefan Heimersheim, and Nicholas Goldowsky-Dill. A list of 45+ mech interp project ideas from apollo research's interpretability team, 2024. https://www.alignmentforum.org/posts/KfkpgXdgRheSRWDy8/a-list-of-45-mech-interp-project-ideas-from-apollo-research.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Tianyi Tang, Wenyang Luo, Haoyang Huang, Dongdong Zhang, Xiaolei Wang, Xin Zhao, Furu Wei, and Ji-Rong Wen. Language-specific neurons: The key to multilingual capabilities in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 5701–5715. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.309. URL https://doi.org/10.18653/v1/2024.acl-long.309.

Adly Templeton, Tom Conerly, Jack Lindsey, Hoagy Cunningham, and Andrew Persic. Circuits updates - august 2024 - interpretability evals case study. *Transformer Circuits Thread*, 2024a. https://transformer-circuits.pub/2024/august-update/index.html.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024b. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Junxuan Wang, Xuyang Ge, Wentao Shu, Qiong Tang, Yunhua Zhou, Zhengfu He, and Xipeng Qiu. Towards universality: Studying mechanistic similarity across language model architectures. *arXiv preprint arXiv:2410.06672*, 2024.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

A INFRASTRUCTURE

A.1 ACTIVATION BUFFER

One of the major challenges in training SAEs is the substantial storage and throughput required for latent activations. While text data requires only 2 bytes per token, latent activations occupy 8K bytes per token—resulting in a 4,096x increase in both storage needs and disk throughput. This, combined with the relatively fast training steps of shallow SAEs, means that data loading quickly becomes the main bottleneck in the training process.

Due to these infrastructure constraints, we do not save activations in advance but instead generate them on-the-fly. This contrasts with the approach taken by Lieberum et al. (2024); Templeton et al. (2024b), where activations are pre-saved and a high-speed dataloading pipeline is built to keep up with training.

To manage this, we adopt a producer-consumer model. Language Models (LMs) generate activations and store them in an activation buffer, while the SAEs consume the activations in random order. The process is serialized: once the buffer is full, SAE training begins, and when half the buffer is consumed, the LMs refill it. Each time the buffer is refilled, we shuffle it to introduce randomness into the training data without needing to save and shuffle all activations at once.

A.2 MIXED PARALLELISM

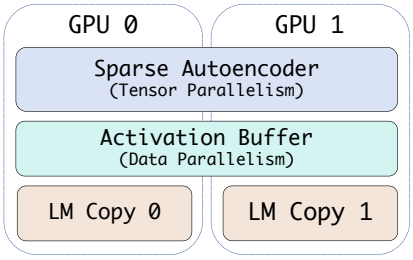

Figure 9: Mixed parallelism strategy for training SAEs.

To efficiently train SAEs with a large number of features, we implement a mixed parallelism strategy, which balances the memory demands of SAE training with the slower process of activation generation. This approach accelerates buffer refilling and reduces the memory bottleneck during training.
As shown in Figure 9, a copy of the Language Model (LM) is loaded on each GPU, while the SAE is distributed across GPUs using tensor parallelism. Each GPU maintains its own independent activation buffer. During training, a minibatch is sampled from the activation buffer on each GPU, and these minibatches are all-gathered to form the input for the SAE.

This setup combines data parallelism for activation generation with tensor parallelism for SAE training. We empirically find that SAEs and their gradients consume significant memory, but individual training steps are relatively fast. In contrast, activation generation is slower but requires less memory. By combining these parallelism techniques, mixed parallelism accelerates activation generation while efficiently managing memory during SAE training.

A.3 COMPARISON TO PRE-SAVING APPROACH

Previous work on training large SAEs using distributed disk reading and extensive storage resources (Lieberum et al., 2024; Templeton et al., 2024b) has typically been conducted in industrial labs with powerful infrastructure. In contrast, our online activation generation approach eliminates the need for vast storage resources, making it more suitable for academic research. With this method, SAEs can be trained on an 8B-size model using just a single NVIDIA A100 GPU, without the need to pre-save activations. In comparison, the pre-saving approach requires approximately 2TB of storage and a disk throughput of at least 500MB/s to ensure that data loading keeps pace with training.

However, a key limitation of our online generation method is the potential for redundant computation when training multiple SAEs on the same position. If we need to train SAEs with different widths at the same position, the same activations must be generated multiple times, leading to 1-2 times the redundant activation generation compared to the pre-saving approach.

