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 a sparse dictionary 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 primarily on GPT-style language models from the Pythia suite (Biderman et al., 2023); see Appendix B.3 for others. We are 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, $\mathrm{Var}$ 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 $k_{\text{aux}}$ 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-$k_{\text{aux}}$ dead latents. Let $\mathrm{Dead}$ 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 $\alpha$:

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

Following Gao et al. (2024), we choose $k_{\text{aux}}$ 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 equal to training $n_L$ single-layer SAEs on the same number of tokens. We trained most MLSAEs on a single NVIDIA GeForce RTX 3090 GPU for between 12 and 24 hours; we trained the largest MLSAEs (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 $\ell$ to the output space of the final layer, called the translator for layer $\ell$ (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:

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

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:

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

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

| Model | FVU | $L^1$ Norm | Loss ↑ |
|---|---|---|---|
| Pythia-70m | 0.097 | 66.5 | 0.565 |
| Pythia-160m | 0.106 | 76.1 | 0.432 |
| Pythia-410m | 0.081 | 84.6 | 0.414 |
| Pythia-1b | 0.095 | 109.7 | 0.404 |
| Pythia-1.4b | 0.106 | 124.7 | 0.487 |
| Gemma 2 2B | 0.210 | 255.5 | 1.483 |
| Llama 3.2 3B | 0.299 | 141.4 | 0.347 |
| GPT-2 small | 0.093 | 196.9 | 0.759 |

(a) Standard

| Model | FVU | $L^1$ Norm | Loss ↑ |
|---|---|---|---|
| Pythia-70m | 0.030 | 61.5 | 0.274 |
| Pythia-160m | 0.088 | 89.6 | −0.080 |
| Pythia-410m | 0.073 | 80.5 | 0.827 |

(b) With tuned lens

Table 1: The mean FVU, $L^1$ norm, and increase in the cross-entropy loss (Loss ↑) for MLSAEs with an expansion factor of $R = 64$ and sparsity $k = 32$, over 1 million tokens from the test set. We provide details of the evaluation metrics and transformer architectures in Appendix B.

In Eq. 9, $\hat{\mathbf{x}}'$ is the transformed output vector of the decoder, $\hat{\mathbf{x}}$ is the output vector, $\mathbf{W}_{\text{lens},\ell} \in \mathbb{R}^{d \times d}$, and $\mathbf{b}_{\text{lens},\ell} \in \mathbb{R}^d$. 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), and we compute the inverse $(\mathbf{I} + \mathbf{W}_{\text{lens},\ell})^{-1}$ for each layer once at the start of training. The pre-trained tuned lenses used were provided by the authors of Belrose et al. (2023), which 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 term 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 measured 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 with our default hyperparameters. The FVU and delta cross-entropy (CE) loss remain consistent across model sizes for Pythia transformers. In most cases, applying tuned-lens transformations decreases the FVU and delta CE loss (see Section 4.4 and Figure 12). We provide results for other hyperparameters, breakdowns by the layer of the input activation vectors, and explicit comparisons to single-layer SAEs 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). 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 the mean cosine similarities increased as expected (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).

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