Gemma Scope 2 - Technical Paper

Callum McDougall[1], **Arthur Conmy**[1], **János Kramár**[1], **Tom Lieberum**[1], **Senthooran Rajamanoharan**[1] **and Neel Nanda**[1]
[1]Google

In response to a surge of recent work using SAEs to study model biology and to analyze circuits that explain complex, multi-step behaviors, we train and release an open suite of JumpReLU sparse autoencoders (SAEs) and skip-transcoders on all layers and sub-layers of Gemma 3 models at 270M, 1B, 4B, 12B, and 27B, as well as a set of multi-layer models to enable circuit-level analyses that span across layers. In this way, we hope to not only enable interpretability research on the Gemma 3 model series (more advanced than the previous Gemma 2 series) but also to enable analysis of multi-layer representations and circuits, which allows the study of more complex and potentially harmful behaviors. We are encouraged by the quality of open-source research enabled by our prior release, and aim to further accelerate this work by releasing updated weights, evaluations, and tooling.

Keywords: gemma scope, sparse autoencoders, transcoders

1. Introduction

A growing body of work suggests many internal activations of language models can be well-approximated by sparse, linear combinations of dictionary vectors ((Elhage et al., 2022); (Gurnee et al., 2023); (Mikolov et al., 2013); (Nanda et al., 2023a); (Olah et al., 2020); (Park et al., 2023)). Sparse autoencoders (SAEs) provide an unsupervised route to discover such directions and have repeatedly yielded causally relevant, interpretable latents ((Bricken et al., 2023); (Cunningham et al., 2023); (Gao et al., 2024); (Marks et al., 2024); (Templeton et al., 2024)). Realizing this promise requires maturing the methodology, validating reliability, and scaling training and evaluation to modern models, so SAEs can support applications like detecting hallucinations, debugging unexpected behaviors, and increasing reliability and safety ((Hubinger, 2022); (Nanda, 2022); (Olah, 2021)).

Despite rapid progress, training comprehensive, high-quality SAE suites remains costly compared to techniques such as steering vectors ((Li et al., 2023); (Turner et al., 2024)) or probing (Belinkov, 2022). Much prior work has focused on single-layer settings ((Engels et al., 2024); (Gao et al., 2024); (Templeton et al., 2024)), leaving open how best to scale to multilayer analyses and circuit-style work.

Recent work from Anthropic on cross-layer transcoders (CLTs) highlights the value of modeling interactions among latents across layers, rather than treating latents as isolated, single-layer objects. Cross-layer approaches can synthesize information flowing through multiple transformer blocks, enabling new forms of understanding and control for complex behaviors such as jailbreaks and unfaithful chain-of-thought reasoning (Lindsey et al., 2025). Together with transcoders (Dunefsky et al., 2024) and multi-layer SAE models, this points toward circuit-level analyses that capture multi-step computations spanning several layers and modules.

To better enable this kind of analysis, we have trained and released the weights of Gemma Scope 2: an open suite of models which builds on our previous Gemma Scope release (Lieberum et al., 2024). This new release includes SAEs and transcoders for every layer and sublayer of Gemma 3 270M, 1B, 4B, 12B, and 27B. We release these weights under a permissive CC-BY-4.0 license on HuggingFace to enable and accelerate research by other members of the research community.

Engineering challenges for this work were greater than our previous Gemma Scope release,

owing to not only the greater scope of single-layer models in the release, but also the added difficulty of training and evaluating multi-layer models. Increasing the number of SAE layers directly impacts compute and memory: input batch sizes scale as $O(\text{layers})$, naive FLOPs scale as $O(\text{layers}^2)$ (since the decoder is dense over every pair of layers), and parameter counts also scale as $O(\text{layers}^2)$. We mitigate the computational overhead by employing a variant of Leo Gao's sparse kernels so that effective FLOPs scale approximately linearly, $O(\text{layers})$, and we address the parameter scaling via extreme model sharding—splitting decoder weights across many devices—while using minimal data sharding to avoid costly all-reduces. This setup allows comprehensive multi-layer training and evaluation while maintaining practical throughput and stability.

