RESIDUAL STREAM ANALYSIS WITH MULTI-LAYER SAEs

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse autoencoders (SAEs) are a promising approach to interpreting the internal representations of transformer language models. However, SAEs are usually trained separately on each transformer layer, making it difficult to use them to study how information flows across layers. To solve this problem, we introduce the multi-layer SAE (MLSAE): a single SAE trained on the residual stream activation vectors from every transformer layer. Given that the residual stream is understood to preserve information across layers, we expected MLSAE latents to 'switch on' at a token position and remain active at later layers. Interestingly, we find that individual latents are often active at a single layer for a given token or prompt, but the layer at which an individual latent is active may differ for different tokens or prompts. We quantify these phenomena by defining a distribution over layers and considering its variance. We find that the variance of the distributions of latent activations over layers is about two orders of magnitude greater when aggregating over tokens compared with a single token. For larger underlying models, the degree to which latents are active at multiple layers increases, which is consistent with the fact that the residual stream activation vectors at adjacent layers become more similar. Finally, we relax the assumption that the residual stream basis is the same at every layer by applying pre-trained tuned-lens transformations, but our findings remain qualitatively similar. Our results represent a new approach to understanding how representations change as they flow through transformers. We release our code to train and analyze MLSAEs at `<redacted>`.

1 INTRODUCTION

Sparse autoencoders (SAEs) learn interpretable directions or 'features' in the representation spaces of language models (Elhage et al., 2022; Cunningham et al., 2023; Bricken et al., 2023). Typically, SAEs are trained on the activation vectors from a single model layer (Gao et al., 2024; Templeton et al., 2024; Lieberum et al., 2024). This approach illuminates the representations within a layer. However, Olah (2024); Templeton et al. (2024) believe that models may encode meaningful concepts by simultaneous activations in multiple layers, which SAEs trained at a single layer do not address. Furthermore, it is not straightforward to automatically identify correspondences between features from SAEs trained at different layers, which may complicate circuit analysis (e.g. He et al., 2024).

To solve this problem, we take inspiration from the residual stream perspective, which states that transformers (Vaswani et al., 2017) selectively write information to and read information from token positions with self-attention and MLP layers (Elhage et al., 2021; Ferrando et al., 2024). The results of subsequent circuit analyses, like the explanation of the indirect object identification task presented by Wang et al. (2022), support this viewpoint and cause us to expect the activation vectors at adjacent layers in the residual stream to be relatively similar (Lad et al., 2024).

To capture the structure shared between layers in the residual stream, we introduce the multi-layer SAE (MLSAE): a single SAE trained on the residual stream activation vectors from every layer of a transformer language model. Importantly, the autoencoder itself has a single hidden layer – it is multi-layer only in the sense that it is trained on activations from multiple layers of the underlying transformer. In particular, we consider the activation vectors from each layer as separate training examples, which is equivalent to training a single SAE at each layer individually but with the parameters tied across layers. We briefly discuss alternative methods in Section 5.

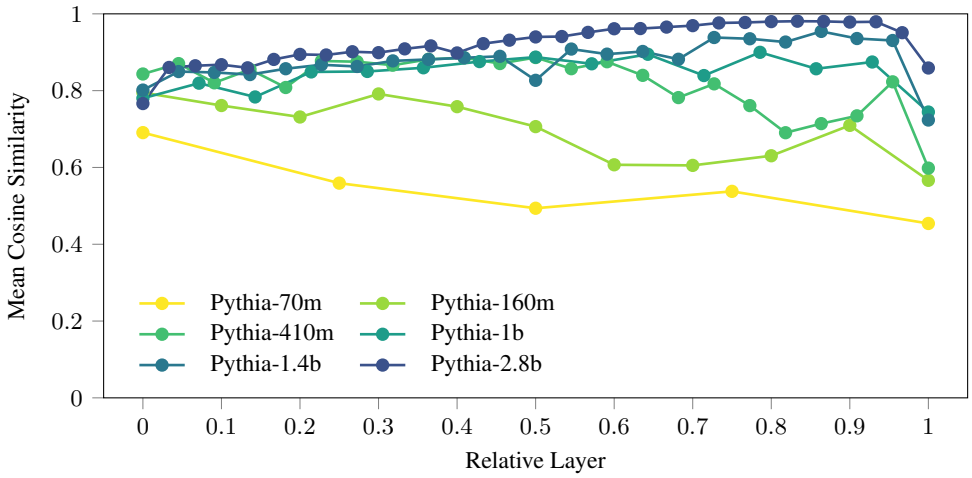


Figure 1: The mean cosine similarities between the residual stream activation vectors at adjacent layers of transformers, over 10 million tokens from the test set. To compare transformers with different numbers of layers, we divide the lower of each pair of adjacent layers by the number of pairs. This 'relative layer' is the x-axis of the plot. We subtract the dataset mean from the activation vectors at each layer before computing cosine similarities to control for changes in the norm between layers (Heimersheim & Turner, 2023), which we demonstrate in Figure 4.

We show that multi-layer SAEs achieve comparable reconstruction error and downstream loss to single-layer SAEs while allowing us to directly identify and analyze features that are active at multiple layers (Section 4.1). When aggregating over a large sample of tokens, we find that individual latents are likely to be active at multiple layers, and this measure increases with the number of latents. However, for a single token, latent activations are more likely to be isolated to a single layer. For larger underlying transformers, we show that the residual stream activation vectors at adjacent layers are more similar and that the degree to which latents are active at multiple layers increases.

Finally, we relax the assumption that the residual stream basis is the same at every layer by applying pre-trained tuned-lens transformations to activation vectors before passing them to the encoder. Surprisingly, this does not obviously increase the extent of multi-layer latent activations.

2 Related work

A sparse code represents many signals, such as sensory inputs, by simultaneously activating a relatively small number of elements, such as neurons (Olshausen & Field, 1996; Bell & Sejnowski, 1997). Sparse dictionary learning (SDL) approximates each input vector by a linear combination of a relatively small number of learned basis vectors. The learned basis is usually overcomplete: it has a greater dimension than the inputs. Independent Component Analysis (ICA) achieves this aim by maximizing the statistical independence of the learned basis vectors by iterative optimization or training (Bell & Sejnowski, 1995; 1997; Hyvärinen & Oja, 2000; Le et al., 2011). Sparse autoencoders (SAEs) can be understood as ICA with the addition of a noise model optimized by gradient descent (Lee et al., 2006; Ng, 2011; Makhzani & Frey, 2014)

The activations of language models have been hypothesized to be a dense, compressed version of a sparse, expanded representation space (Elhage et al., 2021; 2022). Under this view, there are interpretable directions in the dense representation spaces corresponding to distinct semantic concepts, whereas their basis vectors (neurons) are 'polysemantic' (Park et al., 2023). It has been shown theoretically (Wright & Ma, 2022) and empirically (Elhage et al., 2022; Sharkey et al., 2022; Whittington et al., 2023) that SDL recovers ground-truth features in toy models, and that learned dictionary elements are more interpretable than the basis vectors of language models (Cunningham et al., 2023; Bricken et al., 2023) or dense embeddings (O'Neill et al., 2024). Notably, features are not necessarily linear (Wattenberg & Viégas, 2024; Engels et al., 2024; Hernandez et al., 2024).

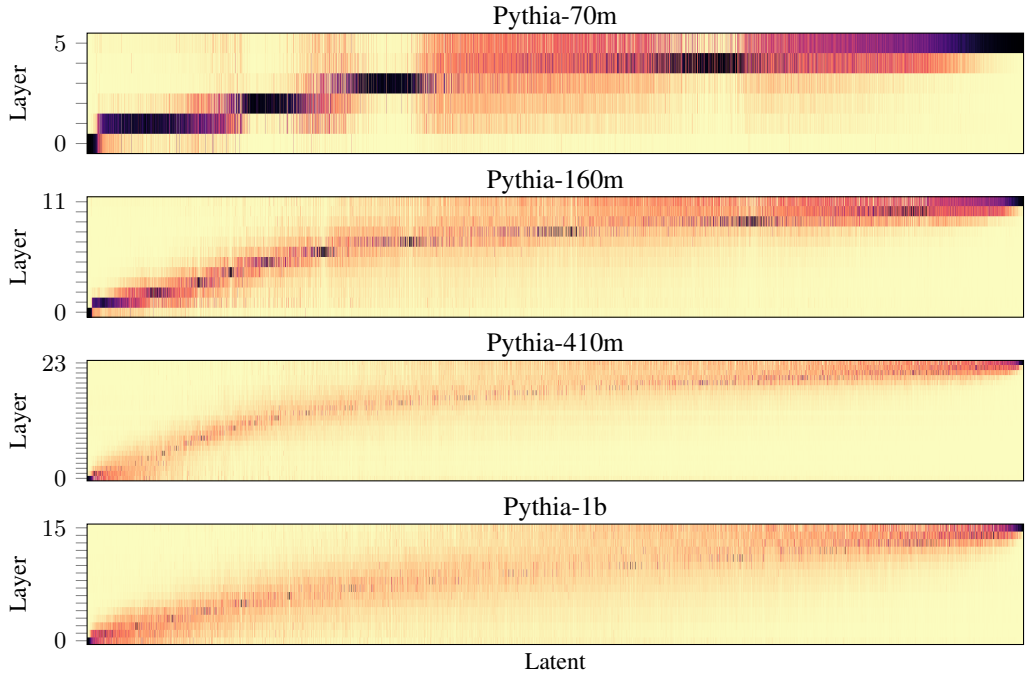


Figure 2: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$. The latents are sorted in ascending order of the expected value of the layer index (Equation 10).

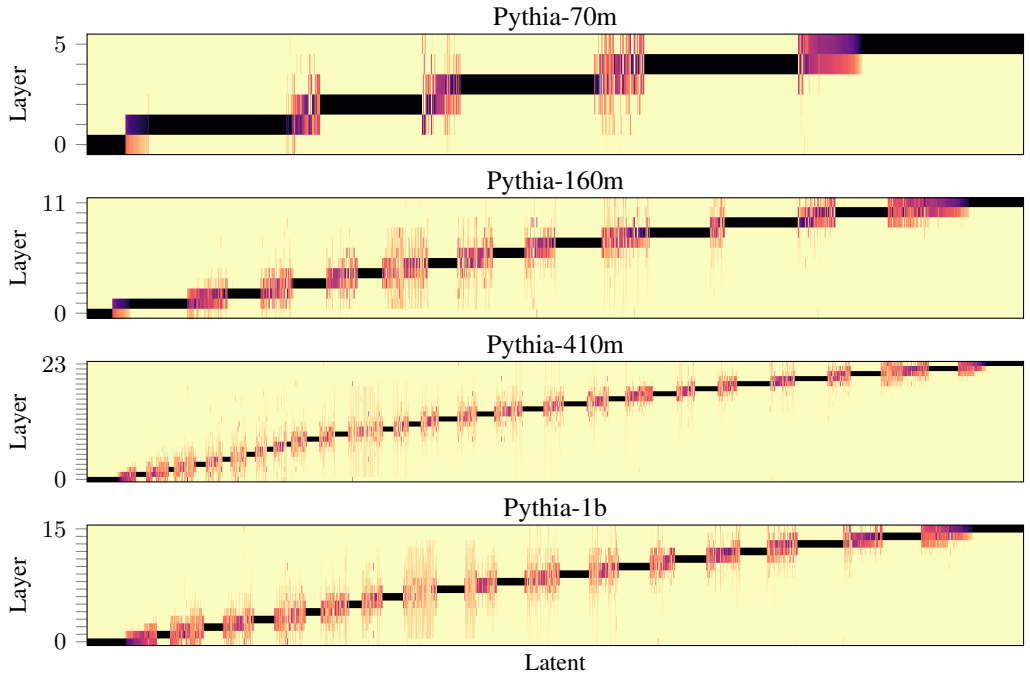


Figure 3: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We exclude latents with maximum activation below 1×10^{-3} and sort latents in ascending order of the expected value of the layer index (Equation 10).

The standard SAE architecture is a single hidden layer with a ReLU activation function and an L^1 sparsity penalty in the training loss (Bricken et al., 2023), but various activation functions (Makhzani & Frey, 2014; Konda et al., 2015; Rajamanoharan et al., 2024b;a) and objectives (Braun et al., 2024) have been proposed. The prevailing approach is to train an SAE on the activation vectors from a single transformer layer, except for Kissane et al. (2024), who concatenate the outputs of multiple attention heads in a single layer, and Yun et al. (2021), who learn an undercomplete basis for the residual stream at multiple layers, albeit by iterative optimization instead of with an autoencoder.

Mechanistic interpretability research often attempts to identify circuits: computational subgraphs of neural networks that implement specific behaviors (Olah et al., 2020; Wang et al., 2022; Conmy et al., 2023; Dunefsky et al., 2024; García-Carrasco et al., 2024; Marks et al., 2024). Representing networks in terms of SAE latents may help to improve circuit discovery (He et al., 2024; O'Neill & Bui, 2024), and these latents can be used to construct steering vectors (Subramani et al., 2022; Templeton et al., 2024; Makelov, 2024), but it is unclear whether SAEs outperform baselines for causal analysis (Chaudhary & Geiger, 2024; Huang et al., 2024). Importantly, SAEs can be scaled up to the activations of large language models, where we expect the number of distinct semantic concepts to be extremely large (Templeton et al., 2024; Gao et al., 2024; Lieberum et al., 2024).

The 'logit lens' is a method to interpret directions in the residual stream by projecting them onto the vocabulary space to elicit token predictions, i.e., multiplying them by the unembedding matrix (nostalgebraist, 2020). However, the residual stream basis is not fixed, so Belrose et al. (2023) introduce the 'tuned lens' approach, where a linear transformation is learned for each layer in the residual stream. The objective is to minimize the KL divergence between the probability distribution over tokens generated by the transformed activations and the 'true' distribution of the model. This approach draws on the perspective of iterative inference (Jastrzębski et al., 2018).

The key difference between previous work (Bricken et al., 2023; Cunningham et al., 2023; Templeton et al., 2024; Gao et al., 2024) and our work is that we introduce the multi-layer SAE, i.e., we train a single SAE at all layers of the residual stream.

3 METHODS

The key idea with a multi-layer SAE is to train a single SAE on the residual stream activation vectors from every layer. In particular, we consider the activations at each layer to be different training examples. Hence, for residual stream activation vectors of model dimension d, the inputs to the multi-layer SAE also have dimension d. For n_T tokens and n_L layers, we train the multi-layer SAE on $n_T n_L$ activation vectors. We use the terms 'SAE feature' and 'latent' interchangeably.

3.1 SETUP

We train MLSAEs on GPT-style language models from the Pythia suite (Biderman et al., 2023). We are primarily interested in the computation performed by self-attention and MLP layers on intermediate representations (Valeriani et al., 2023). Hence, we take the residual stream activation vectors after a given transformer block has been applied, excluding the input embeddings before the first block and taking the last-layer activations before the final layer norm.

We use a k-sparse autoencoder (Makhzani & Frey, 2014; Gao et al., 2024), which directly controls the sparsity of the latent space by introducing a TopK activation function that keeps only the k largest latents. The k largest latents are almost always positive for $k \ll d$, but we follow Gao et al. (2024) in applying a ReLU activation function to guarantee non-negativity. This setup effectively fixes the sparsity (L^0 norm) of the latents at k per activation vector (layer and token) throughout training. For input vectors $\mathbf{x} \in \mathbb{R}^d$ and latent vectors $\mathbf{h} \in \mathbb{R}^n$, the encoder and decoder are defined by:

$$\mathbf{h} = \mathrm{ReLU}(\mathrm{TopK}(\mathbf{W}_{\text{enc}}\mathbf{x} - \mathbf{b}_{\text{pre}})) \tag{1}$$

$$\hat{\mathbf{x}} = \mathbf{W}_{\text{dec}}\mathbf{h} + \mathbf{b}_{\text{pre}} \tag{2}$$

where $\mathbf{W}_{\text{enc}} \in \mathbb{R}^{d \times n}$, $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{n \times d}$, and $\mathbf{b}_{\text{pre}} \in \mathbb{R}^d$. We constrain the pre-encoder bias $\mathbf{b}_{\text{pre}}$ to be the negative of the post-decoder bias, following Bricken et al. (2023); Gao et al. (2024), and standardize activation vectors to zero mean and unit variance before passing them to the encoder.

3.2 TRAINING

We use the fraction of variance unexplained (FVU) as the reconstruction error:

$$\mathrm{FVU}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}{\mathrm{Var}(\mathbf{x})} \tag{3}$$

Here, Var is the variance. We chose the FVU because the input vectors from different layers may have different magnitudes; choosing the mean squared error (MSE) would encourage the autoencoder to prioritize minimizing the reconstruction errors of the layers with the greatest magnitudes.

A potential issue when training SAEs is the occurrence of 'dead' latents, i.e., latent dimensions that are almost always zero. With a k-sparse autoencoder, this means latent dimensions that almost never appear among the k largest latent activations. We follow Bricken et al. (2023); Cunningham et al. (2023) by considering a latent 'dead' if it is not activated within the last 10 million tokens during training. In the multi-layer setting, a latent may be activated by the input vectors from any layer.

Gao et al. (2024, Appendix A.2) propose an auxiliary loss term to minimize the occurrence of dead latents. This AuxK term models the MSE reconstruction error using the k_{aux} largest dead latents:

$$\mathrm{AuxK}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{e} - \hat{\mathbf{e}}\|_2^2 \tag{4}$$

Here, $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ is the reconstruction error of the main model, and $\hat{\mathbf{e}}$ is its reconstruction using the top-k_{aux} dead latents. Let Dead be an 'activation function' that keeps only the dead latents. Then:

$$\mathbf{h}_{\text{dead}} = \mathrm{ReLU}(\mathrm{TopK}_{\text{aux}}(\mathrm{Dead}(\mathbf{W}_{\text{enc}}\mathbf{x} - \mathbf{b}_{\text{pre}}))) \tag{5}$$

$$\hat{\mathbf{e}} = \mathbf{W}_{\text{dec}}\mathbf{h}_{\text{dead}} + \mathbf{b}_{\text{pre}} \tag{6}$$

The full loss is the FVU plus the auxiliary loss term, multiplied by a small coefficient α:

$$\mathcal{L} = \mathrm{FVU}(\mathbf{x}, \hat{\mathbf{x}}) + \alpha \cdot \mathrm{AuxK}(\mathbf{x}, \hat{\mathbf{x}}) \tag{7}$$

Following Gao et al. (2024), we choose k_{aux} as a power of 2 close to $d/2$ and $\alpha = 1/32$.

Our hyperparameters are the expansion factor $R = n/d$, the ratio of the number of latents to the model dimension, and the sparsity k, the number of largest latents to keep in the TopK activation function. We choose expansion factors as powers of 2 between 1 and 256, yielding autoencoders with between 512 and 131072 latents for Pythia-70m, and k as powers of 2 between 16 and 512 (Appendix B).

The computational expense of training a single multi-layer SAE on n_L layers of the residual stream is approximately the same as training n_L single-layer SAEs on the same number of tokens. We ran most experiments on a single NVIDIA GeForce RTX 3090 GPU for between 12 and 24 hours; we ran the largest experiments (e.g., with Pythia-1b or an expansion factor of $R = 256$) on a single NVIDIA A100 80GB GPU for up to three days.

The implementation is based on Gao et al. (2023); Belrose (2024); see Appendix A for details.

3.3 TUNED LENS

In the tuned lens method, an affine transformation is learned from the output space of layer ℓ to the output space of the final layer, called the translator for layer ℓ (Belrose et al., 2023). With our setup, we want to transform the residual stream activation vectors at each layer into more similar bases before passing them to the encoder and invert that transformation after the decoder.