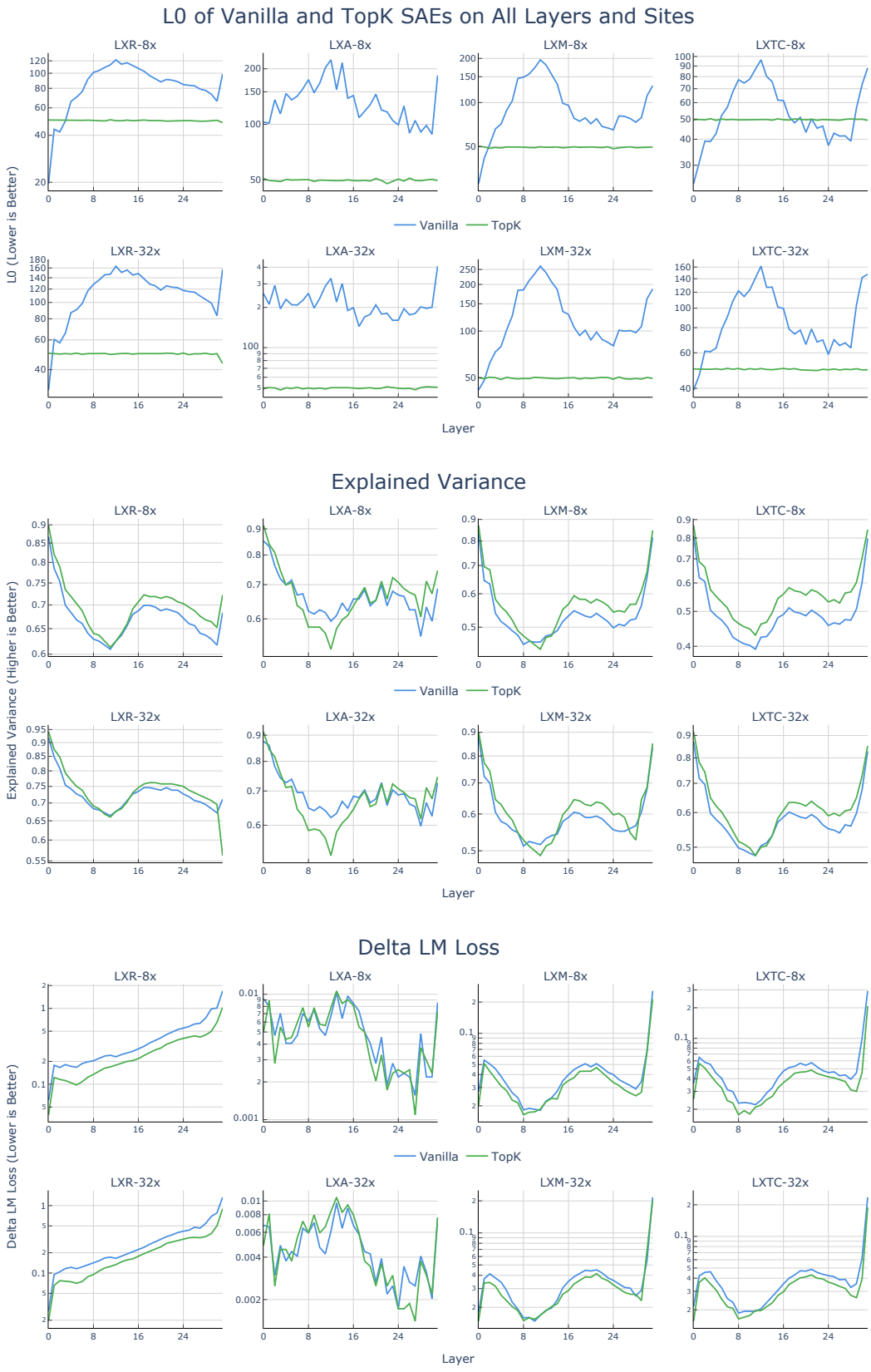

Figure 10: All 256 SAEs are evaluated on L0 sparsity (upper), explained variance (middle) and Delta LM loss (lower).

B Evaluation Result for All SAEs

C Ablation Studies

Post-Processing Does Not Hurt Performance. We find that the post-processing step of Section 3.3.4 does not hurt sparsity-fidelity trade-off. All post training transformations are computationally equivalent except for JumpReLU inference mode, where we do not use TopK sparsity constraint but instead find a threshold where K features fire at each position *in expectation*. We evaluate the L0 sparsity and MSE efficiency of all 32 LXR-8x-TopK SAEs with and without JumpReLU inference mode. The results are nearly identical, as shown in Figure 11.