In Section 2 we provide background on SAEs and transcoders, covering the context relevant for this updated release. Section 3 contains details of our training procedure, hyperparameters and computational infrastructure. We run extensive evaluations on the trained SAEs in Section 4 and a list of open problems that Gemma Scope 2 could help tackle in Section 5.

2. Preliminaries

2.1. Sparse autoencoders

Given activations $\mathbf{x} \in \mathbb{R}^n$ from a language model, a sparse autoencoder (SAE) decomposes and reconstructs the activations using a pair of encoder and decoder functions $(\mathbf{f}, \hat{\mathbf{x}})$ defined by:

$$\mathbf{f}(\mathbf{x}) := \sigma\left(\mathbf{W}_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}\right) \tag{1}$$

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

These functions are trained to map $\hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))$ back to $\mathbf{x}$, making them an autoencoder. Thus, $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^M$ is a set of linear weights that specify how to combine the $M \gg n$ columns of $\mathbf{W}_{\text{dec}}$ to reproduce $\mathbf{x}$. The columns of $\mathbf{W}_{\text{dec}}$, which we denote by $\mathbf{d}_i$ for $i = 1 \ldots M$, represent the dictionary of directions into which the SAE decomposes $\mathbf{x}$. We will refer to these learned directions as latents to disambiguate between learned 'features' and the conceptual features which are hypothesized to comprise the language model's representation vectors.

The decomposition $\mathbf{f}(\mathbf{x})$ is made non-negative and sparse through the choice of activation function σ and appropriate regularization, such that $\mathbf{f}(\mathbf{x})$ typically has much fewer than n non-zero entries. Initial work ((Bricken et al., 2023); (Cunningham et al., 2023)) used a ReLU activation function to enforce non-negativity, and an L1 penalty on the decomposition $\mathbf{f}(\mathbf{x})$ to encourage sparsity. TopK SAEs (Gao et al., 2024) enforce sparsity by zeroing all but the top K entries of $\mathbf{f}(\mathbf{x})$, whereas the JumpReLU SAEs (Rajamanoharan et al., 2024b) enforce sparsity by zeroing out all entries of $\mathbf{f}(\mathbf{x})$ below a positive threshold. Both TopK and JumpReLU SAEs allow for greater separation between the tasks of determining which latents are active, and estimating their magnitudes.

2.2. Transcoders

Transcoders are closely related to SAEs but target a different objective: rather than sparsely reconstructing their inputs, they are trained to sparsely reconstruct the computation of an MLP sublayer. Concretely, a transcoder takes as input the pre-MLP residual stream (just after the pre-MLP RMSNorm) and learns to approximate the MLP's output. This makes transcoders particularly useful for circuit analysis: if we freeze (or otherwise control) attention patterns, the direct connections between two transcoder latents become linear, so both upstream attributions to a latent and downstream effects from that latent can be analyzed with far fewer confounders.

Formally, letting $\mathbf{x}$ denote the pre-MLP residual and $\mathbf{y}_{\text{MLP}}(\mathbf{x})$ the MLP output, a standard transcoder has encoder and decoder

$$\mathbf{f}_{\text{TC}}(\mathbf{x}) := \sigma(\mathbf{W}_{\text{enc}}\,\mathbf{x} + \mathbf{b}_{\text{enc}})$$

$$\hat{\mathbf{y}}_{\text{TC}}(\mathbf{f}) := \mathbf{W}_{\text{dec}}\,\mathbf{f} + \mathbf{b}_{\text{dec}}\,,$$

and is trained to minimize a reconstruction loss

$$\mathcal{L}_{\text{TC}} := \left\| \mathbf{y}_{\text{MLP}}(\mathbf{x}) - \hat{\mathbf{y}}_{\text{TC}}(\mathbf{f}_{\text{TC}}(\mathbf{x})) \right\|_2^2 .$$