Importantly, the authors note that their implementation[1] uses a residual connection:

$$\mathbf{x}' = \mathbf{x} + (\mathbf{W}_{\text{lens}}\mathbf{x} + \mathbf{b}_{\text{lens}}) \tag{8}$$

Here, $\mathbf{x}$ is the input vector to the encoder, and $\mathbf{x}'$ is the transformed input vector. This parameterization ensures that L_2 regularization (weight decay) pushes the transformation towards the identity matrix instead of zero. Hence, to invert the transformation, we need:

$$\hat{\mathbf{x}} = (\mathbf{I} + \mathbf{W}_{\text{lens}})^{-1}(\hat{\mathbf{x}}' - \mathbf{b}_{\text{lens}}) \tag{9}$$

[1] `https://github.com/AlignmentResearch/tuned-lens`, file: `tuned_lens/nn/lenses.py`

Model	Pythia-70m	Pythia-160m	Pythia-410m	Pythia-1b	GPT-2 small
FVU	0.097	0.106	0.081	0.095	0.093
MSE	0.103	0.105	0.113	0.455	5.782
L^1 Norm	66	76	85	110	197
Delta CE Loss	0.565	0.432	0.414	0.404	0.759
KL Divergence	$1.621 \cdot 10^3$	$1.217 \cdot 10^3$	$1.105 \cdot 10^3$	$1.057 \cdot 10^3$	$1.023 \cdot 10^3$

(a) Without tuned lens

Model	Pythia-70m	Pythia-160m	Pythia-410m
FVU	0.030	0.088	0.073
MSE	0.838	0.404	0.133
L^1 Norm	61	90	80
Delta CE Loss	0.274	-0.080	0.827
KL Divergence	$2.718 \cdot 10^3$	$1.962 \cdot 10^3$	$1.448 \cdot 10^3$

(b) With tuned lens

Table 1: The mean reconstruction error and downstream loss metrics for MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$, over 1 million tokens from the test set. We provide further details in Appendix B.

In Eq. 9, $\hat{\mathbf{x}}'$ is the transformed output vector of the decoder, $\hat{\mathbf{x}}$ is the output vector, $\mathbf{W}_{\text{lens}} \in \mathbb{R}^{d \times d}$, and $\mathbf{b}_{\text{lens}} \in \mathbb{R}^d$. With our setup, $\mathbf{x}'$ and $\hat{\mathbf{x}}'$ replace the input and output vectors that we pass to the encoder and use to compute the loss. Notably, we use the transformed vectors to compute reconstruction errors (Figure 12). We compute the inverse $(\mathbf{I} + \mathbf{W}_{\text{lens}})^{-1}$ for each layer once at the start of training.

We use pre-trained tuned lenses provided by the authors of Belrose et al. (2023). Notably, these did not include Pythia-1b at the time of writing.[2]

4 Results

4.1 Evaluation

The key advantage of a multi-layer SAE is to be able to study how information flows across layers in the residual stream. However, this approach is only useful if the MLSAE performs comparably to single-layer SAEs. The FVU reconstruction error in the loss (Section 3.2) is a proxy for the degree to which an SAE explains the behavior of the underlying model. Hence, we also measure the increase in the cross-entropy loss when the residual stream activations at a given layer are replaced by their reconstruction, following Braun et al. (2024); Gao et al. (2024); Lieberum et al. (2024).

Table 1 summarizes the evaluation results for MLSAEs trained on Pythia models with our default hyperparameters. The FVU, delta cross-entropy (CE) loss, and KL divergence remain consistent across model sizes. In most cases, applying tuned-lens transformations decreases the FVU and delta CE loss but not the KL divergence (see Section 4.4 and Figure 12). We provide results for other hyperparameters and breakdowns by the layer of the input activation vectors in Appendix B.

4.2 Representation drift

Guided by the residual stream perspective (Elhage et al., 2021; Ferrando et al., 2024), we expected dense activation vectors to be relatively similar across layers. As an approximate measure of the degree to which information is preserved in the residual stream, we computed the cosine similarities between the activation vectors at adjacent layers, similarly to Lad et al. (2024, Appendix A).

[2] https://huggingface.co/spaces/AlignmentResearch/tuned-lens

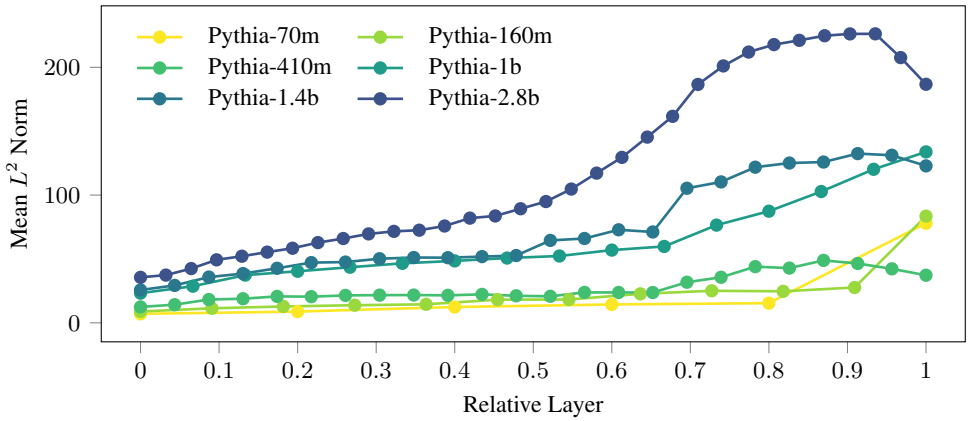


Figure 4: The mean L^2 norm of the residual stream activation vectors at every layer, over 10 million tokens from the test set. To compare transformers with different numbers of layers, we divide the layer index ℓ by the number of layers n_L. This 'relative layer' is the x-axis of the plot.

A similarity of one means that the information represented at a token position is unchanged by the intervening residual block, whereas a similarity of zero means the activation vectors on either side of the block are orthogonal. We had expected changes in the residual stream to become smaller as the model size increased, and we confirmed that the mean cosine similarities increased as the model size increased (Figure 1).

Given that the residual stream activation vectors are relatively similar between adjacent layers, we expected to find many MLSAE latents active at multiple layers. We confirmed this prediction over a large sample of 10 million tokens from the test set (Figure 2). Interestingly, we found that for individual prompts, a much greater proportion of latents are active at only a single layer (Figure 3).

Following Heimersheim & Turner (2023), we verified that the mean L^2 norm of the activation vectors increases across layers, which prompted us to center the vectors at each layer before computing the similarities between them (Figure 4).

4.3 Latent distributions over layers

Given a dataset and MLSAE, each combination of a token and latent produces a distribution of activations over layers. We want to understand the degree to which the variance of that distribution depends on the token versus the latent to quantify the intuition gleaned from Figures 2 and 3.

Consider the layer index L, token T, and latent index J to be random variables. We take $P(J)$ to be a uniform discrete distribution, $P(T \mid J)$ to be a uniform discrete distribution over tokens for which the latent is active (at any layer), and L to be sampled from a conditional distribution proportional to the total latent activation at that layer, aggregating over tokens:

$$P(L = \ell \mid T = t,\, J = j) = \frac{h_j(\mathbf{x}_{t,\ell})}{\sum_{\ell'} h_j(\mathbf{x}_{t,\ell'})} \tag{10}$$

Here, $\mathbf{x}_{t,\ell}$ is the dense residual stream activation vector at token t and layer ℓ, while $h_j(\mathbf{x}_{t,\ell})$ is the activation of the j-th MLSAE latent at that token and layer.

We order latents in all heatmaps using the expected value of the layer index for a single latent $\mathbb{E}[L \mid J = j]$. The variance of the distribution over layers measures the degree to which a latent is active at a single layer (in which case, it is zero) versus multiple layers (in which case, it is positive). We are interested in the following variances of the distribution over layers:

- $\mathrm{Var}[L \mid J = j,\, T = t]$, for a single latent and token
- $\mathrm{Var}[L \mid J = j]$, for a single latent, aggregating over tokens
- $\mathrm{Var}[L]$, aggregating over both latents and tokens

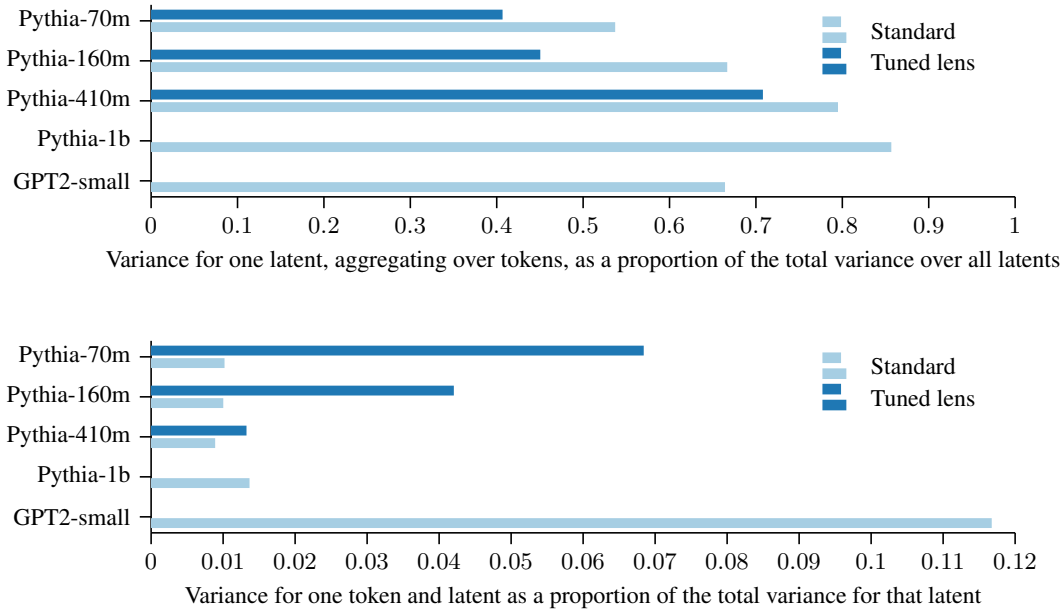


Figure 5: The fraction of the total variance explained by individual latents and the fraction of the variance for an individual latent explained by individual tokens (Equations 11 and 12) for MLSAEs with an expansion factor of $R = 64$ and sparsity $k = 32$, over 10 million tokens from the test set. Importantly, the absence of bars for tuned-lens MLSAEs trained on Pythia-1b and GPT-2 small indicates the absence of results, not that the values are zero.

These quantities are related by the law of total variance (see Appendix E.1). For the moment, we note that the variance of the distribution over layers naturally depends on the number of layers n_L. Hence, to compare different models, we look at ratios between these variances:

$$\begin{array}{c}\text{Variance for one latent, aggregating over tokens,}\\ \text{as a proportion of the total variance over all latents}\end{array} = \frac{\mathbb{E}[\mathrm{Var}(L \mid J)]}{\mathrm{Var}(L)} \tag{11}$$

$$\begin{array}{c}\text{Variance for one token and latent as a}\\ \text{proportion of the total variance for that latent}\end{array} = \frac{\mathbb{E}[\mathrm{Var}(L \mid J, T)]}{\mathbb{E}[\mathrm{Var}(L \mid J)]} \tag{12}$$

The former measures the degree to which latents are active at multiple layers when aggregating over tokens, and the latter compares this to the case for a single token.

The degree to which latents are active at multiple layers when aggregating over tokens is relatively large, between 54 and 86%, and increases uniformly with the model size for fixed hyperparameters (Figure 5). This measure quantifies the observation that, in the aggregate heatmaps (Figure 2), the distributions of latent activations over layers become more 'spread out' as the model size increases. Conversely, we find that the fraction of the variance for an individual latent explained by individual tokens is very small, about 1%. This quantifies the observation that, in the single-prompt heatmaps (Figure 3), the distributions over layers are much less 'spread out' than in the aggregate heatmaps.

4.4 Tuned lens

Thus far, we have assumed that the residual stream basis is the same at every layer. We relaxed this assumption by applying pre-trained tuned-lens transformations to the residual stream activations at each layer before the encoder (Section 3.3). We had expected that these transformations would increase the degree to which latents were active at multiple layers because they translate the activations at every layer into a basis more similar to the basis of the output layer. The aggregate and single-prompt heatmaps (Figures 6 and 7) indicate a modest increase in the degree to which latents are active at multiple layers compared with the standard approach.

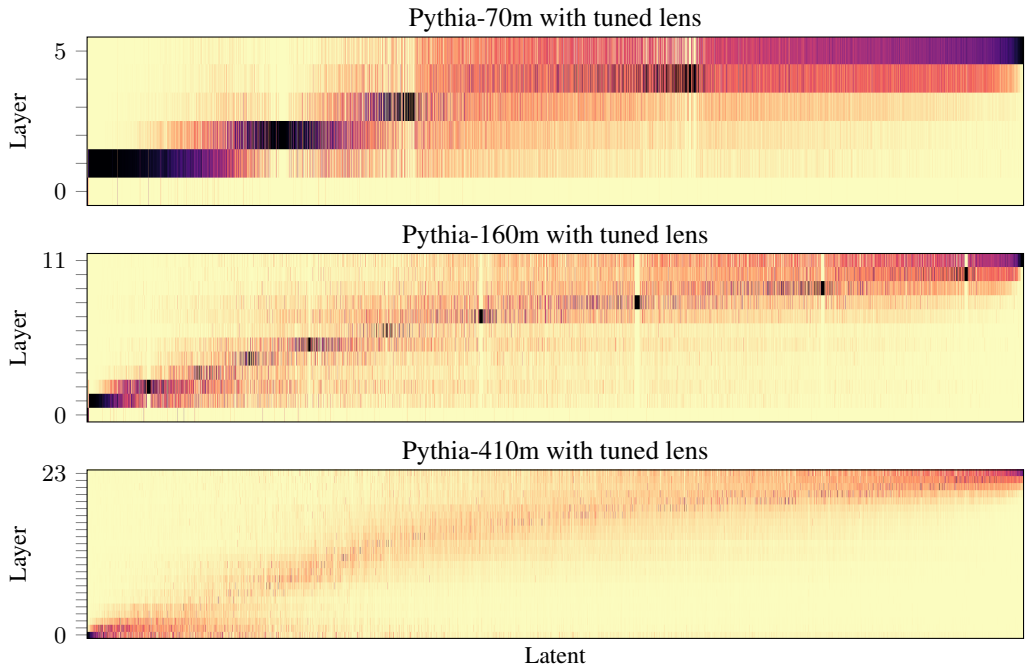


Figure 6: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$. For standard MLSAEs, see Figure 2. We note that a pre-trained tuned lens was not available for Pythia-1b (Section 3.3).

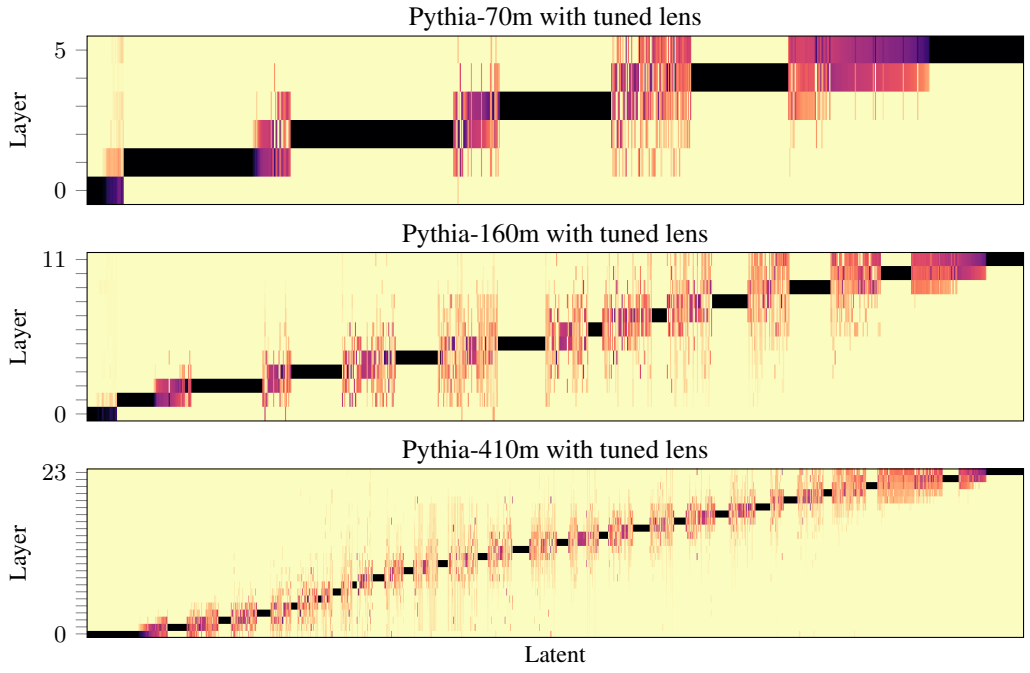


Figure 7: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). For standard MLSAEs, see Figure 3. We note that a pre-trained tuned lens was not available for Pythia-1b (Section 3.3).

The variance ratios in Figures 5 and 22 clarify that the tuned-lens approach decreases the degree to which latents are active at multiple layers when aggregating over tokens. This ratio remains approximately constant as the expansion factor increases (between 37% and 41%). Conversely, the variances for a single token relative to a single latent are larger, i.e., the single-prompt heatmaps are more 'spread out' compared with the standard approach, except for Pythia-410m.

5 Discussion

We considered the activation vectors from different layers as different training examples, so we passed $n_L n_T$ vectors of length d to the autoencoder, where n_T is the number of tokens, n_L is the number of layers, and d is the dimension of the residual stream. This approach might be called a 'data-stacked' MLSAE. An alternative would be a 'feature-stacked' MLSAE, i.e., to concatenate the activation vectors from different layers into a single vector of dimension $n_L d$. This alternative might be better suited to capturing the notion of 'cross-layer superposition,' which we take to mean a small number of simultaneously active sparse features at multiple layers encoding a single meaningful concept (Olah, 2024; Templeton et al., 2024).

We began by pursuing the feature-stacked approach but discarded it. The essential issue is that a single set of sparse features describes the residual stream activations at every layer, which makes it difficult to understand how information flows through a transformer. For example, it would not be possible to plot the activations of sparse features across layers. Moreover, to compute this set of features, one must first compute the activations at every layer, which makes it more difficult to evaluate performance by traditional measures like single-layer reconstruction errors. Finally, the information encoded at one token position may differ substantially between layers due to self-attention. In the early layers, the representation is likely to primarily encode the input token and position embedding, whereas in the later layers, the representation may encode more complex properties of the surrounding context. It is not immediately apparent that jointly encoding this information by a single SAE is sensible. Instead, one might wish to separately capture the different information present at a token position across layers, which is allowed with our data-stacked approach.

6 Conclusion

We introduced the multi-layer SAE (MLSAE), where we train a single SAE on the activations at every layer of the residual stream. This allowed us to study both how information is represented within a single transformer layer and how information flows through the residual stream.

We confirmed that residual stream activations are relatively similar across layers by looking at cosine similarities before considering the distributions of latent activations over layers. When aggregating over a large sample of ten million tokens, we observed that most latents were active at multiple layers, but for a single prompt, most latent activations were isolated to a single layer. To quantify these observations, we computed the fraction of the total variance explained by individual latents and the fraction of the variance for an individual latent explained by individual tokens. This analysis confirmed that the degree to which latents are active at multiple layers when aggregating over tokens was large, increasing with the model size and expansion factor, and that the fraction of the variance explained by individual tokens was small.

Understanding how representations change as they flow through transformers is critical to identifying meaningful circuits, which is a core task of mechanistic interpretability. Despite the utility of the residual stream perspective, our results demonstrate that representation drift, and perhaps the increasing magnitude of changes to the residual stream across layers, is a significant obstacle to identifying meaningful computational variables with SAEs. Nevertheless, we argue that an approach such as the MLSAE, which considers the representations at multiple layers in parallel, is necessary for future methods that seek to interpret the internal computations of transformer language models.

