Scaling and evaluating sparse autoencoders

Leo Gao* **Tom Dupré la Tour†** **Henk Tillman†**

Gabriel Goh **Rajan Troll** **Alec Radford**

Ilya Sutskever **Jan Leike** **Jeffrey Wu†**

OpenAI

Abstract

Sparse autoencoders provide a promising unsupervised approach for extracting interpretable features from a language model by reconstructing activations from a sparse bottleneck layer. Since language models learn many concepts, autoencoders need to be very large to recover all relevant features. However, studying the properties of autoencoder scaling is difficult due to the need to balance reconstruction and sparsity objectives and the presence of dead latents. We propose using k-sparse autoencoders [Makhzani and Frey, 2013] to directly control sparsity, simplifying tuning and improving the reconstruction-sparsity frontier. Additionally, we find modifications that result in few dead latents, even at the largest scales we tried. Using these techniques, we find clean scaling laws with respect to autoencoder size and sparsity. We also introduce several new metrics for evaluating feature quality based on the recovery of hypothesized features, the explainability of activation patterns, and the sparsity of downstream effects. These metrics all generally improve with autoencoder size. To demonstrate the scalability of our approach, we train a 16 million latent autoencoder on GPT-4 activations for 40 billion tokens. We release code and autoencoders for open-source models, as well as a visualizer.[3]

1 Introduction

Sparse autoencoders (SAEs) have shown great promise for finding features [Cunningham et al., 2023, Bricken et al., 2023, Templeton et al., 2024, Goh, 2016] and circuits [Marks et al., 2024] in language models. Unfortunately, they are difficult to train due to their extreme sparsity, so prior work has primarily focused on training relatively small sparse autoencoders on small language models.

We develop a state-of-the-art methodology to reliably train extremely wide and sparse autoencoders with very few dead latents on the activations of any language model. We systematically study the scaling laws with respect to sparsity, autoencoder size, and language model size. To demonstrate that our methodology can scale reliably, we train a 16 million latent autoencoder on GPT-4 [OpenAI, 2023] residual stream activations.

Because improving reconstruction and sparsity is not the ultimate objective of sparse autoencoders, we also explore better methods for quantifying autoencoder quality. We study quantities corresponding

*Primary Contributor. Correspondence to lg@openai.com.

†Core Research Contributor. This project was conducted by the Superalignment Interpretability team. Author contributions statement in Appendix I.

[3]Our open source code can be found at https://github.com/openai/sparse_autoencoder and our visualizer is hosted at https://openaipublic.blob.core.windows.net/sparse-autoencoder/sae-viewer/index.html

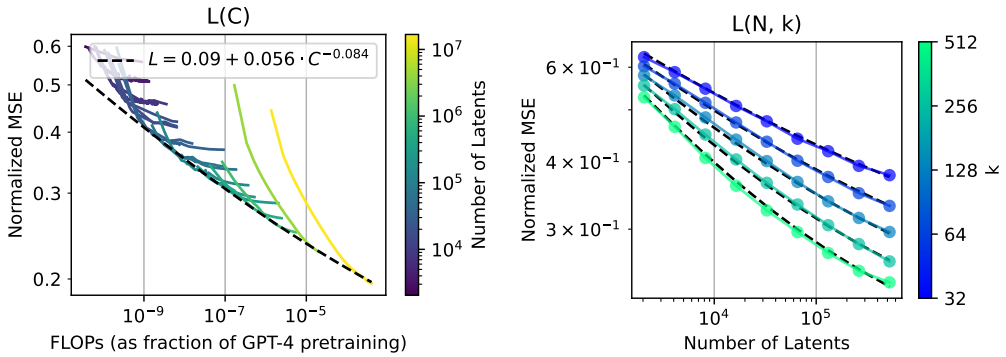


Figure 1: Scaling laws for TopK autoencoders trained on GPT-4 activations. (Left) Optimal loss for a fixed compute budget. (Right) Joint scaling law of loss at convergence with fixed number of total latents n and fixed sparsity (number of active latents) k. Details in Section 3.

to: whether certain hypothesized features were recovered, whether downstream effects are sparse, and whether features can be explained with both high precision and recall.

Our contributions:

1. In Section 2, we describe a state-of-the-art recipe for training sparse autoencoders.
2. In Section 3, we demonstrate clean scaling laws and scale to large numbers of latents.
3. In Section 4, we introduce metrics of latent quality and find larger sparse autoencoders are generally better according to these metrics.

We also release code, a full suite of GPT-2 small autoencoders, and a feature visualizer for GPT-2 small autoencoders and the 16 million latent GPT-4 autoencoder.

2 Methods

2.1 Setup

Inputs: We train autoencoders on the residual streams of both GPT-2 small [Radford et al., 2019] and models from a series of increasing sized models sharing GPT-4 architecture and training setup, including GPT-4 itself [OpenAI, 2023][4]. We choose a layer near the end of the network, which should contain many features without being specialized for next-token predictions (see Section F.1 for more discussion). Specifically, we use a layer $\frac{5}{6}$ of the way into the network for GPT-4 series models, and we use layer 8 ($\frac{3}{4}$ of the way) for GPT-2 small. We use a context length of 64 tokens for all experiments. We subtract the mean over the d_{model} dimension and normalize to all inputs to unit norm, prior to passing to the autoencoder (or computing reconstruction errors).