Skip transcoders Despite their nonlinearity, it has been theorized that MLP sublayers exhibit some degree of linear behavior (Dunefsky et al., 2024). To capture such structure explicitly, we follow the approach in the aforementioned work and train skip transcoders that include an affine skip connection from the input directly to the output:

$$\hat{\mathbf{y}}_{\text{skip}}(\mathbf{f}, \mathbf{x}) := \mathbf{W}_{\text{dec}}\,\mathbf{f} + \mathbf{b}_{\text{dec}} + \mathbf{W}_{\text{skip}}\,\mathbf{x} .$$

Another motivation for this choice comes from the phenomena of **cross-layer superposition**, as described in e.g. Anthropic's circuit tracing work (Lindsey et al., 2024). This term describes when a single feature is distributed over latents in several layers, so just training SAEs on each layer independently can give an incomplete picture. In such cases, asking a transcoder to model this component as a learned linear map $\mathbf{W}_{\text{skip}}$ is more faithful and leads to cleaner attributions: the decoder $\mathbf{W}_{\text{dec}}$ focuses on genuinely new or nonlinear structure, while the skip term captures direct linear carry-through of latents such as rotations or other affine mappings.

2.3. JumpReLU SAEs

As in the previous release, we focus heavily on JumpReLU SAEs as they have been shown to be a slight Pareto improvement over other approaches, and have additional beneficial properties for training which will be discussed later in this section.

JumpReLU activation The JumpReLU activation is a shifted Heaviside step function as a gating mechanism together with a conventional ReLU:

$$\sigma(\mathbf{z}) = \text{JumpReLU}_{\boldsymbol{\theta}}(\mathbf{z}) := \mathbf{z} \odot H(\mathbf{z} - \boldsymbol{\theta}). \tag{3}$$

Here, $\boldsymbol{\theta} > 0$ is the JumpReLU's vector-valued learnable threshold parameter, $\odot$ denotes elementwise multiplication, and H is the Heaviside step function, which is 1 if its input is positive and 0 otherwise. Intuitively, the JumpReLU leaves the pre-activations unchanged above the threshold, but sets them to zero below the threshold, with a different learned threshold per latent.

Loss function As loss function we use a squared error reconstruction loss, and directly regularize the number of active (non-zero) latents using the L0 penalty:

$$\mathcal{L} := \|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))\|_2^2 + \lambda \|\mathbf{f}(\mathbf{x})\|_0, \tag{4}$$

where λ is the sparsity penalty coefficient. Since the L0 penalty and JumpReLU activation function are piecewise constant with respect to threshold parameters $\boldsymbol{\theta}$, we use straight-through estimators (STEs) to train $\boldsymbol{\theta}$, using the approach described in (Rajamanoharan et al., 2024b). This introduces an additional hyperparameter, the kernel density estimator bandwidth ε, which controls the quality of the gradient estimates used to train the threshold parameters $\boldsymbol{\theta}$.

Quadratic L0 penalty To target a specific expected sparsity, we also consider replacing the linear L0 term with a quadratic penalty around a target number of active latents $L_0^\star$:

$$\mathcal{L}_{\text{quad}} := \|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))\|_2^2 + \lambda \left(\frac{2}{L_0^\star}\right) \left(\|\mathbf{f}(\mathbf{x})\|_0 - L_0^\star\right)^2 . \tag{5}$$

The factor $\frac{2}{L_0^\star}$ scales gradients so that, when $\|\mathbf{f}(\mathbf{x})\|_0 \approx 2L_0^\star$, the magnitude of the sparsity gradient roughly matches that of the linear JumpReLU objective (Eq. (4)) at the same effective sparsity. This stabilizes training around the target $L_0^\star$ while providing a smooth force toward the desired activation frequency.

Direct frequency penalization One other advantage of JumpReLU SAEs is that we can directly target high-density latents by using their frequency in our sparsity penalty. We do this by using the STE approximation for L0, since the frequency of a given latent is simply the average L0 across a batch of data. This method was described in (Rajamanoharan et al., 2024), but we modify it slightly for the models trained in this release: rather than replacing the sparsity

penalty with one directly targeting frequency, we use the quadratic L0 penalty as our primary sparsity penalty but add a secondary penalty which specifically targets high-frequency latents.