References

Anthony J. Bell and Terrence J. Sejnowski. An Information-Maximization Approach to Blind Separation and Blind Deconvolution. *Neural Computation*, 7(6):1129–1159, November 1995.

ISSN 0899-7667. doi: 10.1162/neco.1995.7.6.1129. URL https://ieeexplore.ieee.org/abstract/document/6796129. Conference Name: Neural Computation.

Anthony J. Bell and Terrence J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, December 1997. ISSN 0042-6989. doi: 10.1016/S0042-6989(97)00121-1. URL https://www.sciencedirect.com/science/article/pii/S0042698997001211.

Nora Belrose. EleutherAI/sae, May 2024. URL https://github.com/EleutherAI/sae.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting Latent Predictions from Transformers with the Tuned Lens, November 2023. URL http://arxiv.org/abs/2303.08112. arXiv:2303.08112 [cs].

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 2397–2430. PMLR, July 2023. URL https://proceedings.mlr.press/v202/biderman23a.html. ISSN: 2640-3498.

Dan Braun, Jordan Taylor, Nicholas Goldowsky-Dill, and Lee Sharkey. Identifying Functionally Important Features with End-to-End Sparse Dictionary Learning, May 2024. URL http://arxiv.org/abs/2405.12241. arXiv:2405.12241 [cs].

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, and Amanda Askell. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features.

Maheep Chaudhary and Atticus Geiger. Evaluating Open-Source Sparse Autoencoders on Disentangling Factual Knowledge in GPT-2 Small, September 2024. URL http://arxiv.org/abs/2409.04478. arXiv:2409.04478 [cs].

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards Automated Circuit Discovery for Mechanistic Interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Abstract-Conference.html.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models, October 2023. URL http://arxiv.org/abs/2309.08600. arXiv:2309.08600 [cs].

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders Find Interpretable LLM Feature Circuits, June 2024. URL http://arxiv.org/abs/2406.11944. arXiv:2406.11944 [cs].

Nelson Elhage, Neel Nanda, Catherine Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, and T. Conerly. A Mathematical Framework for Transformer Circuits, 2021. URL https://transformer-circuits.pub/2021/framework/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, September 2022. URL http://arxiv.org/abs/2209.10652. arXiv:2209.10652 [cs].

Joshua Engels, Isaac Liao, Eric J. Michaud, Wes Gurnee, and Max Tegmark. Not All Language Model Features Are Linear, May 2024. URL http://arxiv.org/abs/2405.14860. arXiv:2405.14860 [cs].

Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R. Costa-jussà. A Primer on the Inner Workings of Transformer-based Language Models, May 2024. URL http://arxiv.org/abs/2405.00208. arXiv:2405.00208 [cs].

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling, December 2020. URL http://arxiv.org/abs/2101.00027. arXiv:2101.00027 [cs].

Leo Gao, Tom Dupré la Tour, and Jeffrey Wu. openai/sparse_autoencoder, December 2023. URL https://github.com/openai/sparse_autoencoder.

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, June 2024. URL http://arxiv.org/abs/2406.04093. arXiv:2406.04093 [cs].

Jorge García-Carrasco, Alejandro Maté, and Juan Carlos Trujillo. How does GPT-2 Predict Acronyms? Extracting and Understanding a Circuit via Mechanistic Interpretability. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, pp. 3322–3330. PMLR, April 2024. URL https://proceedings.mlr.press/v238/garcia-carrasco24a.html. ISSN: 2640-3498.

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, February 2024. URL http://arxiv.org/abs/2402.12201. arXiv:2402.12201 [cs].

Stefan Heimersheim and Alex Turner. Residual stream norms grow exponentially over the forward pass, May 2023. URL https://www.alignmentforum.org/posts/8mizBCm3dyc432nK8/residual-stream-norms-grow-exponentially-over-the-forward.

Evan Hernandez, Arnab Sen Sharma, Tal Haklay, Kevin Meng, Martin Wattenberg, Jacob Andreas, Yonatan Belinkov, and David Bau. Linearity of Relation Decoding in Transformer Language Models, February 2024. URL http://arxiv.org/abs/2308.09124. arXiv:2308.09124 [cs].

Jing Huang, Zhengxuan Wu, Christopher Potts, Mor Geva, and Atticus Geiger. RAVEL: Evaluating Interpretability Methods on Disentangling Language Model Representations, August 2024. URL http://arxiv.org/abs/2402.17700. arXiv:2402.17700 [cs].

A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411–430, June 2000. ISSN 0893-6080. doi: 10.1016/S0893-6080(00)00026-5. URL https://www.sciencedirect.com/science/article/pii/S0893608000000265.

Stanisław Jastrzębski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual Connections Encourage Iterative Inference, March 2018. URL http://arxiv.org/abs/1710.04773. arXiv:1710.04773 [cs].

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

Connor Kissane, Robert Krzyzanowski, Joseph Isaac Bloom, Arthur Conmy, and Neel Nanda. Interpreting Attention Layer Outputs with Sparse Autoencoders. June 2024. URL https://openreview.net/forum?id=fewUBDwjji.

Kishore Konda, Roland Memisevic, and David Krueger. Zero-bias autoencoders and the benefits of co-adapting features, April 2015. URL http://arxiv.org/abs/1402.3337. arXiv:1402.3337 [cs, stat].

Vedang Lad, Wes Gurnee, and Max Tegmark. The Remarkable Robustness of LLMs: Stages of Inference?, June 2024. URL http://arxiv.org/abs/2406.19384. arXiv:2406.19384 [cs].

Quoc Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Ng. ICA with Reconstruction Cost for Efficient Overcomplete Feature Learning. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/hash/233509073ed3432027d48b1a83f5fbd2-Abstract.html.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/hash/2d71b2ae158c7c5912cc0bbde2bb9d95-Abstract.html.

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, August 2024. URL http://arxiv.org/abs/2408.05147. arXiv:2408.05147 [cs].

Aleksandar Makelov. Sparse Autoencoders Match Supervised Features for Model Steering on the IOI Task. June 2024. URL https://openreview.net/forum?id=JdrVuEQih5.

Alireza Makhzani and Brendan Frey. k-Sparse Autoencoders, March 2014. URL http://arxiv.org/abs/1312.5663. arXiv:1312.5663 [cs].

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, March 2024. URL http://arxiv.org/abs/2403.19647. arXiv:2403.19647 [cs].

Andrew Ng. Sparse autoencoder, 2011. URL https://graphics.stanford.edu/courses/cs233-21-spring/ReferencedPapers/SAE.pdf.

nostalgebraist. Interpreting GPT: the logit lens, August 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Chris Olah. The Next Five Hurdles, July 2024. URL https://transformer-circuits.pub/2024/july-update/index.html#hurdles.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom In: An Introduction to Circuits. *Distill*, 5(3), March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.001. URL https://distill.pub/2020/circuits/zoom-in.

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 1996. ISSN 1476-4687. doi: 10.1038/381607a0. URL https://www.nature.com/articles/381607a0. Publisher: Nature Publishing Group.

Charles O'Neill and Thang Bui. Sparse Autoencoders Enable Scalable and Reliable Circuit Identification in Language Models, May 2024. URL http://arxiv.org/abs/2405.12522. arXiv:2405.12522 [cs].

Charles O'Neill, Christine Ye, Kartheik Iyer, and John F. Wu. Disentangling Dense Embeddings with Sparse Autoencoders, August 2024. URL http://arxiv.org/abs/2408.00657. arXiv:2408.00657 [cs].

Kiho Park, Yo Joong Choe, and Victor Veitch. The Linear Representation Hypothesis and the Geometry of Large Language Models, November 2023. URL http://arxiv.org/abs/2311.03658. arXiv:2311.03658 [cs, stat].

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, Janos Kramar, Rohin Shah, and Neel Nanda. Improving Sparse Decomposition of Language Model Activations with Gated Sparse Autoencoders. June 2024a. URL https://openreview.net/forum?id=Ppj5KvzU8Q.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping Ahead: Improving Reconstruction Fidelity with JumpReLU Sparse Autoencoders, July 2024b. URL http://arxiv.org/abs/2407.14435. arXiv:2407.14435 [cs].

Lee Sharkey, Dan Braun, and Beren Millidge. Taking features out of superposition with sparse autoencoders, December 2022. URL https://www.alignmentforum.org/posts/z6QQJbtpkEAX3Aojj/interim-research-report-taking-features-out-of-superposition.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. Extracting Latent Steering Vectors from Pretrained Language Models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 566–581, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.48. URL https://aclanthology.org/2022.findings-acl.48.

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, Alex Tamkin, Esin Durmus, Tristan Hume, Francesco Mosconi, 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, May 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Lucrezia Valeriani, Diego Doimo, Francesca Cuturello, Alessandro Laio, Alessio Ansuini, and Alberto Cazzaniga. The geometry of hidden representations of large transformer models. *Advances in Neural Information Processing Systems*, 36:51234–51252, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a0e66093d7168b40246af1cddc025daa-Abstract-Conference.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small, November 2022. URL http://arxiv.org/abs/2211.00593. arXiv:2211.00593 [cs].

Martin Wattenberg and Fernanda Viégas. Relational Composition in Neural Networks: A Survey and Call to Action. June 2024. URL https://openreview.net/forum?id=zzCEiUIPk9.

James C. R. Whittington, Will Dorrell, Surya Ganguli, and Timothy E. J. Behrens. Disentanglement with Biological Constraints: A Theory of Functional Cell Types, March 2023. URL http://arxiv.org/abs/2210.01768. arXiv:2210.01768 [cs, q-bio].

John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 1 edition, January 2022. ISBN 978-1-108-77930-2 978-1-108-48973-7. doi: 10.1017/9781108779302. URL https://www.cambridge.org/highereducation/product/9781108779302/book.

Zeyu Yun, Yubei Chen, Bruno Olshausen, and Yann LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. In Eneko Agirre, Marianna Apidianaki, and Ivan Vulić (eds.), *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 1–10, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.deelio-1.1. URL https://aclanthology.org/2021.deelio-1.1.

A TRAINING

We train each autoencoder on 1 billion tokens from the Pile (Gao et al., 2020), excluding the copyrighted Books3 dataset,[3] for a single epoch. Specifically, we concatenate a batch of 1024 text samples with the end-of-sentence token, tokenize the concatenated text, and divide the output into sequences of 2048 tokens, discarding the final incomplete sequence. We use an effective batch size of 131072 tokens (64 sequences) for all experiments.

We do not compute activation vectors and cache them to disk before training, which minimizes storage overhead at the expense of repeated computation. We construct a batch of activation vectors to input to the autoencoder by performing the forward pass of the underlying transformer for a sequence of tokens, collecting the residual stream activation vectors at every layer, and stacking them together. Following Lieberum et al. (2024), we exclude activation vectors corresponding to special tokens (end-of-sentence, beginning-of-sentence, and padding). Hence, each batch has an equal number of activation vectors from each layer, which is the number of non-special tokens.

Following the optimization guidelines in Bricken et al. (2023); Gao et al. (2024), we initialize the pre-encoder bias $\mathbf{b}_{\text{pre}}$ to the geometric median of the first training batch; we initialize the decoder weight matrix $\mathbf{W}_{\text{dec}}$ to the transpose of the encoder $\mathbf{W}_{\text{enc}}$; we scale the decoder weight vectors to unit norm at initialization and after each training step; and we remove the component of the gradient of the decoder weight matrix parallel to its weight vectors after each training step.

We use the Adam optimizer (Kingma & Ba, 2017) with the default β parameters, a constant learning rate of 1×10^{-4}, and $\epsilon = 6.25 \times 10^{-10}$. Unlike Gao et al. (2024), we do not use gradient clipping or weight averaging, and we use FP16 mixed precision to reduce memory use.

B EVALUATION

B.1 RECONSTRUCTION ERROR AND SPARSITY

While we use the FVU instead of MSE as the reconstruction error in the training loss, we record both metrics for the inputs from each transformer layer and the mean over all layers (Figure 8). The L^0 norm of the latents is fixed at k per activation vector (layer and token), but we record the L^1 norm (Figure 9). We report the values of these metrics over one million tokens from the test set.

For Pythia-70m, the FVU at each layer is comparable to Marks et al. (2024, p. 21), who trained separate SAEs with $n = 32768$ and L^0 norms between 54 and 108, as well as Cunningham et al. (2023, p. 13). For Pythia-160m, the FVU is similar to Gao et al. (2024), who report the normalized MSE on layer 8 of GPT-2 small.

B.2 DOWNSTREAM LOSS

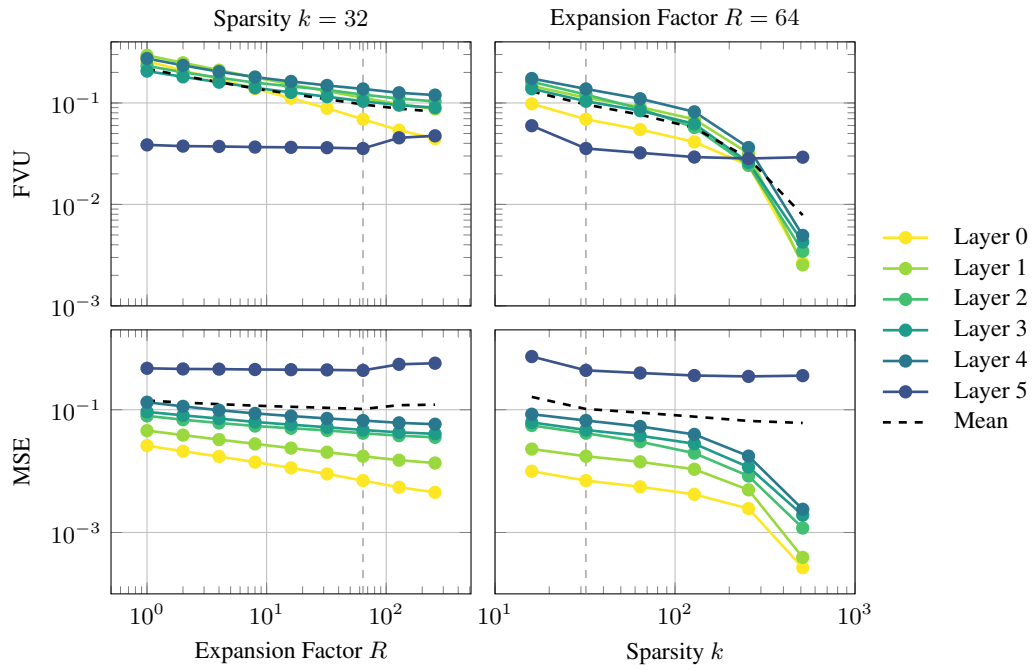
In addition to the increase in cross-entropy (CE) loss, we record the Kullback-Leibler (KL) divergence between probability distributions when the residual stream activations at a given layer are replaced by their reconstruction (Section 4.1). We report the values of these metrics over one million tokens from the test set (Figure 10). The increase in cross-entropy loss is comparable to Marks et al. (2024, p. 21) for Pythia-70m, Gao et al. (2024, p. 5) and Braun et al. (2024) for GPT-2 small, and Lieberum et al. (2024, p. 7-8) for layer 20 of Gemma 2 2B and 9B.

B.3 GPT-2 SMALL

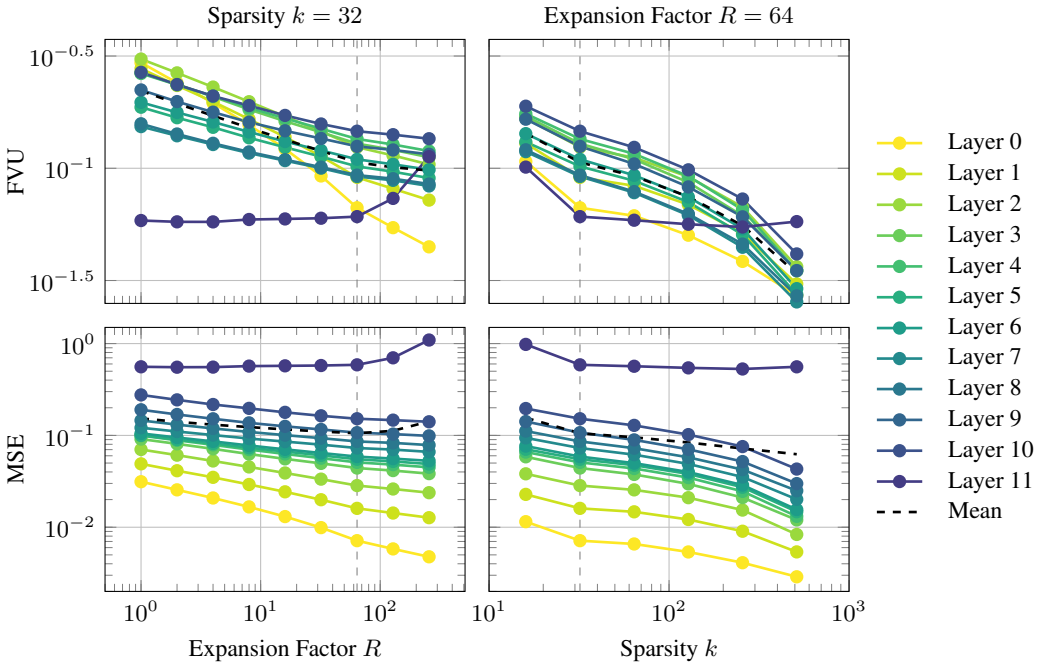
We predominantly study GPT-style models from the Pythia suite (Section 3.1). While we do not expect our results to depend strongly on the underlying transformer architecture, we additionally trained an MLSAE on GPT-2 small (Radford et al., 2019) with our default hyperparameters, i.e., an expansion factor of $R = 64$ and sparsity $k = 32$.

We include quantitative results for GPT-2 small in Table 1, Figure 5, Table 2, and Figure 18; we include heatmaps of the distributions of latent activations over layers in Figures 13 and 14, which are qualitatively similar to Pythia models. We note that GPT-2 small is similar in size to Pythia-160m.