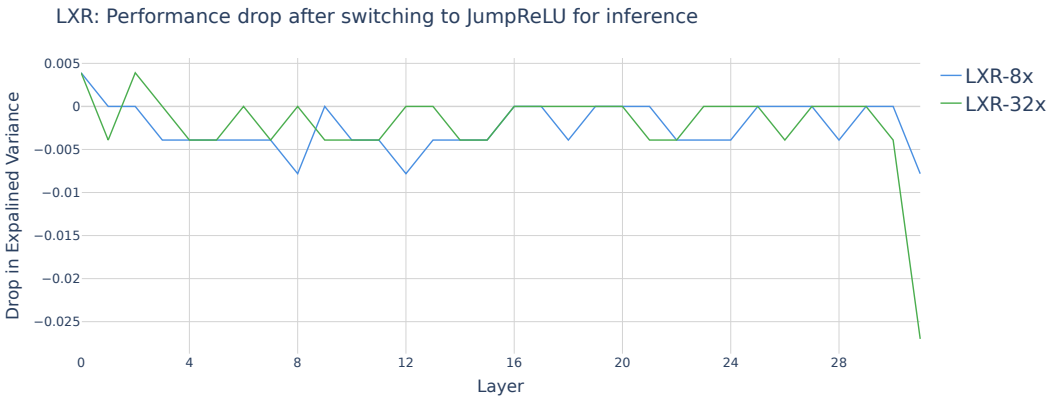

Figure 11: Drop in performance is negligible after switching to JumpReLU inference mode.

K-Annealing Accelerates Convergence. Figure 12 shows the training curve of L15R-8x-TopK SAEs with and without annealing K at start of training. By annealing K from 4096 to 50 in the first 10% of the training steps, we find that the SAEs latents are active during the whole training process (almost all features fire at least once over 1e6 tokens). In comparison, the SAE features without K-annealing do not begin to activate until about 1/3 of the training steps. Besides, both training curves converge to the same MSE loss but the K-annealing curve converges faster.

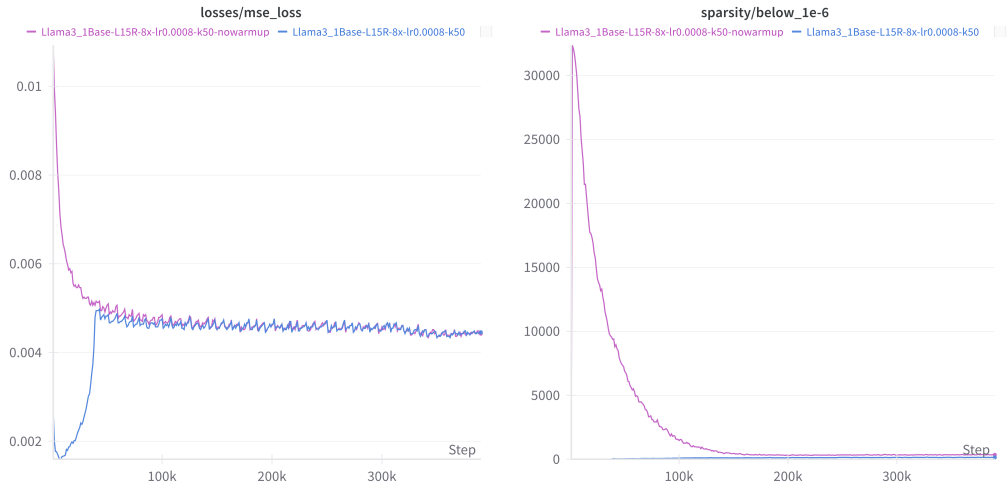

Figure 12: Convergence of L15R-8x-TopK SAEs with and without K-Annealing.

Including Decoder L2 Norm Improves Reconstruction? We also ablated the inclusion of decoder L2 norm in the topk sparsity constraint (Equation 5) with all other hyperparameters fixed. However, the original version reported by Gao et al. (2024) where the decoder L2 norm is fixed to 1 after each training step is significantly worse in terms of MSE loss (at least a 0.05 drop in variance explained). We suspect there are bugs in replicating this baseline and leave it as future work.

D Prompt Design for Feature Interpretability

```
We are analyzing the activation levels of features in a neural network,
where each feature activates certain tokens in a text. Each token's
activation value indicates its relevance to the feature, with higher
values showing stronger association. Your task is to give this feature
a  monosemanticity score based on the following scoring rubric:

Activation Consistency

5: Clear pattern with no deviating examples

4: Clear pattern with one or two deviating examples

3: Clear overall pattern but quite a few examples not fitting that
pattern

2: Broad consistent theme but lacking structure

1: No discernible pattern

Consider the following activations for a feature in the neural
network. Activation values are non-negative, with higher values
indicating a stronger connection between the token and the
feature. You only need to return a number. It
represents your score for feature monosemanticity.

[Context]
Sentence 1:
<START>
 students        0.0
 to        0.0
 build        0.0
 and        0.0
 repair        0.0
 all        0.0
 varieties        0.0
 of        0.0
 aviation        0.0
 instrumentation        0.0
 .        0.0
 The        0.0
 se        0.0
 ap        0.0
 lane        0.0
 base        0.0
 continued        0.0
 operations        0.0
 during        0.0
 the        0.0
 war        3.1
 as        0.0
 an        0.0
 all        0.0
 -f        0.0
<END>
```

. . .