2.4 End-to-End SAEs

After training our JumpReLU SAEs with MSE as our reconstruction loss, we finetune a select few using the end-to-end finetuning method introduced in (Braun et al., 2024) and further refined in (Karvonen, 2025). These methods propagate gradients through the base model during a short finetuning phase, with the goal of learning latents which are functionally important for the model's predictions rather than just for reconstructing activations.

Concretely, we finetune our SAEs and transcoders by optimizing the following finetuning objective:

$$\mathcal{L}_{\text{finetune}} := \frac{\text{MSE} + \alpha\,\beta\,\text{KL}(\mathbf{p}(\mathbf{x}), \mathbf{p}(\hat{\mathbf{x}}))}{1+\beta}. \tag{6}$$

where MSE denotes the SAE reconstruction loss, $\text{KL}(\mathbf{p}(\mathbf{x}), \mathbf{p}(\hat{\mathbf{x}}))$ is the KL divergence between the base model's distribution $p(\mathbf{x})$ and the distribution with SAE reconstructions injected into the model's forward pass $\mathbf{p}(\hat{\mathbf{x}})$, and β is a user-defined hyperparameter (e.g. if $\beta = 0$ this reduces to regular MSE training). Following the general motivation of KL-regularized E2E training in (Braun et al., 2024; Karvonen, 2025), we use a dynamically adjusted scaling factor

$$\alpha := \frac{\text{MSE}}{\text{KL} + 10^{-8}}. \tag{7}$$

which is treated as a constant with respect to gradients. This normalization ensures that β can be interpreted as the intended relative weight of the KL penalty compared to the reconstruction error, independent of their absolute magnitudes. This stabilizes training and simplifies hyperparameter selection across different layers, widths, and sparsity targets.

2.5. Instruction-tuned (IT) SAEs

For instruction-tuned models, we depart from the pretraining (PT) setup in two ways. First, rather than sampling from the same pretraining distribution, we construct training data from actual model rollouts (specifically, we take open-source datasets of user prompts and generate responses from the corresponding Gemma models). Second, we do not train IT SAEs from scratch: we initialize from the corresponding PT SAEs and finetune on the rollout-derived datasets. This approach is consistent with prior DeepMind results indicating that PT-to-IT transfer typically does not require resampling a large fraction of latents, and preserves both reconstruction quality and interpretability of learned latents (cf. (Kissane et al., 2024b)). In practice, initializing from PT SAEs accelerates convergence, stabilizes sparsity calibration, and yields IT SAEs that can be directly compared to their PT counterparts for circuit-level analyses.

2.6 Multi-layer SAEs

We release two different types of multi-layer autoencoder models: **weakly causal crosscoders** and **cross-layer transcoders**.

Weakly causal crosscoders Crosscoders were first introduced in (Lindsey et al., 2024). They are variants of regular sparse autoencoders which are trained not on a single activation site but on the concatenation of activations from multiple sites. This could mean the concatenation of activations from different base models, or from the same base model at different layers. In this paper, we refer only to the latter. Much like skip transcoders, the motivation for these models is to recover features which have been distributed across multiple layers, due to linear components of MLP or attention layers, or other effects. There are many variants of multi-layer crosscoders, depending on which layers are trained on and which architectural restrictions are imposed on the model. In this paper we focus on crosscoders which are only trained on a partial subset of layers rather than the full model, and assume weak causality: in other words, a latent's encoder weights are restricted to a single layer and its decoder may reconstruct activations from that layer or any future layer. This ensures latents cannot use future-layer information to encode past activations.

Cross-layer transcoders The cross-layer

transcoder (CLT) architecture was introduced in (Lindsey et al., 2024). Much like crosscoders generalize SAEs by training on the concatenation of multiple layers, cross-layer transcoders generalize transcoders by training to reconstruct the map from concatenated pre-MLP activations to concatenated MLP outputs. Note that cross-layer transcoders can also be combined with affine skip connections in exactly the same way as skip transcoders, with each affine skip connection only mapping from a layer's MLP input to that same layer's MLP output.