[3] `https://huggingface.co/datasets/monology/pile-uncopyrighted`

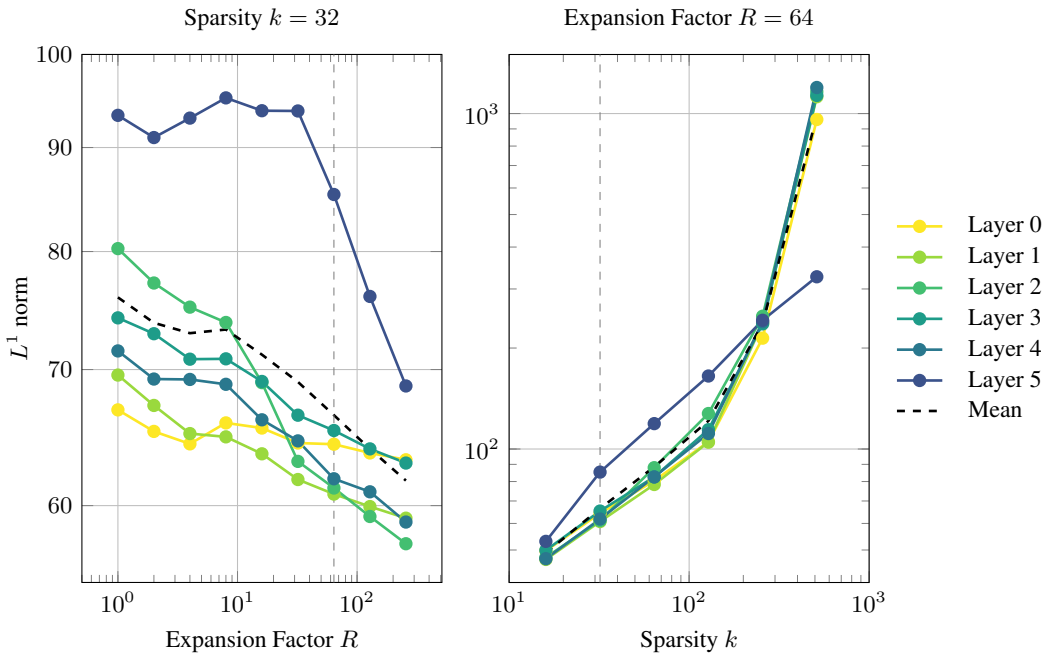


(a) Pythia-70m

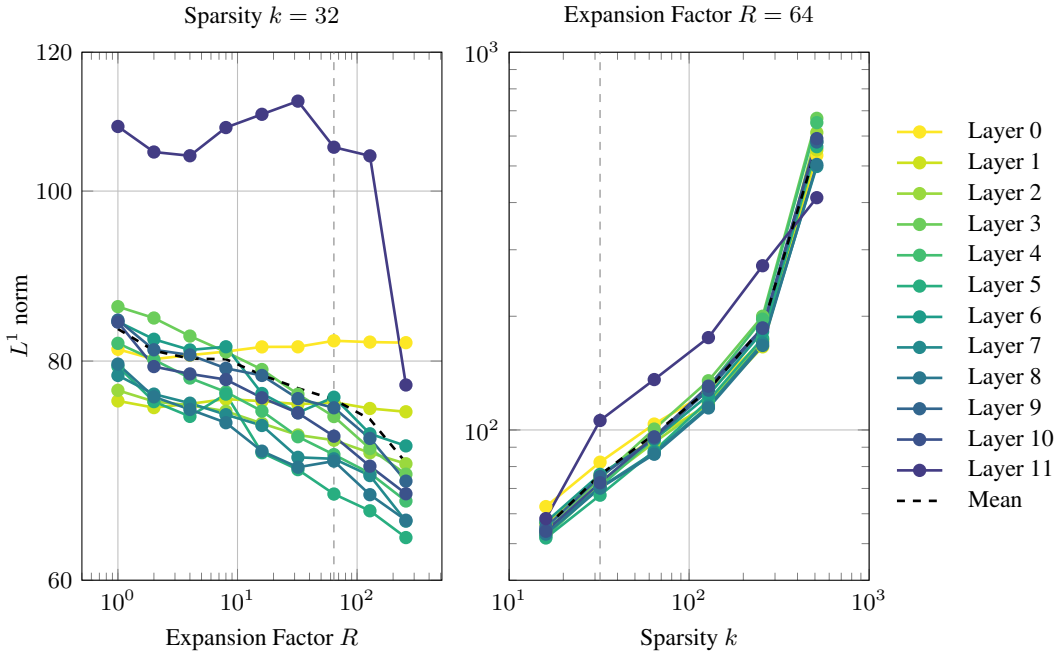


(b) Pythia-160m

Figure 8: With fixed sparsity $k = 32$, the FVU and MSE generally decrease as the expansion factor R increases. For inputs from the last layer, they increase for the largest expansion factors, which we attribute to fluctuations in the percentage of dead latents (Figure 11). With fixed expansion factor $R = 64$, the FVU and MSE decrease as the sparsity k increases. While all inputs are standardized before passing them to the encoder, the decoder outputs are rescaled afterward. Hence, the MSE increases across layers because it is not divided by the variance of the inputs.

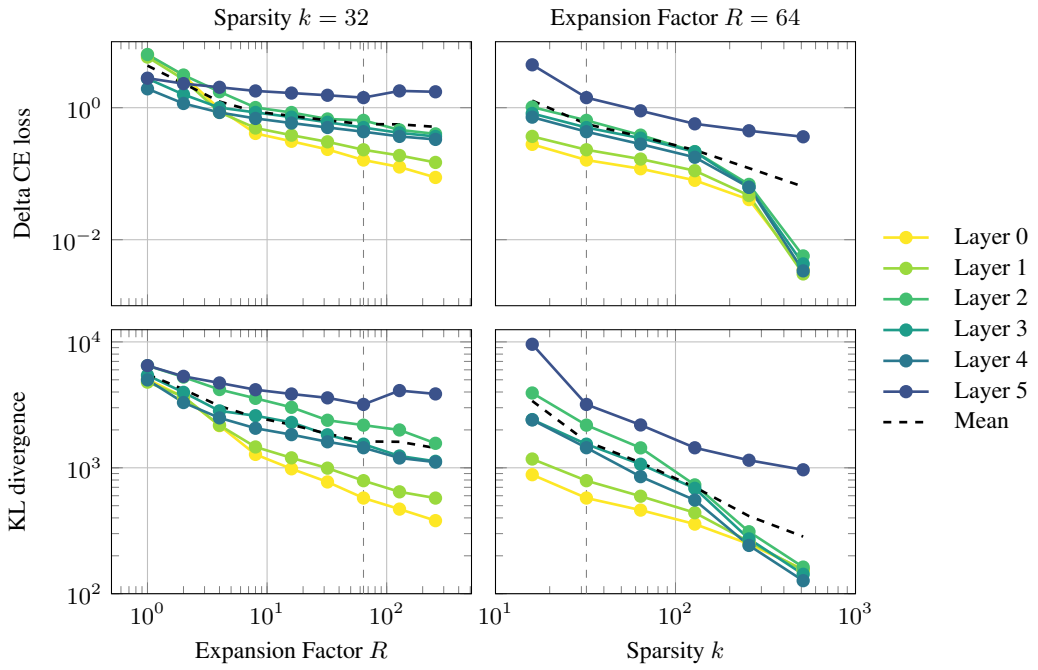


(a) Pythia-70m

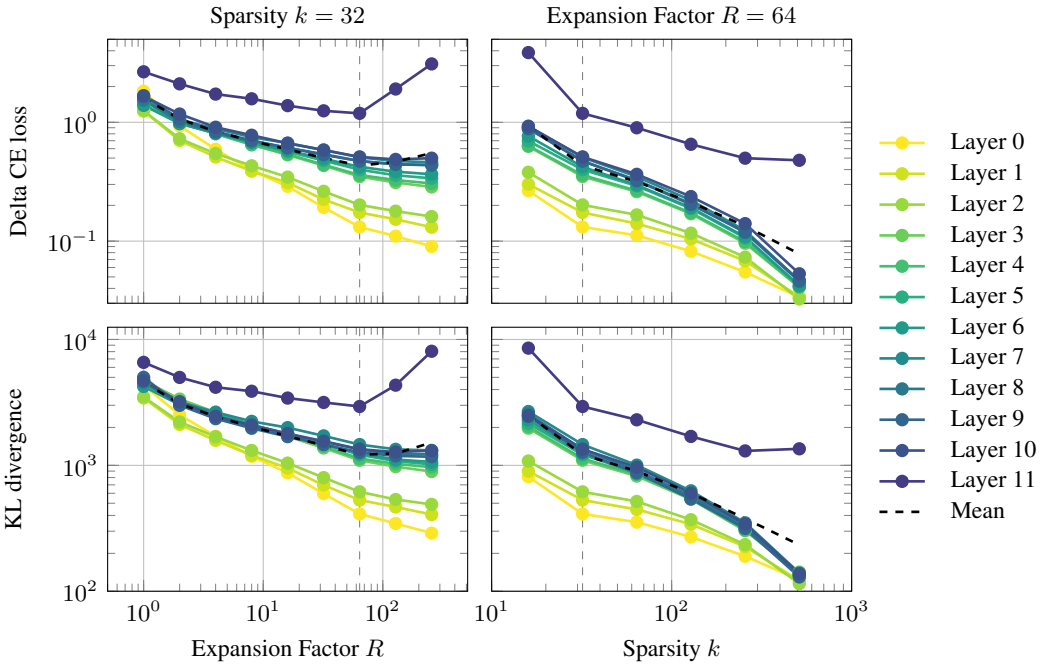


(b) Pythia-160m

Figure 9: With fixed sparsity $k = 32$, the L^1 norm per token (the sum of absolute activations) generally decreases as the expansion factor R increases. With fixed expansion factor $R = 64$, the L^1 norm increases as the sparsity k increases. Recall that the L^0 norm per token (the count of non-zero activations) is fixed at k.



(a) Pythia-70m



(b) Pythia-160m

Figure 10: With fixed sparsity $k = 32$, the delta CE loss and KL divergence generally decrease as the expansion factor increases, except for inputs from the last layer. With fixed expansion factor $R = 64$, both metrics decrease as the sparsity k increases, similarly to the FVU and MSE (Figure 8).

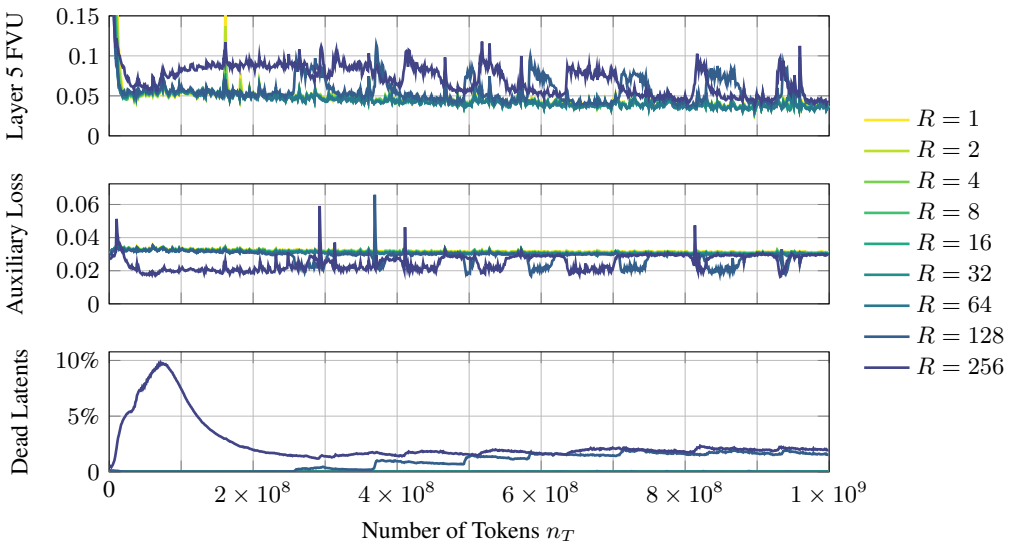


Figure 11: An illustration of the FVU for inputs from the last layer, compared to the auxiliary loss and percentage of dead latents, for MLSAEs trained on Pythia-70m with fixed sparsity $k = 32$. An increase in dead latents correlates with a decrease in the auxiliary loss and an increase in the FVU at the last layer. We attribute this to the increased scale of the inputs because the auxiliary loss depends on the MSE (Figure 8). The auxiliary loss is multiplied by its coefficient $\alpha = 1/32$ in the training loss.

B.4 Single-layer SAEs

While we compare the performance of our multi-layer SAEs to single-layer SAEs from the literature in Appendix B.1 and B.2, we also trained multiple single-layer SAEs on Pythia-70m and 160m, leaving the remainder of the experimental setup unchanged, with our default hyperparameters.

Predictably, we find that a single-layer SAE trained on data from a given layer performs best on test data from the same layer (Figures 15 and 16). A multi-layer SAE trained on data from every layer performs comparably to the corresponding single-layer SAE, and more consistently across test data from different layers. Interestingly, applying the corresponding tuned-lens transformation to the input activations from each layer during training and evaluation degrades the performance of single-layer SAEs on test data from different layers of Pythia-70m, unlike multi-layer SAEs (Figure 12).

Importantly, the results for the last layer are excluded from these figures. This is because we take the residual stream activation vectors after a given layer has been applied (Section 3.1), such that the last-layer activations represent the next-token predictions of the model only and not intermediate computational variables. Hence, we expect these activations to have a significantly different structure to the preceding layers, which could distort our comparisons across layers.

C Latent cosine similarities

Sharkey et al. (2022) define the Mean Max Cosine Similarity (MMCS) between a learned dictionary X and a ground-truth dictionary X'. There is no ground-truth dictionary for language models, so a larger learned dictionary or the k nearest neighbors to each dictionary element are commonly used.

$$\text{MMCS}(X, X') = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \max_{\mathbf{x}' \in X'} \cos \text{sim}(\mathbf{x}, \mathbf{x}') \tag{13}$$

The MMCS serves as a proxy measure for 'feature splitting' (Bricken et al., 2023; Braun et al., 2024): as the number of features increases, we expect the decoder weight vectors to be more similar to their nearest neighbors. We compute the MMCS with $k = 1$ after training, finding it decreases slightly as the model size increases with fixed hyperparameters (Table 2).

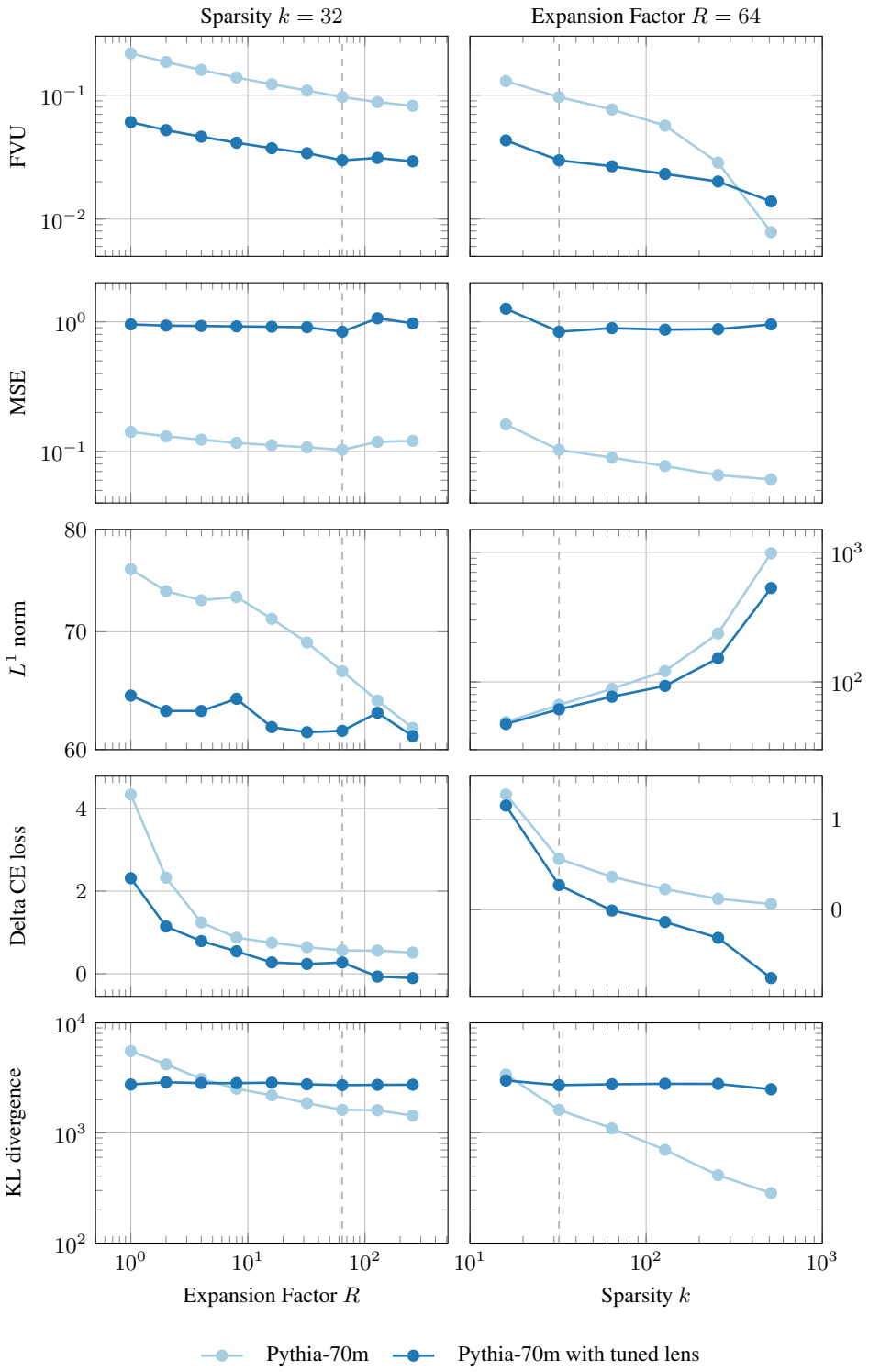


Figure 12: For Pythia-70m, applying tuned-lens transformations decreases the mean FVU and delta cross-entropy loss but not the KL divergence. Importantly, we compute reconstruction errors before applying the inverse transformation and downstream loss metrics afterward (Section 3.3). Unlike Figure 10, we use a linear scale for the delta cross-entropy loss because, surprisingly, it is negative for tuned-lens MLSAEs with a large expansion factor R or sparsity k.

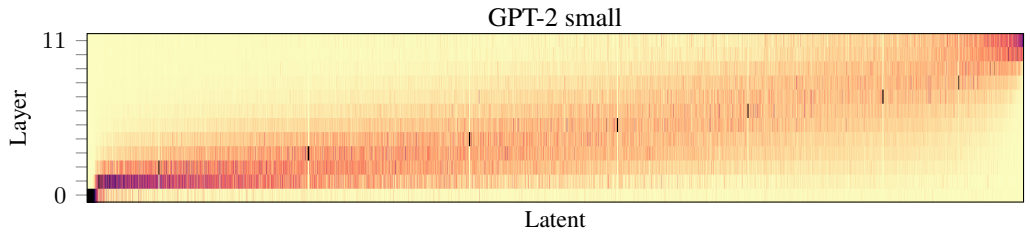


Figure 13: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on GPT-2 small with an expansion factor of $R = 64$. We provide further details in Figure 2.

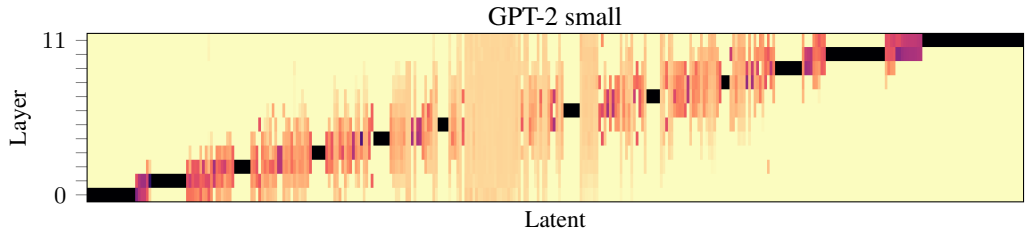


Figure 14: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on GPT-2 small with an expansion factor of $R = 64$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

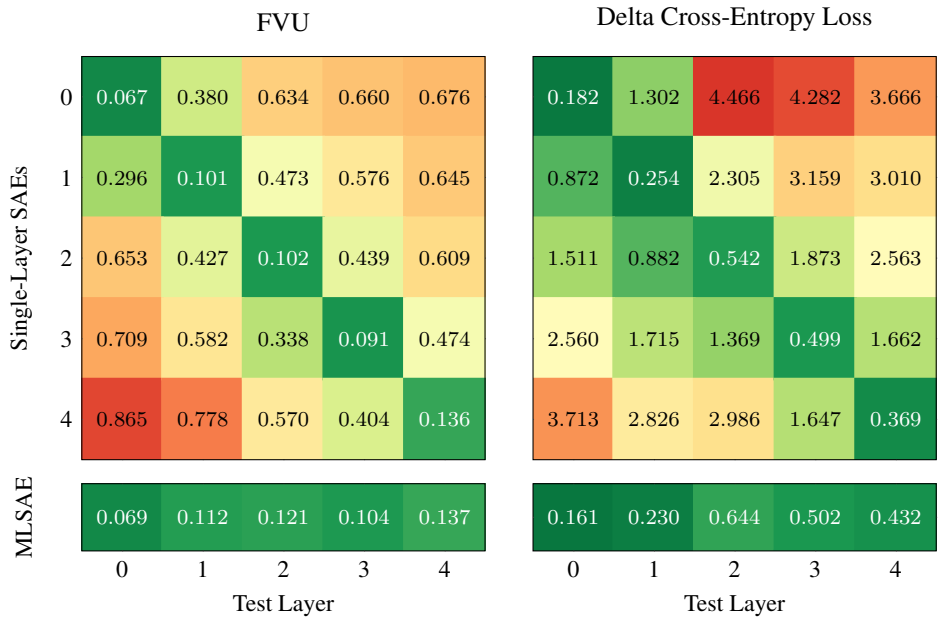


Figure 15: The FVU reconstruction error and delta cross-entropy loss for single-layer SAEs trained on each layer of Pythia-70m, compared with a single multi-layer SAE trained on every layer. The colormap ranges between 0 and 1 for the FVU heatmap and between 0 and 5 for the loss heatmap.