Evaluation: After training, we evaluate autoencoders on sparsity L_0, and reconstruction mean-squared error (MSE). We report a normalized version of all MSE numbers, where we divide by a baseline reconstruction error of always predicting the mean activations.

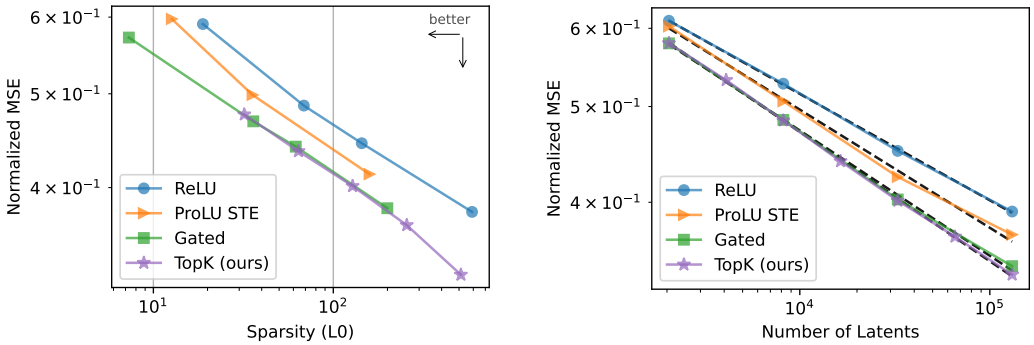
Hyperparameters: To simplify analysis, we do not consider learning rate warmup or decay unless otherwise noted. We sweep learning rates at small scales and extrapolate the trend of optimal learning rates for large scale. See Appendix A for other optimization details.

2.2 Baseline: ReLU autoencoders

For an input vector $x \in \mathbb{R}^d$ from the residual stream, and n latent dimensions, we use baseline ReLU autoencoders from [Bricken et al., 2023]. The encoder and decoder are defined by:

$$\begin{aligned} z &= \text{ReLU}(W_{\text{enc}}(x - b_{\text{pre}}) + b_{\text{enc}}) \\ \hat{x} &= W_{\text{dec}} z + b_{\text{pre}} \end{aligned} \tag{1}$$

[4]All presented results either have qualitatively similar results on both GPT-2 small and GPT-4 models, or were only ran on one model class.



(a) At a fixed number of latents ($n = 32768$), TopK has a better reconstruction-sparsity trade off than ReLU and ProLU, and is comparable to Gated.

(b) At a fixed sparsity level ($L_0 = 128$), scaling laws are steeper for TopK than ReLU.[6]

Figure 2: Comparison between TopK and other activation functions.

with $W_{\text{enc}} \in \mathbb{R}^{n \times d}$, $b_{\text{enc}} \in \mathbb{R}^n$, $W_{\text{dec}} \in \mathbb{R}^{d \times n}$, and $b_{\text{pre}} \in \mathbb{R}^d$. The training loss is defined by $\mathcal{L} = ||x - \hat{x}||_2^2 + \lambda ||z||_1$, where $||x - \hat{x}||_2^2$ is the reconstruction MSE, $||z||_1$ is an L_1 penalty promoting sparsity in latent activations z, and λ is a hyperparameter that needs to be tuned.

2.3 TopK activation function

We use a k-sparse autoencoder [Makhzani and Frey, 2013], which directly controls the number of active latents by using an activation function (TopK) that only keeps the k largest latents, zeroing the rest. The encoder is thus defined as:

$$z = \text{TopK}(W_{\text{enc}}(x - b_{\text{pre}})) \tag{2}$$

and the decoder is unchanged. The training loss is simply $\mathcal{L} = ||x - \hat{x}||_2^2$.

Using k-sparse autoencoders has a number of benefits:

- It removes the need for the L_1 penalty. L_1 is an imperfect approximation of L_0, and it introduces a bias of shrinking all positive activations toward zero (Section 5.1).
- It enables setting the L_0 directly, as opposed to tuning an L_1 coefficient λ, enabling simpler model comparison and rapid iteration. It can also be used in combination with arbitrary activation functions.[5]
- It empirically outperforms baseline ReLU autoencoders on the sparsity-reconstruction frontier (Figure 2a), and this gap increases with scale (Figure 2b).
- It increases monosemanticity of random activating examples by effectively clamping small activations to zero (Section 4.3).

2.4 Preventing dead latents

Dead latents pose another significant difficulty in autoencoder training. In larger autoencoders, an increasingly large proportion of latents stop activating entirely at some point in training. For example, Templeton et al. [2024] train a 34 million latent autoencoder with only 12 million alive latents, and in our ablations we find up to 90% dead latents[7] when no mitigations are applied (Figure 15). This results in substantially worse MSE and makes training computationally wasteful. We find two important ingredients for preventing dead latents: we initialize the encoder to the transpose of the decoder, and we use an auxiliary loss that models reconstruction error using the top-k_{aux} dead latents (see Section A.2 for more details). Using these techniques, even in our largest (16 million latent) autoencoder only 7% of latents are dead.

[5] In our code, we also apply a ReLU to guarantee activations to be non-negative. However, the training curves are indistinguishable, as the k largest activations are almost always positive for reasonable choices of k.

[6] Non-TopK cannot set a precise L_0, so we interpolated using a piecewise linear function in log-log space.

[7] We follow Templeton et al. [2024], and consider a latent dead if it has not activated in 10 million tokens