3. Training details

For this release, we largely kept to the same training methodology as (Lieberum et al., 2024). In particular, the topology and sharding configuration for our single-layer models was identical to the description given in the original Gemma Scope technical report, as is our shuffling method. In this report, we will only discuss an attribute of our training in detail when it differs from our methodology in the original release.

3.1. Data

We train SAEs on the activations of Gemma 3 models generated using text data from the same distribution as the pretraining text data for Gemma 3 (Gemma Team, 2025). For the instruction-tuned models, we finetuned our SAEs using chat data: the user prompts were taken from the open-source datasets OpenAssistant/oasst1 (Köpf et al., 2023) and LMSYS-Chat-1M (Zheng et al., 2023).

During training, activation vectors are normalized by a fixed scalar to have unit mean squared norm. This allows more reliable transfer of hyperparameters between layers and sites, as the raw activation norms can vary over multiple orders of magnitude, changing the scale of the reconstruction loss in Eq. (4). Once training is complete, we rescale the trained SAE parameters so that no input normalization is required for inference (see Appendix A in (Lieberum et al., 2024) for more details). This process is similar for multi-layer models; the only difference is that we normalize each layer separately. This increases stability of training especially when we initialize our multi-layer models from the concatenated weights of single-layer models (see Section 3.3).

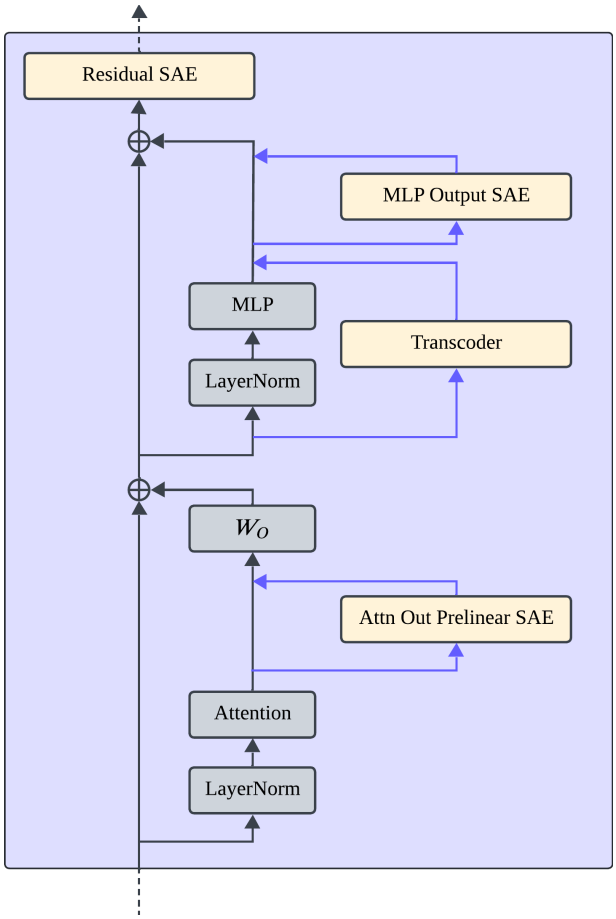

Figure 1 | Illustration of the three locations per layer where SAEs are trained: attention head outputs, MLP outputs, and post-MLP residual stream.

Location As in the previous Gemma Scope release, we train SAEs on three locations per layer. We train on the attention head outputs before the final linear transformation W_O and RMSNorm has been applied (Kissane et al., 2024a), on the MLP outputs after the RMSNorm has been applied and on the post MLP residual stream. For the attention output SAEs, we concatenate the outputs of the individual attention heads and learn a joint SAE for the full set of heads. We zero-index the layers, so layer 0 refers to the first transformer block after the embedding layer. We also train a full suite of skip transcoders on every layer. This is illustrated in Fig. 1. Additionally, for each model we train a partial weakly causal crosscoder on 4 layers chosen at fixed-depth percentiles (25%, 50%, 65%