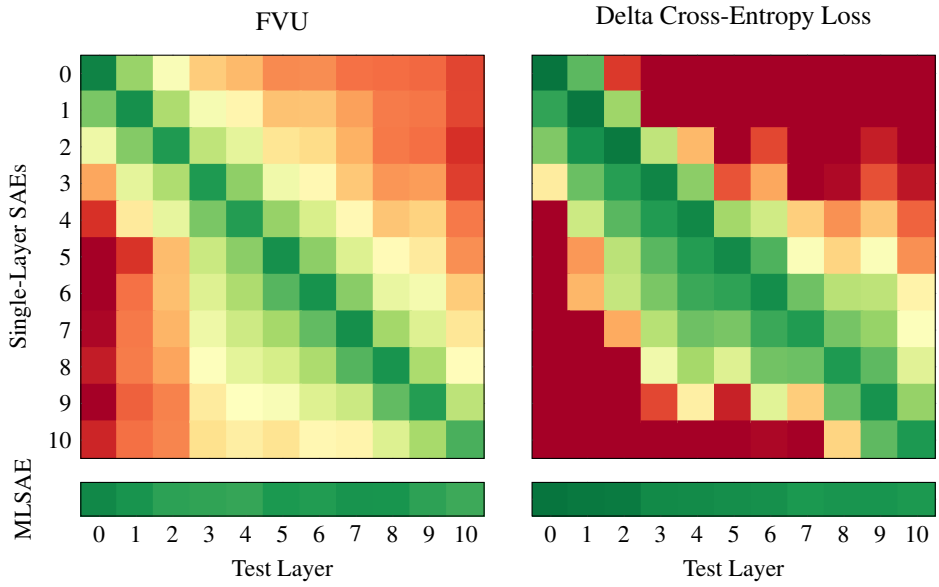


Figure 16: The FVU reconstruction error and delta cross-entropy loss for single-layer SAEs trained on each layer of Pythia-160m, compared with a single multi-layer SAE trained on every layer. We omit the numeric values for brevity, but the colormap ranges between 0 and 1 for the FVU heatmap and between 0 and 5 for the loss heatmap, following Figure 15.

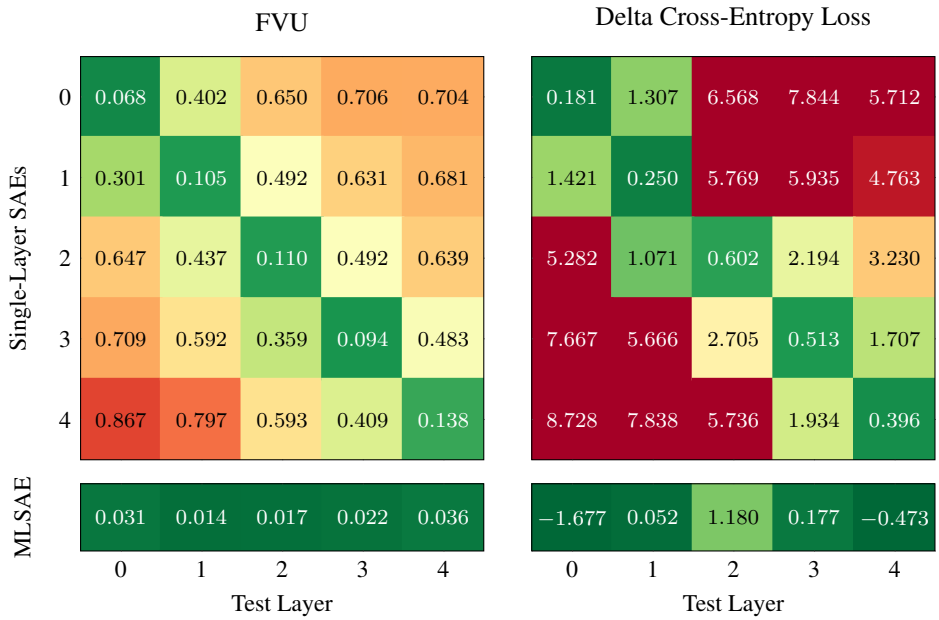


Figure 17: The FVU reconstruction error and delta cross-entropy loss for single-layer SAEs trained on each layer of Pythia-70m compared with a single multi-layer SAE trained on every layer, applying tuned-lens transformations during training and evaluation (Section 3.3). The colormap ranges between 0 and 1 for the FVU heatmap and between 0 and 5 for the loss heatmap, following Figure 15. Notably, the cross-entropy loss decreases for some tuned-lens MLSAEs (Figure 12).

Model	Mean	Std. Dev.
Pythia-70m	0.275	0.0843
Pythia-160m	0.250	0.0928
Pythia-410m	0.221	0.0868
Pythia-1b	0.201	0.0989
GPT-2 small	0.258	0.0703

(a) Without tuned lens

Model	Mean	Std. Dev.
Pythia-70m	0.261	0.0763
Pythia-160m	0.206	0.0734
Pythia-410m	0.216	0.0864

(b) With tuned lens

Table 2: The mean and standard deviation of the maximum cosine similarity between decoder weight vectors for MLSAEs with an expansion factor of $R = 64$ and sparsity $k = 32$.

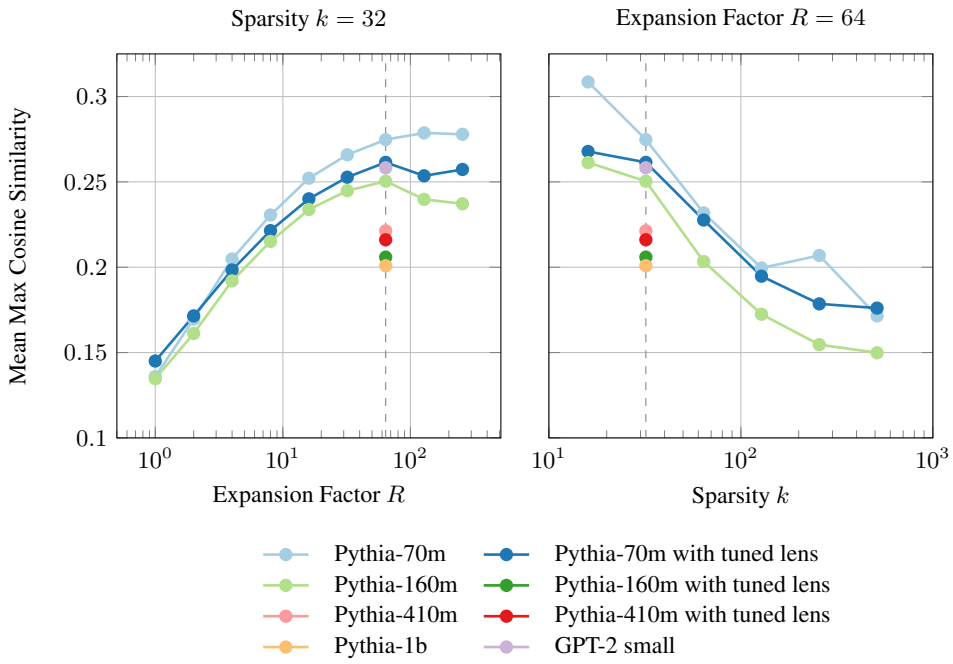


Figure 18: The Mean Max Cosine Similarity between decoder weight vectors for standard and tuned-lens MLSAEs. The MMCS increases as the expansion factor R increases and decreases as the sparsity k increases. Applying tuned-lens transformations tends to slightly decrease the MMCS.

A potential issue when training multi-layer SAEs is that one could learn multiple versions of 'the same' latent that are active at different layers. In this case, we would expect to find pairs of latents with large cosine similarities between their decoder weight vectors but different observed distributions of activations over layers (Section 4.3). We investigated this possibility by comparing the pairwise cosine similarities between decoder weight vectors for trained MLSAEs to reference distributions.

As a negative control, we generated an equal number (the number of latents n) of normal independently and identically distributed (i.i.d.) vectors $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ of the same length (the model dimension d). In this case, the pairwise cosine similarities follow a normal distribution $\cos\text{sim}(\mathbf{x}, \mathbf{x}') \sim \mathcal{N}(0, {}^1/_d)$. As a positive control, we generated a smaller number of normal i.i.d. vectors (the number of latents n divided by the number of layers n_L), copied the vectors n_L times, and added noise $\sim \mathcal{N}(0, 1)$ to each copy. In this case, we expect an additional frequency peak for large, positive cosine similarities.

Figure 19 shows that the distributions of pairwise cosine similarities for decoder weight vectors are slightly heavier-tailed and right-shifted compared with the negative control, i.e., a pair of MLSAE latents are slightly more likely to have high cosine similarity than a pair of i.i.d. normal vectors. However, the number of pairs with large, positive cosine similarities is small compared to the positive control, which has a second peak around 0.5 (not visible).

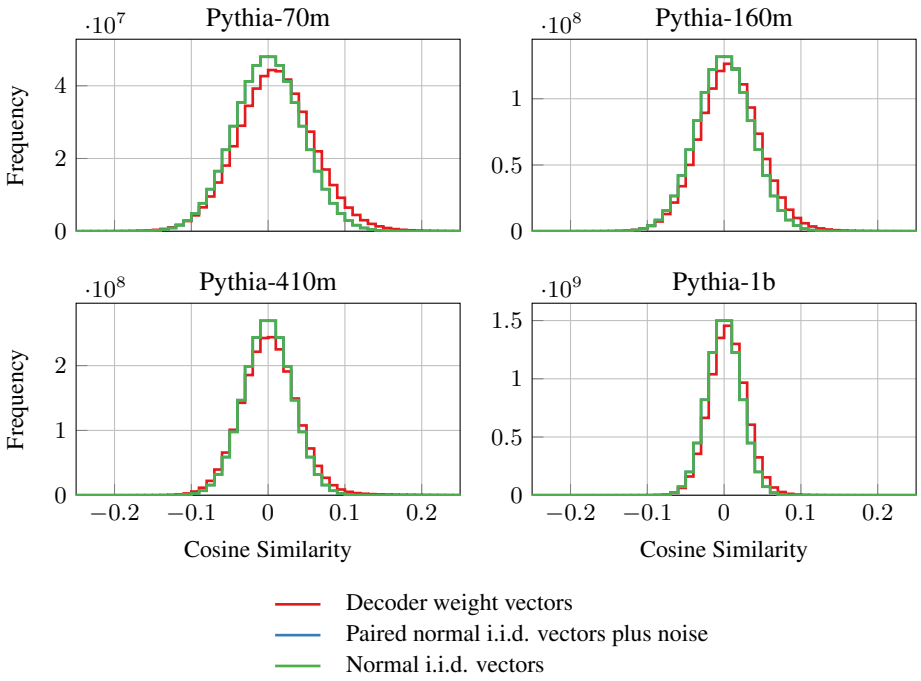


Figure 19: Histograms of the frequencies of pairwise cosine similarities between decoder weight vectors, compared to an equal number of normal i.i.d. vectors of the same length, and n_L copies of a smaller number of normal i.i.d. vectors with added noise. Here, we report the frequencies for MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$.

D Normalizing latent activations

In the aggregate and single-prompt heatmaps such as Figures 2 and 3, we plot the distributions of latent activations over layers, taken to be proportional to the total activations when aggregating over tokens (Eq. 10). We chose to normalize the latent activations in this way to visually compare the aggregate and single-prompt heatmaps, as well as individual latents within a heatmap, which is beneficial due to the wide range of activation counts and totals across latents.

Normalizing the activations discards the relative frequencies and magnitudes of activations for different latents, so we reproduce Figures 2 and 3 with the un-normalized totals of latent activations in Figures 20 and 21. We use power-law normalization for the colormaps, i.e., $y = x^\gamma$ where $\gamma = 1/4$ to account for the wide range of values; all other heatmaps have linear colormaps. As with all other single-prompt heatmaps, we exclude latents from Figure 21 that never activate.

Interestingly, the un-normalized single-prompt heatmaps suggest that latents with relatively large total activations are more likely to have a bimodal distribution of activations over layers, i.e., to be active at non-adjacent layers. However, this is not apparent in the equivalent aggregate heatmaps, and it is unclear whether the effect is reproducible across example prompts and hyperparameters.

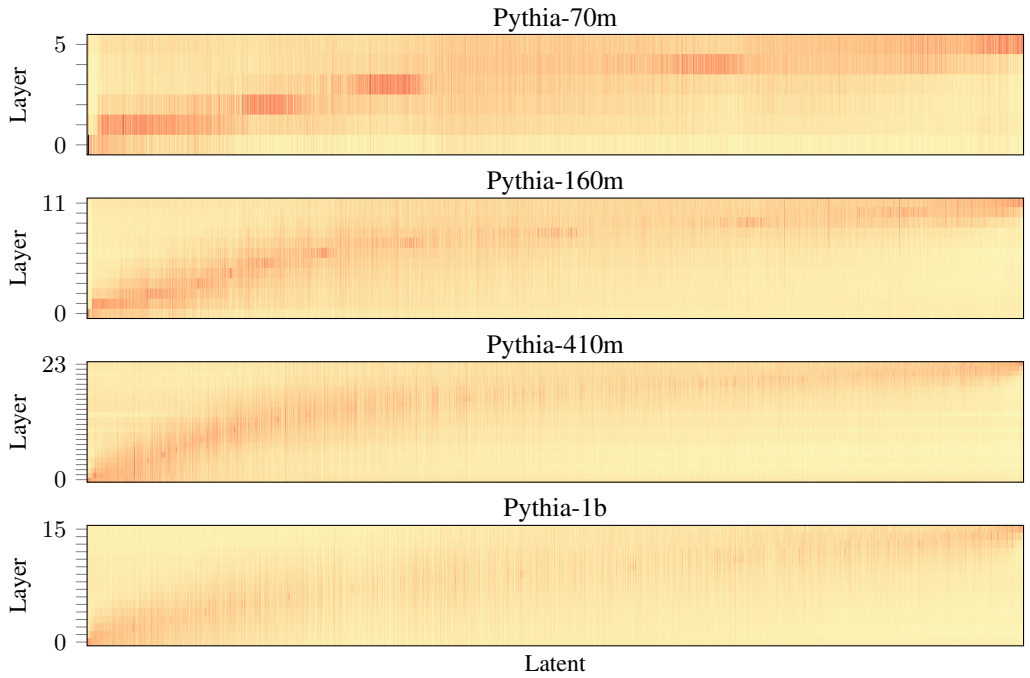


Figure 20: Heatmaps of the total latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the totals for MLSAEs trained on Pythia models with an expansion factor of $R = 64$ and sparsity $k = 32$. We provide further details in Figure 2. The colormaps use power-law normalization with $\gamma = 1/4$.

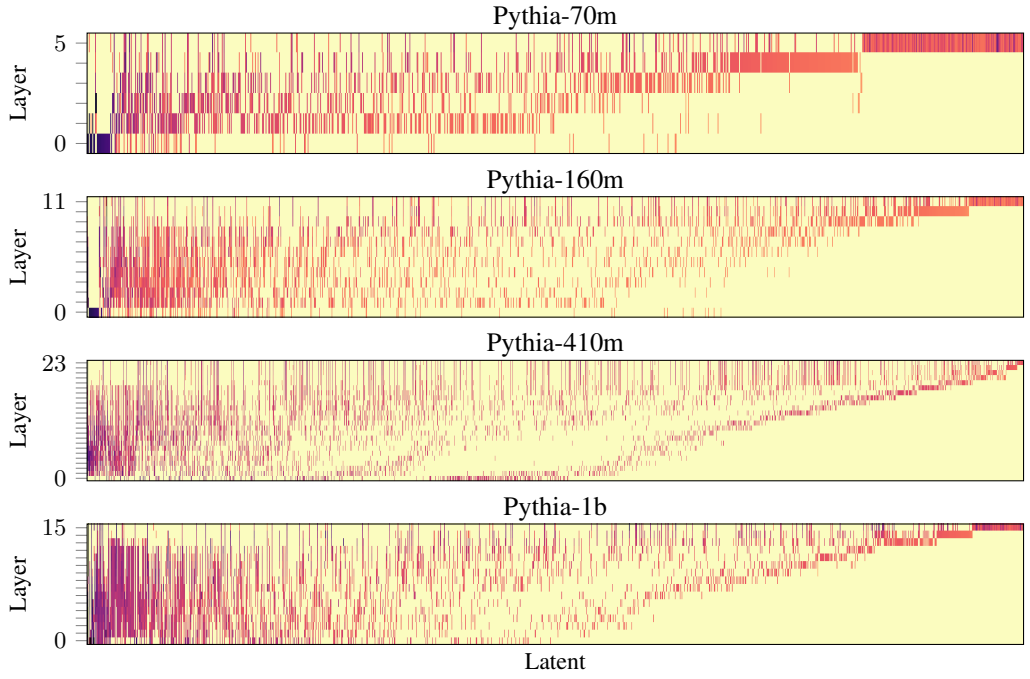


Figure 21: Heatmaps of the total latent activations over layers for a single example prompt. Here, we plot the totals for MLSAEs with an expansion factor of $R = 64$ and sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3. The colormaps use power-law normalization with $\gamma = 1/4$.

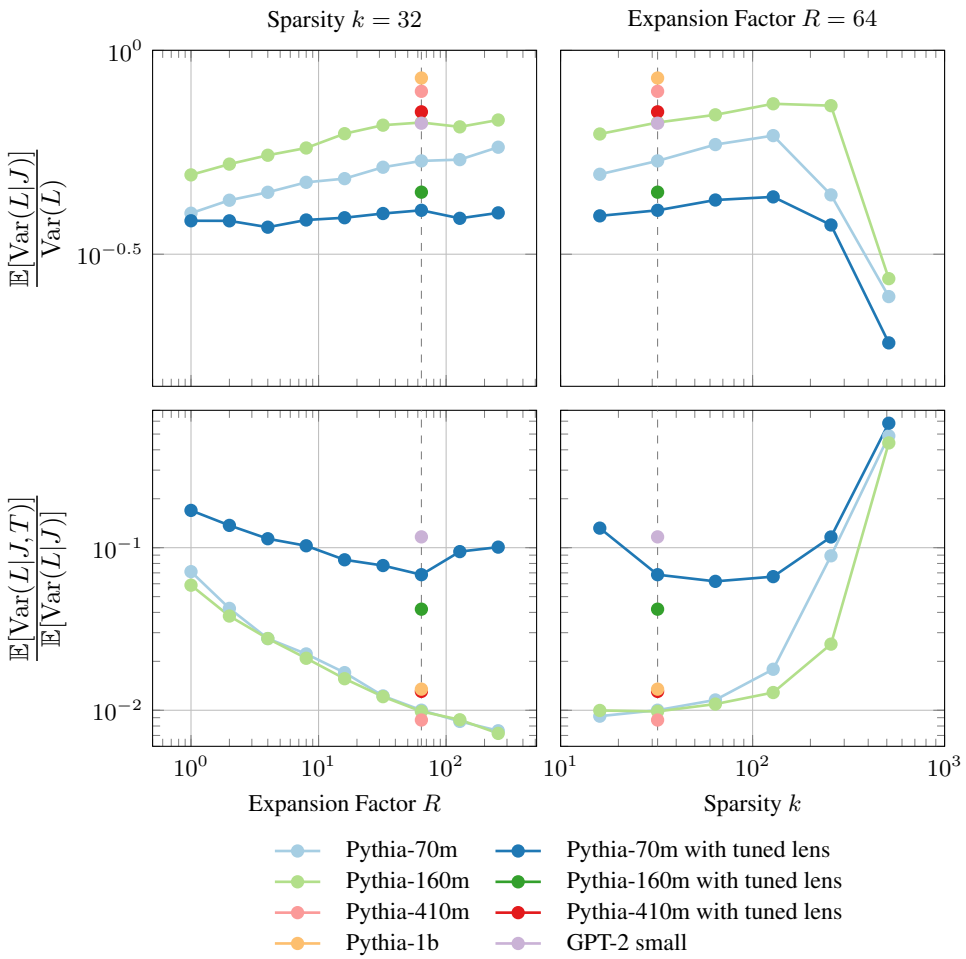


Figure 22: The fraction of the total variance explained by individual latents and the fraction of the variance for an individual latent explained by individual tokens (Eqs. 11 and 12). Here, we plot the variance ratios for standard and tuned-lens MLSAEs over 10 million tokens from the test set.

E Measures of latents active at multiple layers

E.1 Variance of the layer index

Recall that we consider the layer L, token T, and latent index J as random variables (Section 4.3). For a single latent, we have, by the law of total variance:

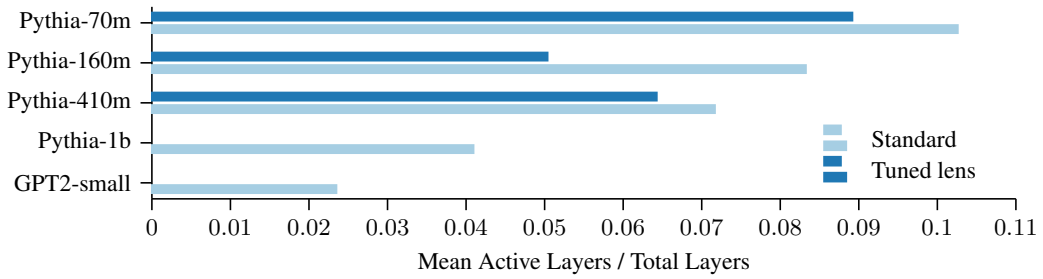
$$\mathrm{Var}\left[L\right] = \mathbb{E}[\mathrm{Var}\left[L \mid T\right]] + \mathrm{Var}\left[\mathbb{E}[L \mid T]\right] \tag{14}$$

We are interested in the first two terms:

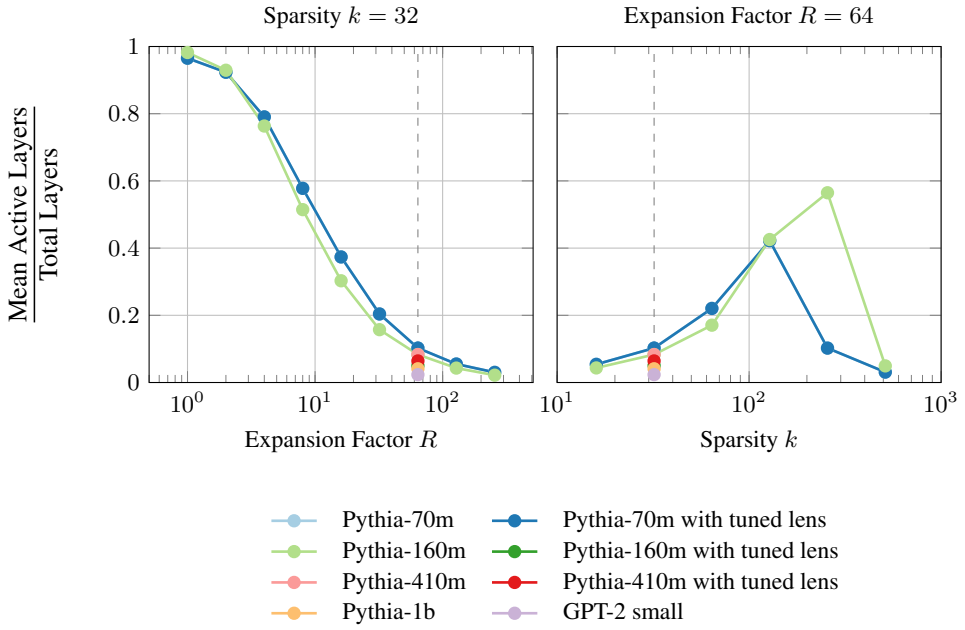
- $\mathrm{Var}\left[L\right]$ is the variance of the distribution over layers, aggregating over tokens;
- $\mathbb{E}[\mathrm{Var}\left[L \mid T\right]]$ is the mean variance of the distributions over layers for each token; and
- $\mathrm{Var}\left[\mathbb{E}[L \mid T]\right]$ is the variance of the mean layers for each token.

Aggregating over latents, we have:

$$\mathbb{E}[\mathrm{Var}\left[L \mid J\right]] = \mathbb{E}[\mathrm{Var}\left[L \mid T, J\right]] + \mathbb{E}[\mathrm{Var}\left[\mathbb{E}[L \mid T, J] \mid J\right]] \tag{15}$$



(a) Varying the model with an expansion factor of $R = 64$ and sparsity $k = 32$



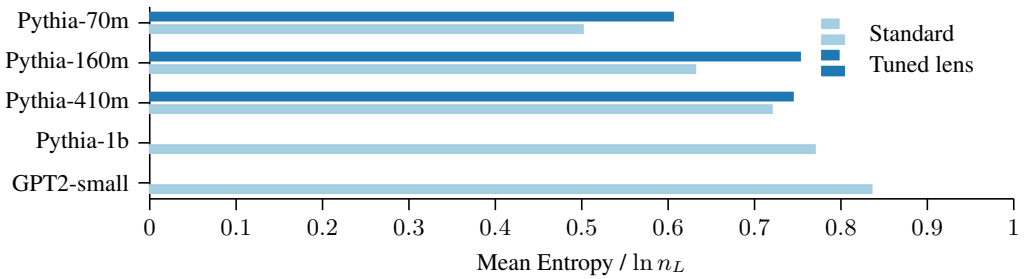
(b) Varying the expansion factor R with sparsity $k = 32$ and k with $R = 64$

Figure 23: The mean number of layers at which latents have a count of non-zero activations above a threshold, divided by the total layers for the model, over 10 million tokens from the test set. The threshold is 10 thousand tokens (0.1%). As in Figure 5, the absence of bars for tuned-lens MLSAEs trained on Pythia-1b and GPT-2 small indicates the absence of results, not that the values are zero.

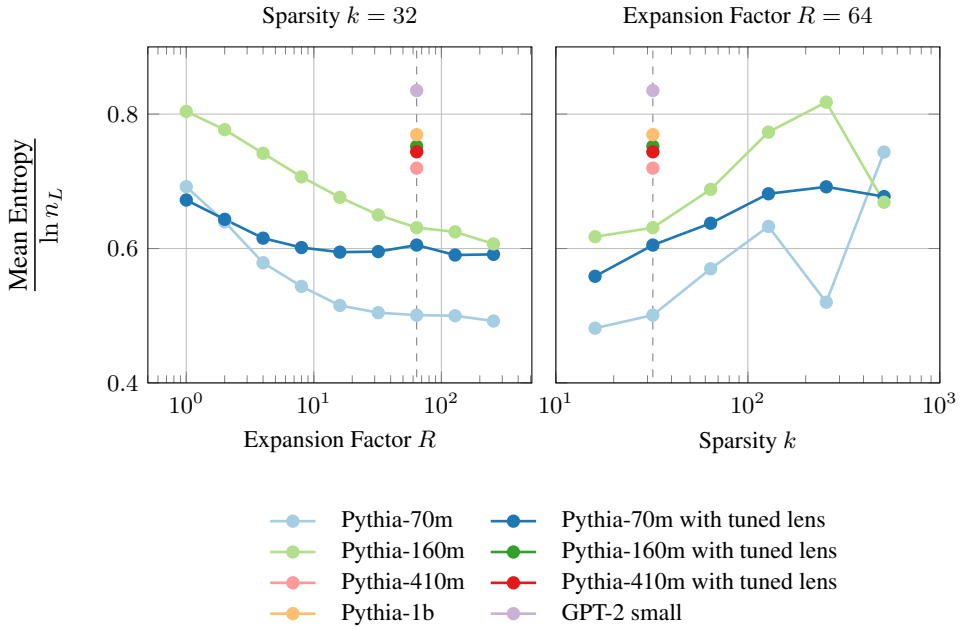
E.2 Number of layers above a threshold

The count of layers at which a latent is active does not necessarily positively correlate with the variance of the layer index considered in Section 4.3. For example, the variance of 0 and 5 (two distinct values) is greater than the variance of 2, 3, and 4 (three distinct values). Strictly speaking, the layer index is ordinal data, but we implicitly treat it as interval data by taking the arithmetic mean and variance. We chose this approach because we expected latents to be active over a contiguous range of layers, which is validated by the normalized heatmaps (e.g., Figures 2 and 3).

For comparison, we computed the number of layers at which each latent has a count of non-zero activations above a threshold (the 'active layers'), divided by the total number of model layers n_L. We selected a threshold count of 10k tokens (0.1% of a sample of 10M tokens). When aggregating over latents, the relative mean active layers decreases as the model size increases for Pythia models (Figure 23a) and as the number of latents increases relative to the model dimension (Figure 23b). Importantly, this measure depends strongly on the choice of threshold.



(a) Varying the model with an expansion factor of $R = 64$ and sparsity $k = 32$



(b) Varying the expansion factor R with sparsity $k = 32$ and k with $R = 64$

Figure 24: The mean entropy of the observed discrete distributions of latent activations over layers (Eq. 10) divided by the maximum entropy of $\ln n_L$, over 10 million tokens from the test set. As in Figure 5, the absence of bars for tuned-lens MLSAEs trained on Pythia-1b and GPT-2 small indicates the absence of results, not that the values are zero.

E.3 Entropy

A further measure of the degree to which a latent is active at multiple layers is the statistical distance between the observed discrete distribution of activations over layers (Eq. 10) and a reference distribution. At one extreme is a Dirac distribution with probability mass 1 for a single layer index and 0 elsewhere, in which case the latent is active at a single layer. The other extreme is the discrete uniform distribution $\mathcal{U}(0, n_L)$, in which case the latent is equally active at every layer. Hence, the entropy of the observed distribution must range between 0 and $\ln n_L$. This measure is agnostic with respect to the numeric values of the layer indices and their order.

We computed the entropy of the observed distributions of activations over layers and took the mean over latents, dividing it by $\ln n_L$ to compare models with different numbers of layers. The normalized mean entropy increases slightly as the model size increases for Pythia models (Figure 24a), like the variance of the layer index (Section 4.3). However, it decreases as the number of latents increases relative to the model dimension, similarly to the mean active layers (Figure 24b).

F ADDITIONAL HEATMAPS

For completeness, we include equivalent aggregate and single-prompt heatmaps to Figures 2 and 3 for different models and combinations of hyperparameters:

- Varying R for Pythia-70m and $k = 32$ (Figures 25 and 26)
- Varying k for Pythia-70m and $R = 64$ (Figures 27 and 28)
- Varying R for Pythia-160m and $k = 32$ (Figures 29 and 30)
- Varying k for Pythia-160m and $R = 64$ (Figures 31 and 32)
- Varying R for Pythia-70m with tuned lens and $k = 32$ (Figures 35 and 35)
- Varying k for Pythia-70m with tuned lens and $R = 64$ (Figures 35 and 36)

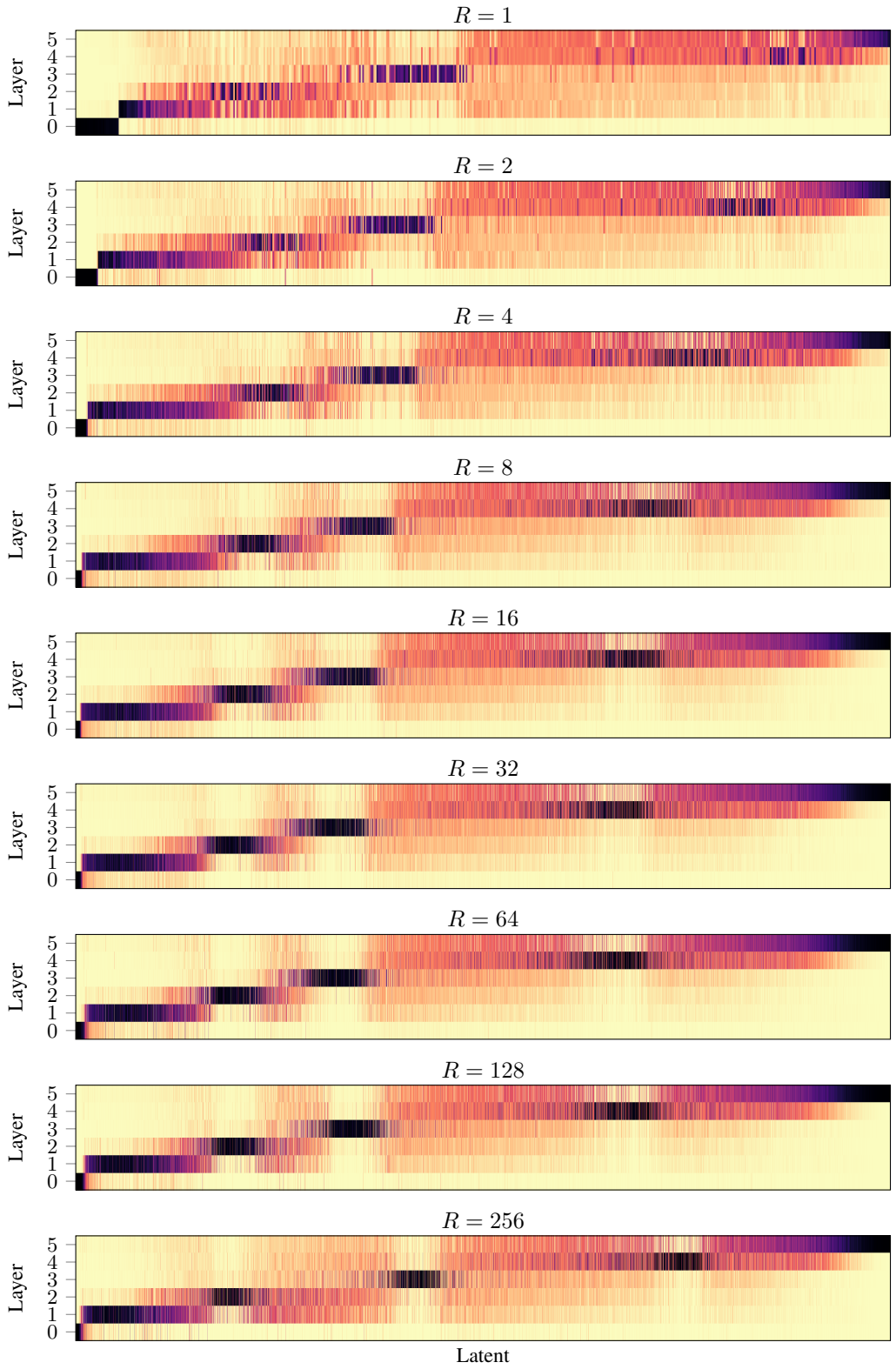


Figure 25: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on Pythia-70m with sparsity $k = 32$. We provide further details in Figure 2.

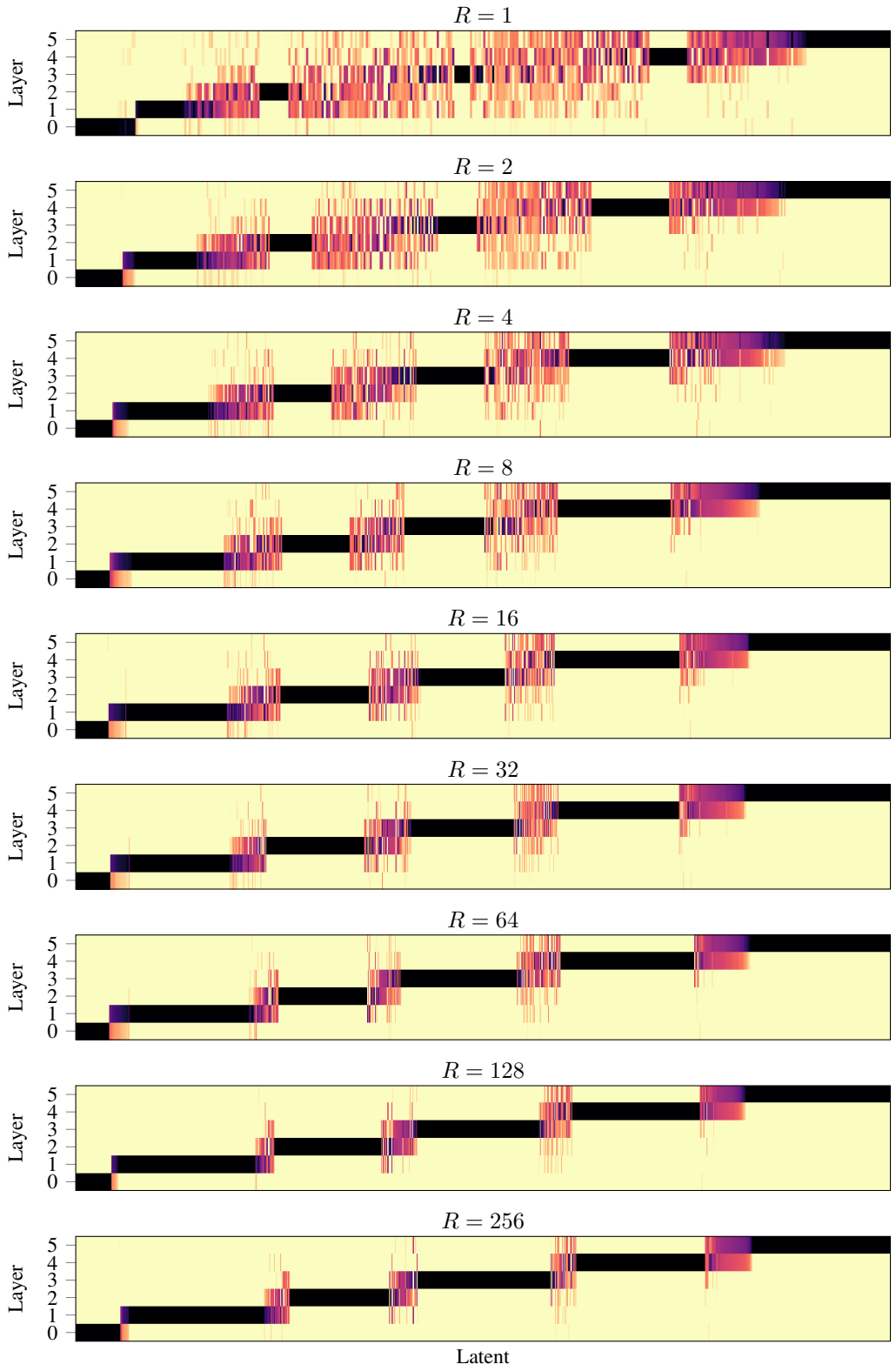


Figure 26: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on Pythia-70m with sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