Gemma 3 Model	SAE Type	Layers	SAE Widths	L0s
	SAE [a]	All	{16k, 256k}	{10, 100}
	SAE [a,c]	{5, 9, 12, 15}	{16k, 64k, 256k, 1m}	{10, 50, 150}
	transcoder [b]	All	{16k, 256k}	{10, 100}
	transcoder [b,c]	{5, 9, 12, 15}	{16k, 64k, 256k}	{10, 50, 150}
	crosscoder [c]	{5, 9, 12, 15}	{64k, 256k, 512k, 1m}	{50, 150}
	CLT [b,c]	All	{256k, 512k}	{50, 150}
	SAE [a]	All	{16k, 256k}	{10, 100}
	SAE [a,c]	{7, 13, 17, 22}	{16k, 64k, 256k, 1m}	{10, 50, 150}
	transcoder [b]	All	{16k, 256k}	{10, 100}
	transcoder [b,c]	{7, 13, 17, 22}	{16k, 64k, 256k}	{10, 50, 150}
	crosscoder [c]	{7, 13, 17, 22}	{64k, 256k, 512k, 1m}	{50, 150}
	CLT [b,c]	All	{256k, 512k}	{50, 150}
	SAE [a]	All	{16k, 256k}	{10, 100}
	SAE [a,c]	{9, 17, 22, 29}	{16k, 64k, 256k, 1m}	{10, 50, 150}
	transcoder [b]	All	{16k, 256k}	{10, 100}
	transcoder [b,c]	{9, 17, 22, 29}	{16k, 64k, 256k}	{10, 50, 150}
	crosscoder [c]	{9, 17, 22, 29}	{64k, 256k, 512k, 1m}	{50, 150}
	SAE [a]	All	{16k, 256k}	{10, 100}
	SAE [a,c]	{12, 24, 31, 41}	{16k, 64k, 256k, 1m}	{10, 50, 150}
	transcoder [b]	All	{16k, 256k}	{10, 100}
	transcoder [b,c]	{12, 24, 31, 41}	{16k, 64k, 256k}	{10, 50, 150}
	crosscoder [c]	{12, 24, 31, 41}	{64k, 256k, 512k, 1m}	{50, 150}
	SAE [a]	All	{16k, 256k}	{10, 100}
	SAE [a,c]	{16, 31, 40, 53}	{16k, 64k, 256k, 1m}	{10, 50, 150}
	transcoder [b]	All	{16k, 256k}	{10, 100}
	transcoder [b,c]	{16, 31, 40, 53}	{16k, 64k, 256k}	{10, 50, 150}
	crosscoder [c]	{16, 31, 40, 53}	{64k, 256k, 512k, 1m}	{50, 150}

Table 1 | Overview of the SAEs & variants that were trained for different Gemma 3 models. The layers column indicates either multiple different releases for the single-layer models, or multiple layers trained on simultaneously for the multi-layer models. [a] Each SAE corresponds to an SAE trained on 3 different sites: attention output, MLP output and post-MLP residual. Only the residual stream SAEs have a 1m-width model released. [b] Each transcoder and CLT corresponds to a sweep over 2 different configs: with and without affine skip connections. [c] These variants also include random seeds for exactly one of the combinations of SAE width & L0. Every model listed in this table comes with a finetuned variant for the instruction-tuned version of the corresponding Gemma 3 model.

and 85% of the way through the model), and for the two smaller models (270M, 1B) we also train cross-layer transcoders.

3.2. Hyperparameters

Optimization We use the same bandwidth $\varepsilon = 0.001$ and learning rate $\eta = 7 \times 10^{-5}$ across all training runs. We use a cosine learning rate warmup from 0.1η to η over the first 1,000 training steps. We train with the Adam optimizer (Kingma and Ba, 2017) with $(\beta_1, \beta_2) = (0, 0.999)$, $\epsilon = 10^{-8}$ and a batch size of 4,096. We use a quadratic L0 penalty, and combine this with a linear warmup for the sparsity coefficient from 0 to λ over the first 50,000 training steps.