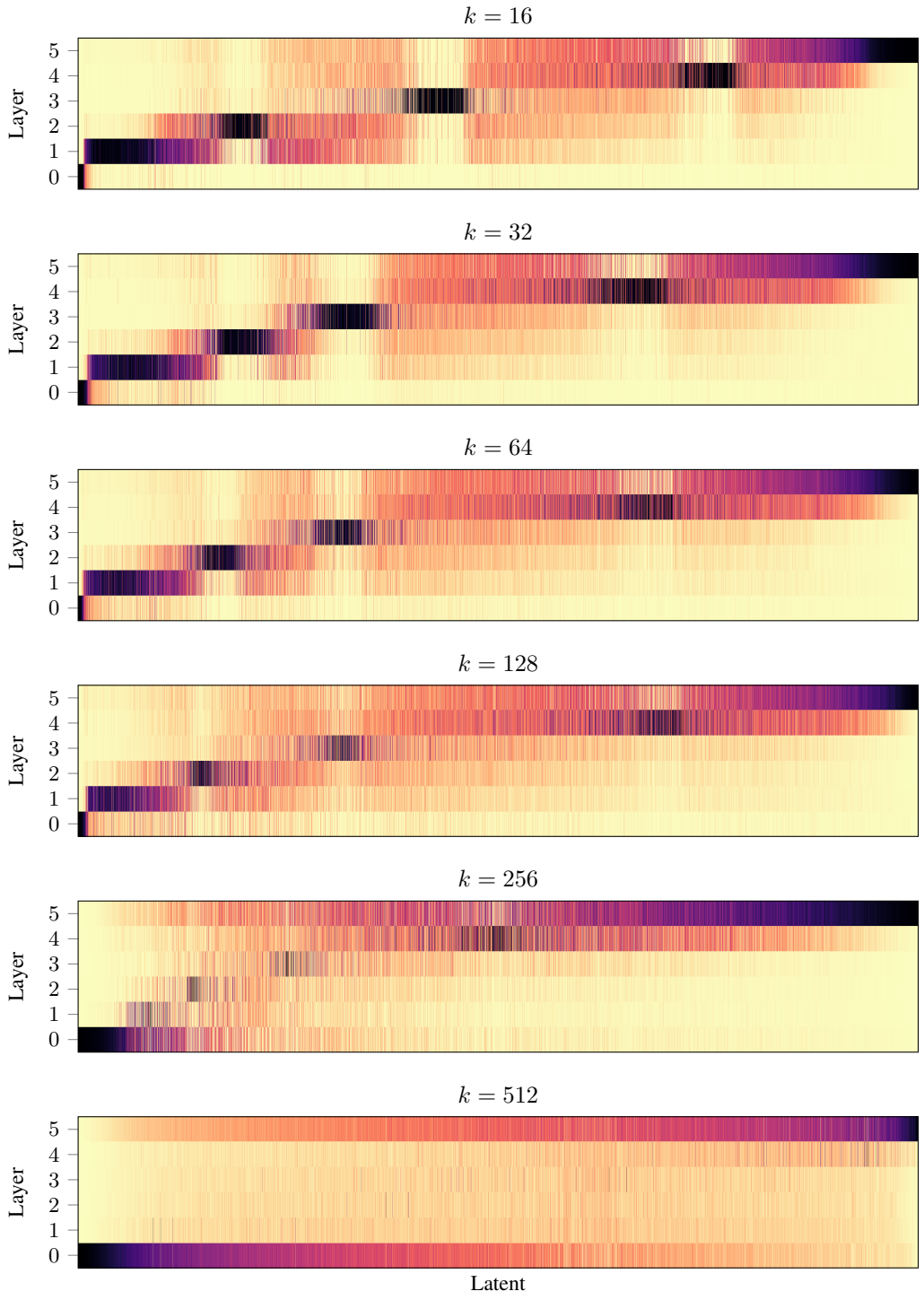


Figure 27: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on Pythia-70m with an expansion factor of $R = 64$. We provide further details in Figure 2.

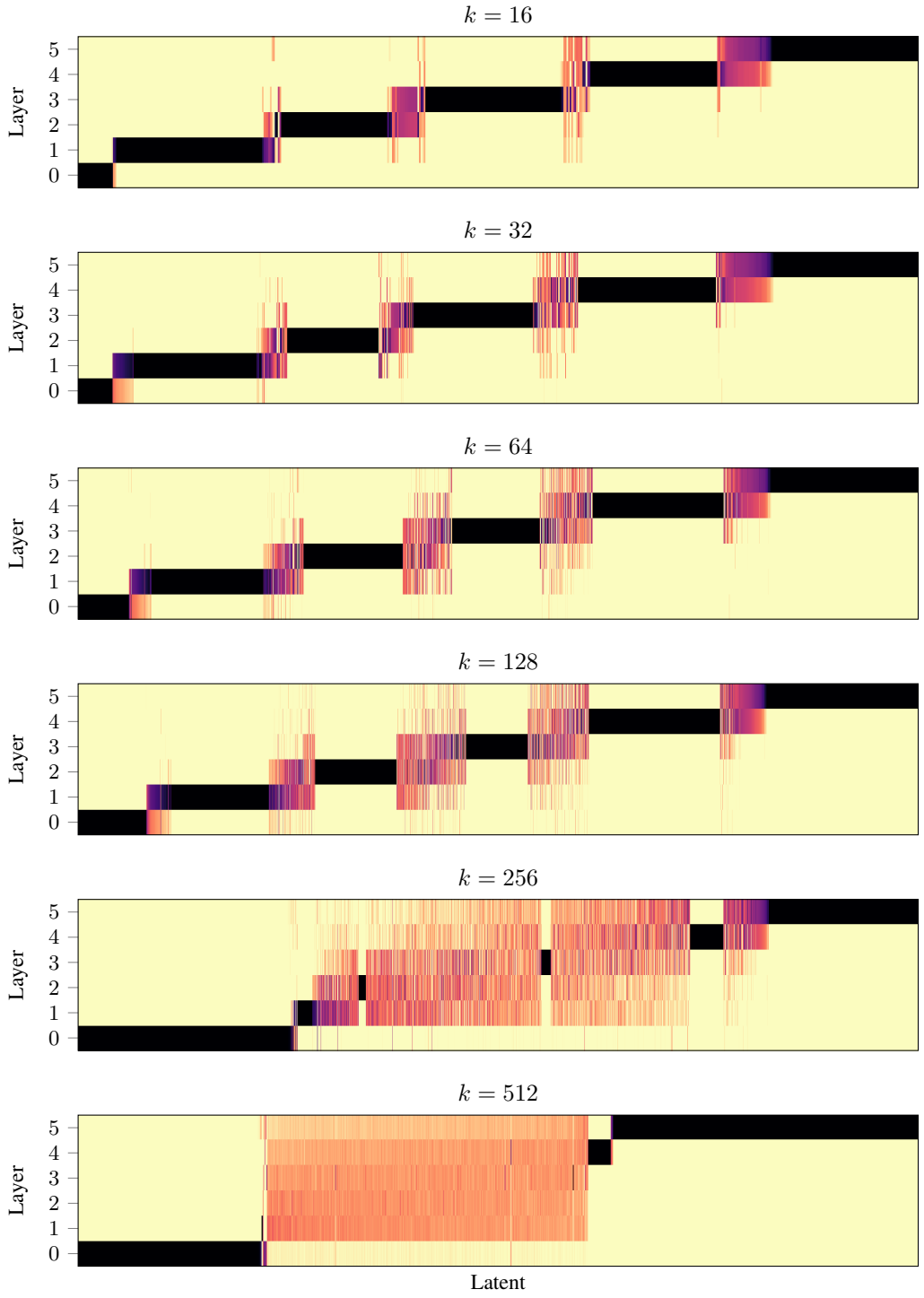


Figure 28: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on Pythia-70m with an expansion factor of $R = 64$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

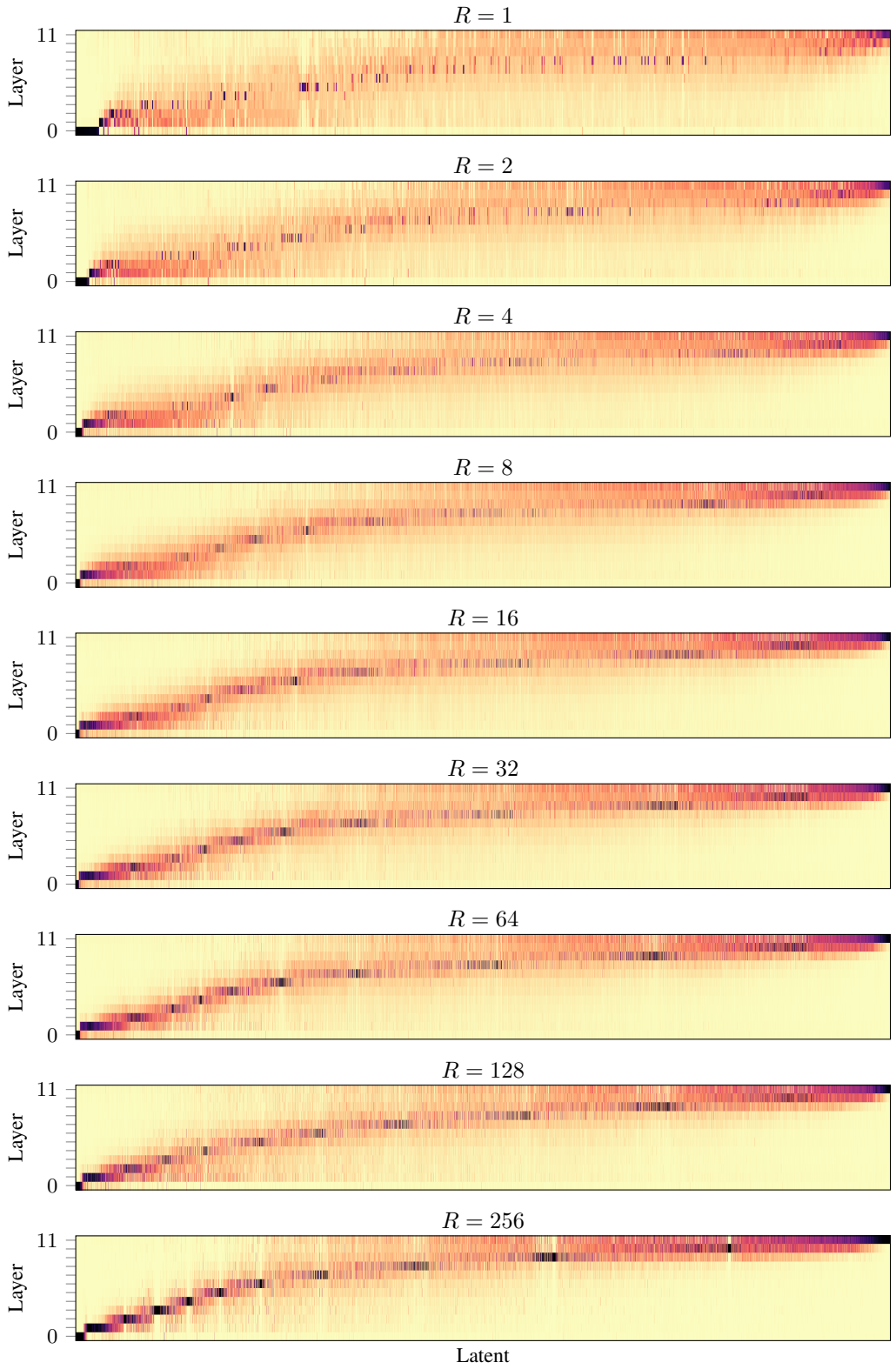


Figure 29: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on Pythia-160m with sparsity $k = 32$. We provide further details in Figure 2.

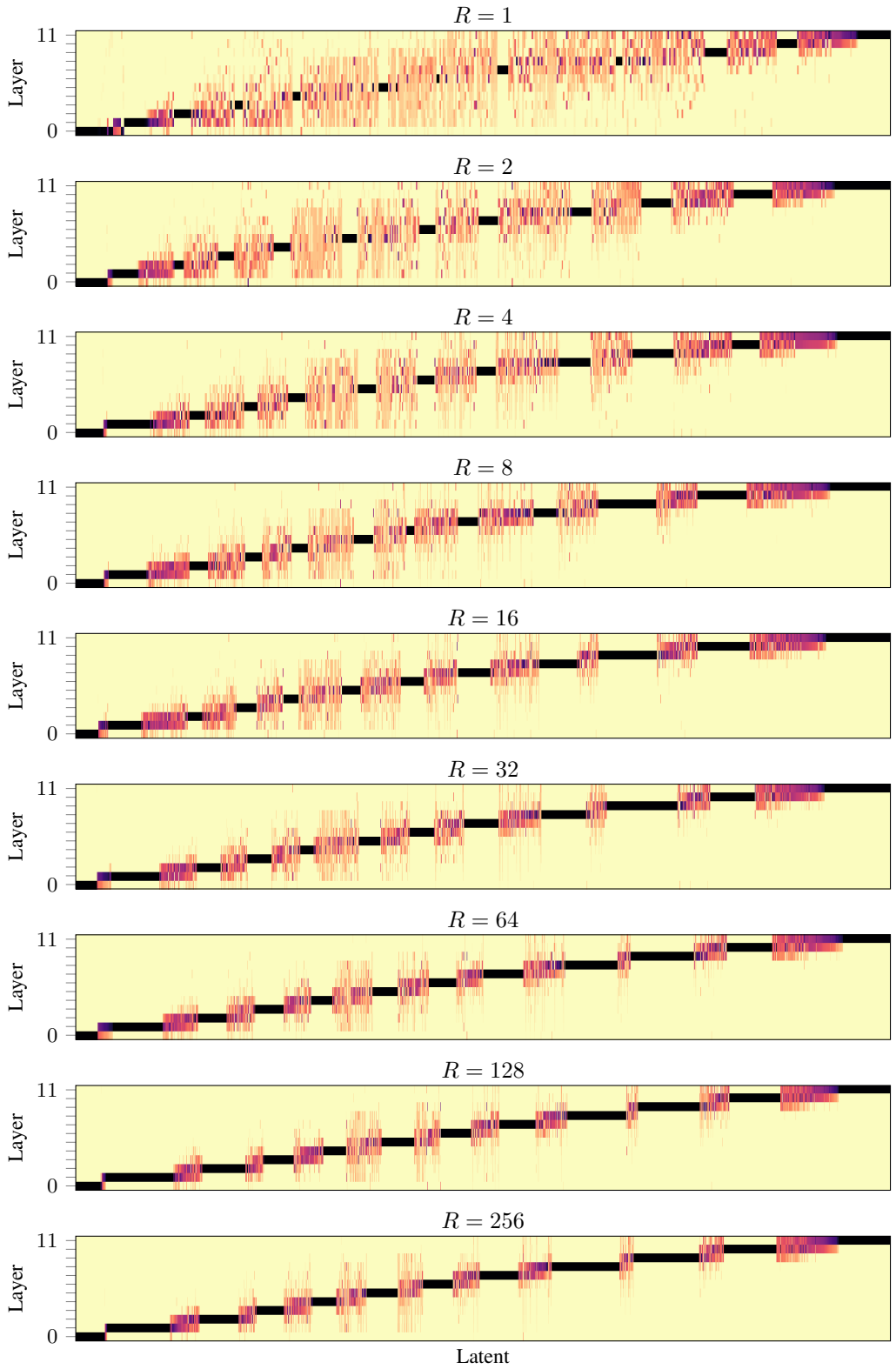


Figure 30: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on Pythia-160m with sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

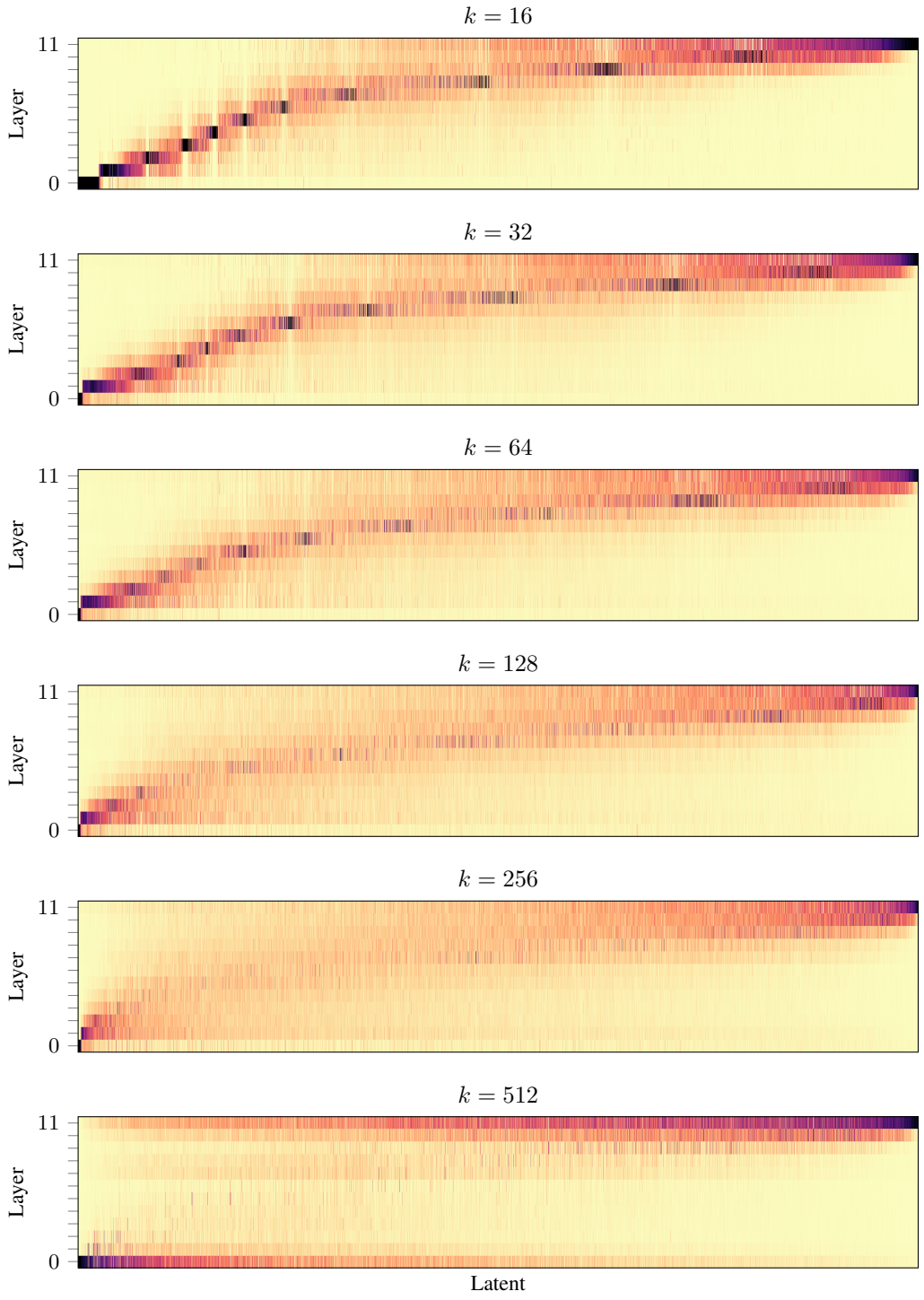


Figure 31: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for MLSAEs trained on Pythia-160m with an expansion factor of $R = 64$. We provide further details in Figure 2.