During training, we parameterize the SAE using a pre-encoder bias (Bricken et al., 2023), subtracting $\mathbf{b}_{\text{dec}}$ from activations before the encoder. However, after training is complete, we fold in this bias into the encoder parameters, so that no pre-encoder bias needs to be applied during inference. Throughout training, we restrict the columns of $\mathbf{W}_{\text{dec}}$ to have unit norm by renormalizing after every update. We also project out the part of the gradients parallel to these columns before computing the Adam update, as described in (Bricken et al., 2023).

Initialization We initialize the JumpReLU threshold as the vector $\boldsymbol{\theta} = \{0.001\}^M$. We initialize $\mathbf{W}_{\text{dec}}$ using He-uniform (He et al., 2015) initialization and rescale each latent vector to be unit norm. $\mathbf{W}_{\text{enc}}$ is initialized as the transpose of $\mathbf{W}_{\text{dec}}$, but they are not tied afterwards ((Conerly et al., 2024); (Gao et al., 2024)). The biases $\mathbf{b}_{\text{dec}}$ and $\mathbf{b}_{\text{enc}}$ are initialized to zero vectors. For multi-layer models we initialize using the parameters of the corresponding single-layer models, as we discuss in Section 3.3.

3.3 Multi-layer model initialization

Despite these improvements, multi-layer models are still much more costly than single-layer models to train. To overcome these issues, we initialize our multi-layer models using our single-layer models as a starting point.

One possible method we explored was to initialize our multi-layer models by simply concatenating single-layer models. Motivated by the fact that we were using Matryoshka training for our SAEs, we would choose prefixes of latents from each single-layer SAE to include in the multi-layer model. The problem we ran into was redundant latents: this method would pick latents on different layers which represented more or less the same concept. To fix this, we developed a novel initialization strategy which works as follows: we iterate through SAEs (starting from the earliest layers), choosing prefixes of latents from each SAE. For each latent we choose, we mark off each of the latents in later-layer SAEs which have the maximum similarity to this latent (as measured by the dot product between the early-layer decoder and later-layer encoder). In this way, we get much better global coverage, because at each layer we will avoid choosing latents which were too similar to one that we already chose in a previous layer.

Generally for our multi-layer models we target smaller L0 values than we would get from this initialization strategy, but we also want the finetuning process to be stable. To fix this, we initially set the target L0 value high (based on the sum of the single-layer L0 values of all the latents we've chosen in our initialization strategy) and then decay it over 50,000 steps to our target value for the multi-layer model. We do this for both the weakly causal crosscoders and the CLTs.

4. Evaluation

In this section we evaluate the trained SAEs from various different angles. We note however that as of now there is no consensus on what constitutes a reliable metric for the quality of a sparse autoencoder or its learned latents and that this is an ongoing area of research and debate ((Gao et al., 2024); (Karvonen et al., 2024); (Makelov et al., 2024)).

Unless otherwise noted, all evaluations are on sequences from the same distribution as the SAE training data, i.e. the pretraining distribution of Gemma 3.

4.1. Evaluating the sparsity-fidelity trade-off

Methodology For a fixed dictionary size, we trained SAEs of varying levels of sparsity by sweeping the L0 target value $L_0^\star$. We then plot curves showing the level of reconstruction fidelity attainable at a given level of sparsity.

Metrics We use the mean L0-norm of latent activations, $\mathbb{E}_{\mathbf{x}}\|\mathbf{f}(\mathbf{x})\|_0$, as a measure of sparsity. To measure reconstruction fidelity, our primary metrics are **delta LM loss** which is the increase in the cross-entropy loss experienced by the LM when we splice the SAE into the LM's forward pass, and **fraction of variance unexplained (FVU)**, also called the normalized loss (Gao et al., 2024) - as a measure of reconstruction fidelity. FVU is mean reconstruction loss $\mathcal{L}_{\text{reconstruct}}$ of a SAE normalized by the reconstruction loss obtained by always predicting the dataset mean. Note that FVU is purely a measure of the SAE's ability to reconstruct the input activations, not taking into account the causal effect of any error on the downstream loss.