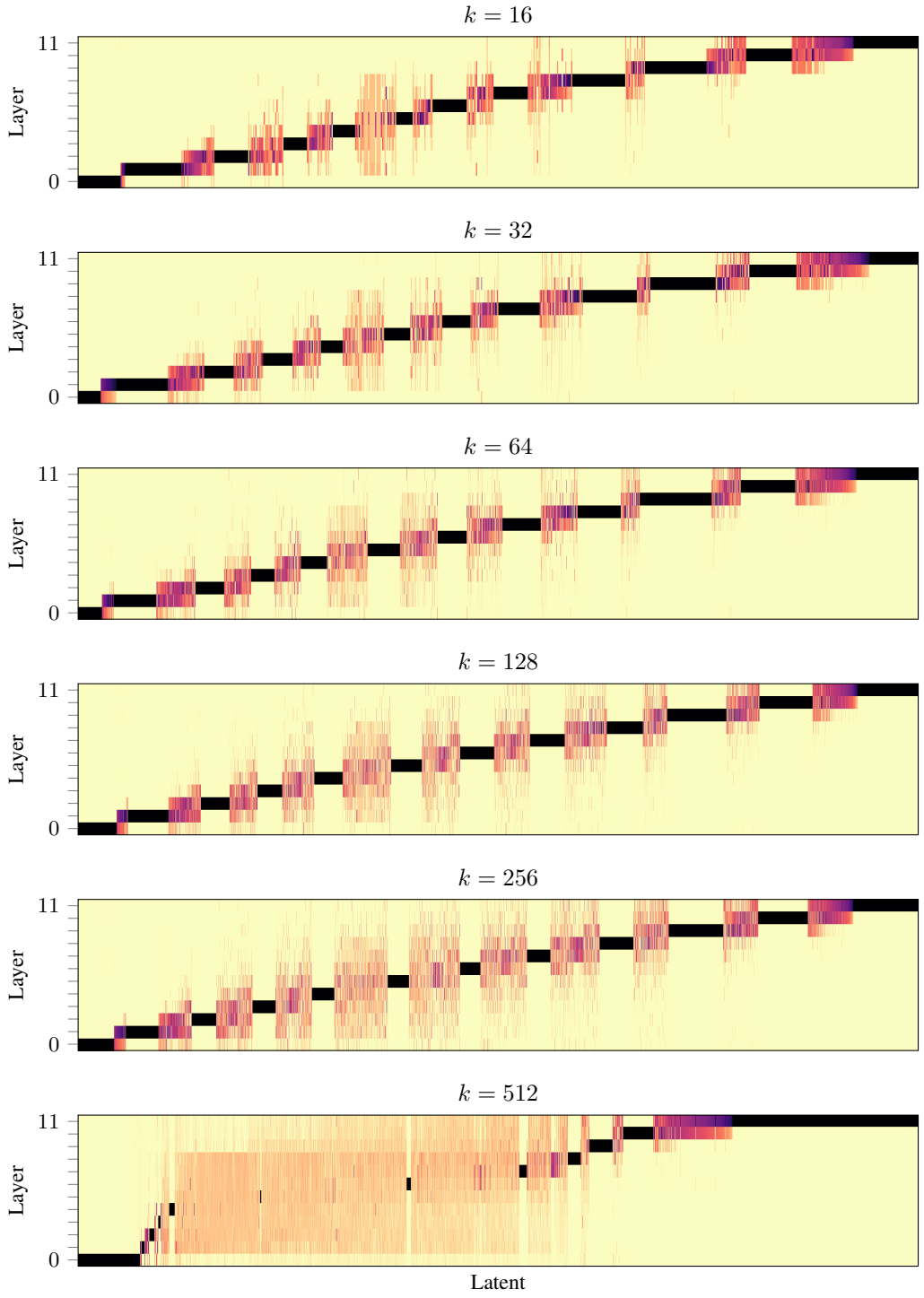


Figure 32: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for MLSAEs trained on Pythia-160m with an expansion factor of $R = 64$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

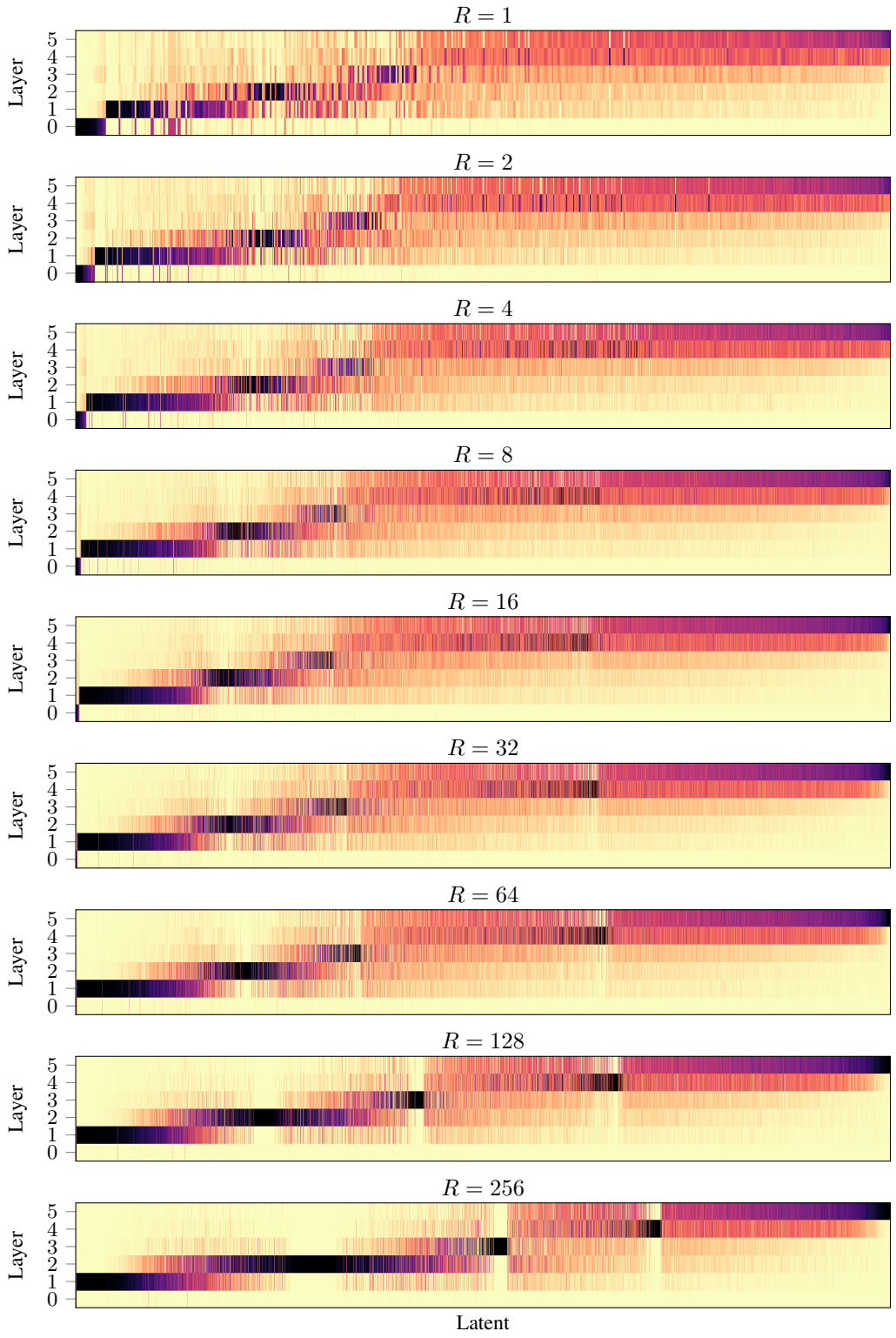


Figure 33: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia-70m with sparsity $k = 32$. We provide further details in Figure 2.

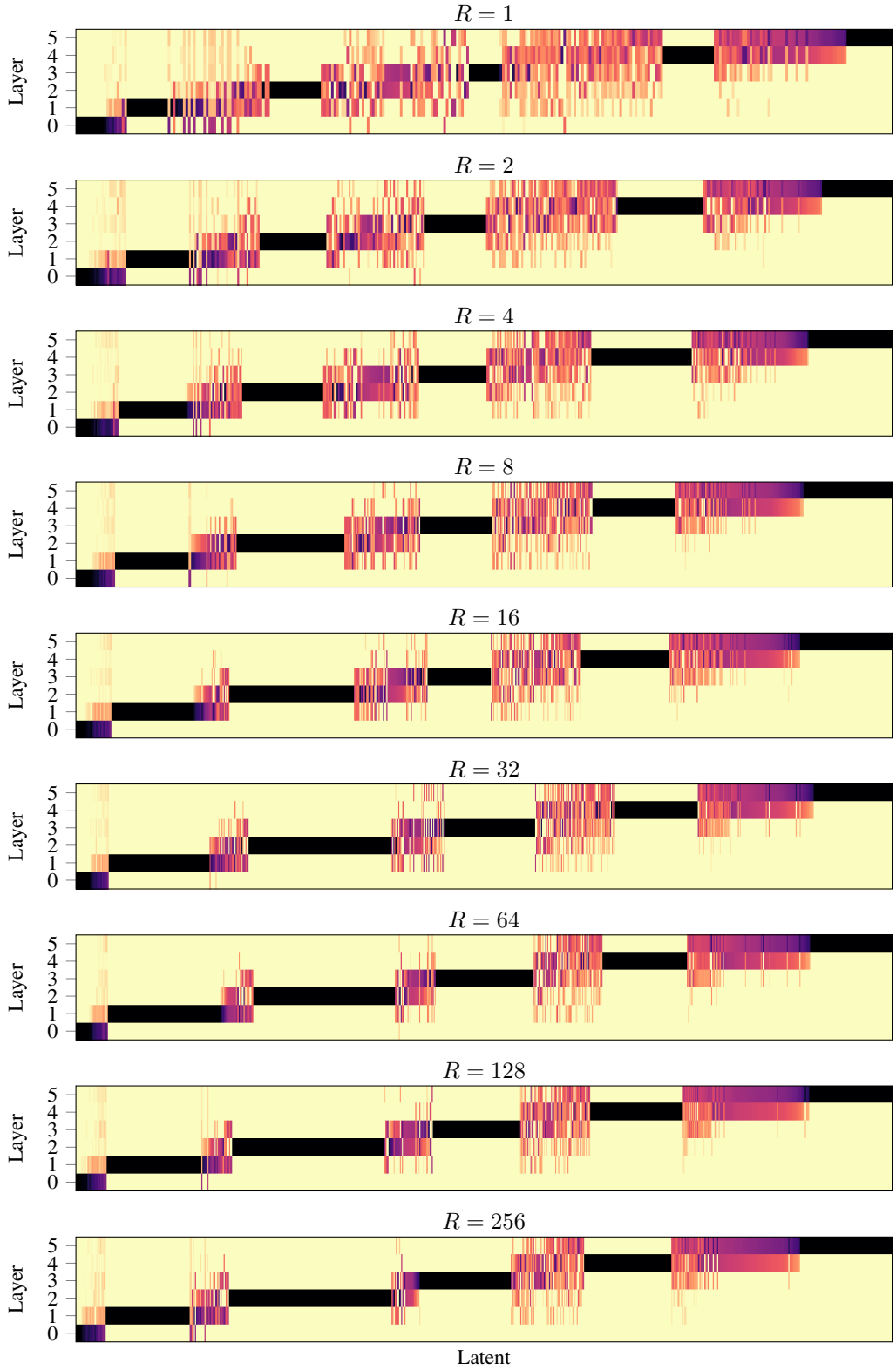


Figure 34: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia-70m with sparsity $k = 32$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.

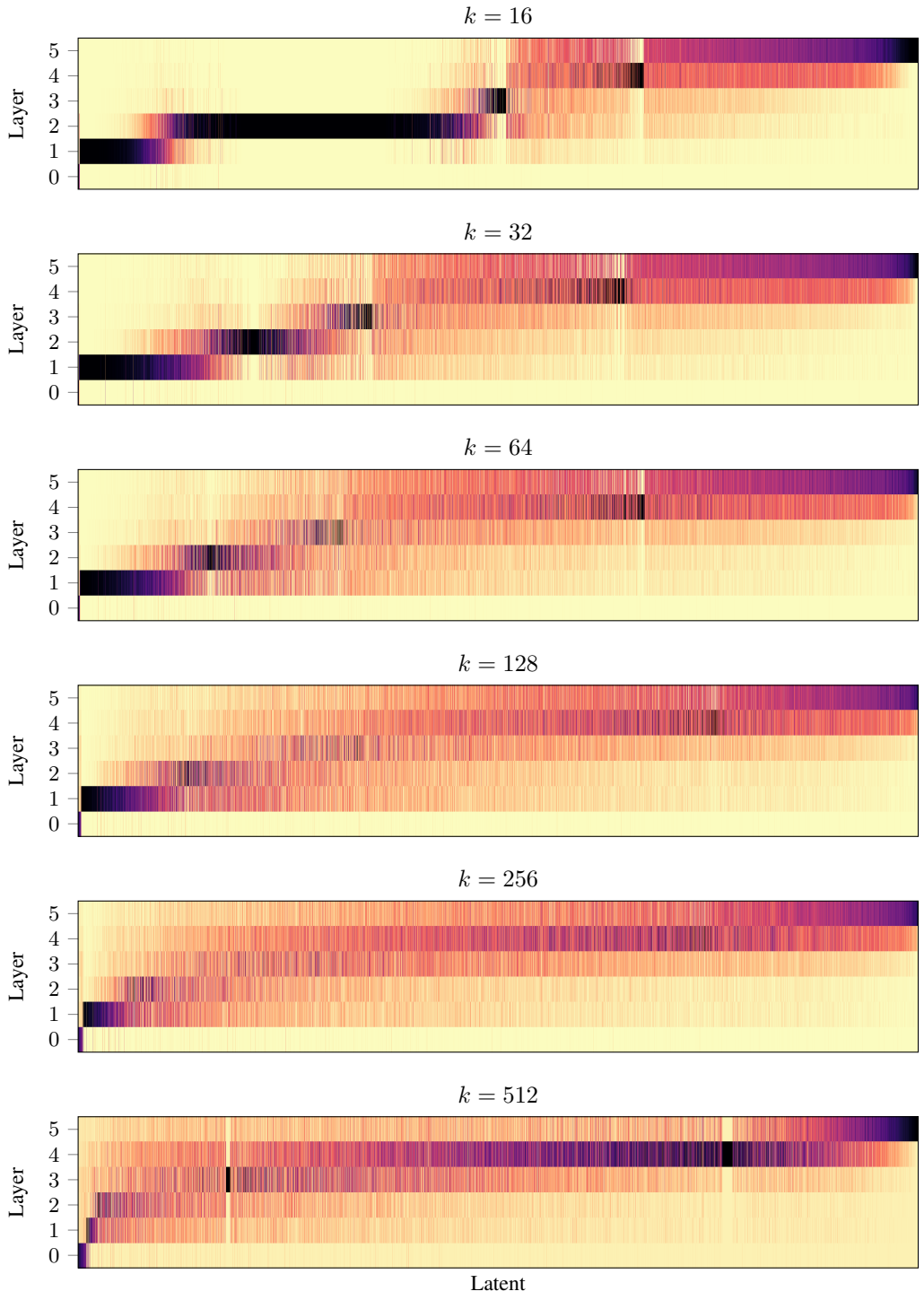


Figure 35: Heatmaps of the distributions of latent activations over layers when aggregating over 10 million tokens from the test set. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia-70m with an expansion factor of $R = 64$. We provide further details in Figure 2.

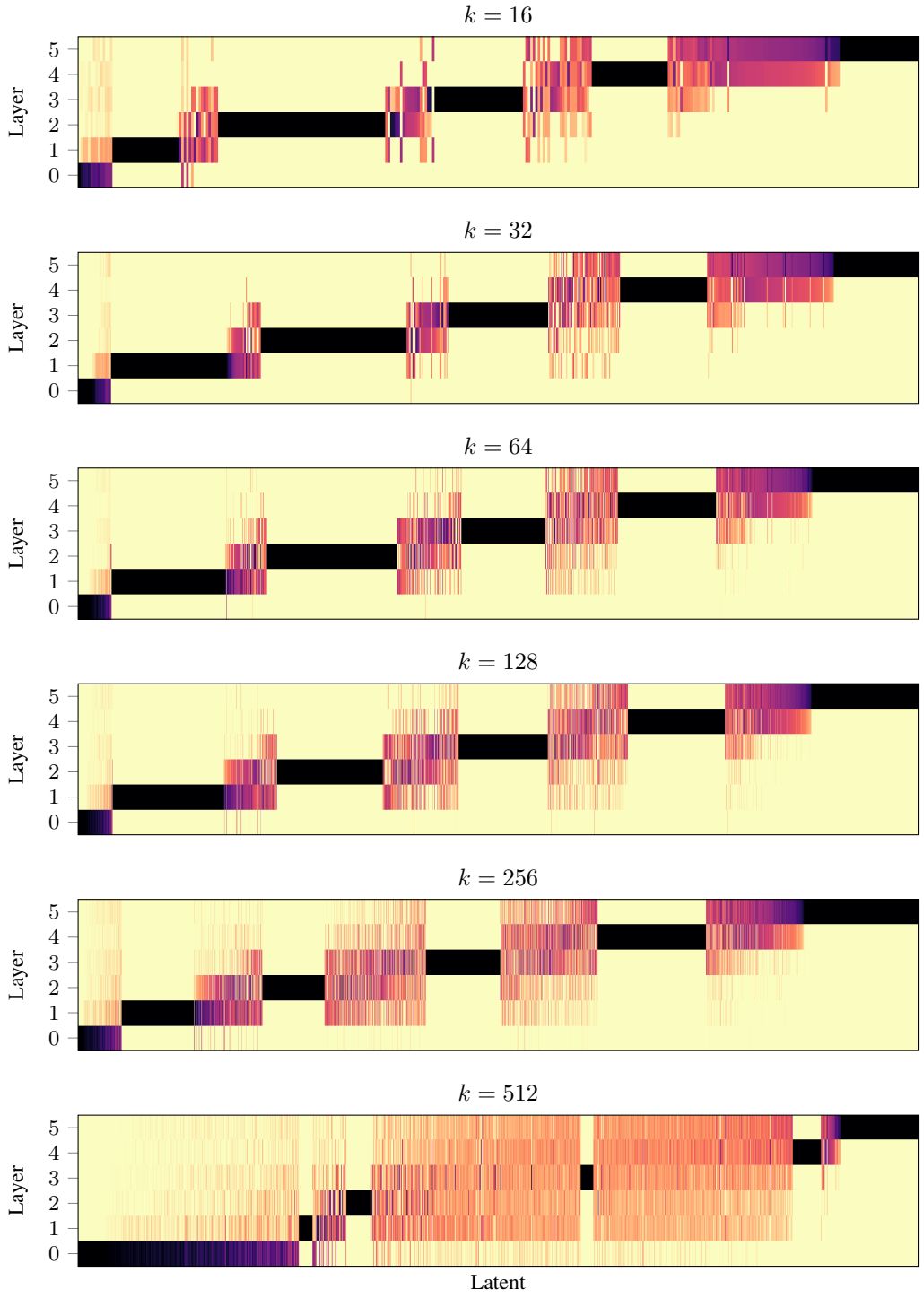


Figure 36: Heatmaps of the distributions of latent activations over layers for a single example prompt. Here, we plot the distributions for tuned-lens MLSAEs trained on Pythia-70m with an expansion factor of $R = 64$. The example prompt is "When John and Mary went to the store, John gave" (Wang et al., 2022). We provide further details in Figure 3.