All metrics were computed on 2,048 sequences of length 1,024, after masking out special tokens (pad, start and end of sequence) when aggregating the results.

Results The sparsity-fidelity trade-off for SAEs in the middle of each Gemma model is illustrated in Figure 7. As in the previous release, we found delta loss to be consistently higher for residual stream SAEs compared to MLP and attention SAEs, whereas FVU is roughly comparable across sites.

4.2. Latent firing frequency

Fig. 2 shows the distribution of latent activation frequencies for the latents in the residual stream SAEs across model sizes and depths. This was computed across a set of 50,000 sequences of length 1,024 after masking out special tokens. With an aggressive version of the dense latent penalization we discussed in Section 2.3, we find that we can entirely eliminate latents with frequency greater than 10%.

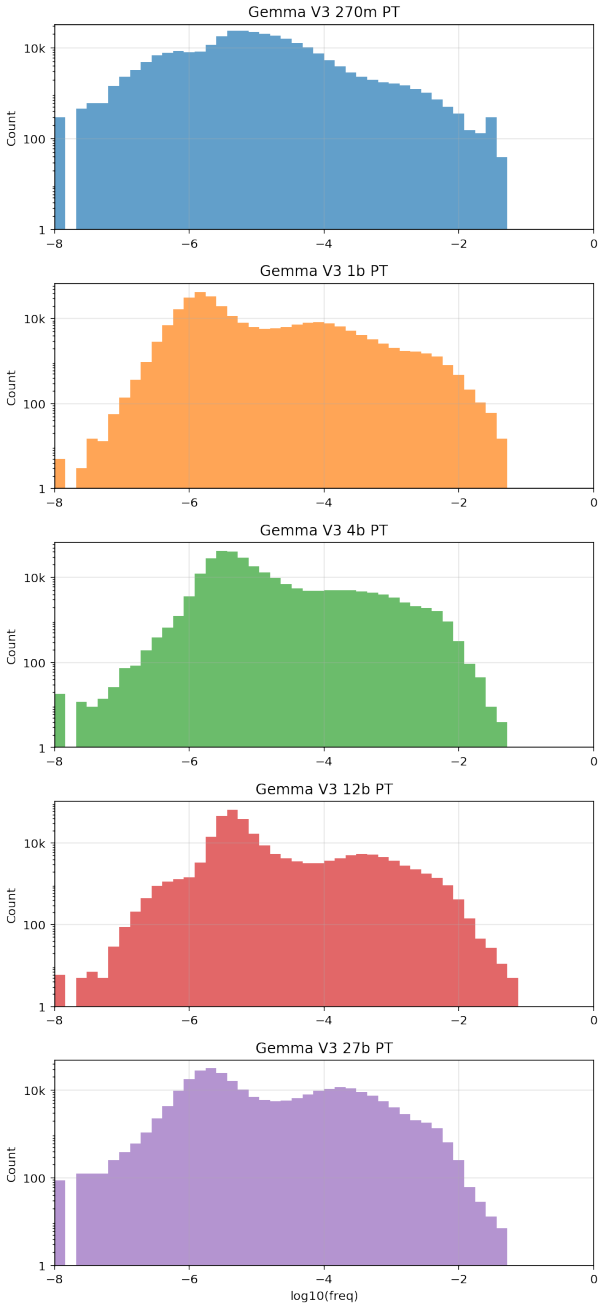

Figure 2 | Feature activation frequency distributions for residual post-MLP SAEs across model sizes and depths; most latents remain low frequency with long-tailed densities.

4.3 Interpretability of latents

We evaluate interpretability using an automated interpretability system rather than human raters. The method involves binary classification: we present sequences where a particular latent fires and sequences where it doesn't, and ask a model to generate an explanation for this feature. Next, we present this explanation along with a randomly ordered list of sequences (some of which cause the feature to fire, some of which don't) and ask the model to classify which ones fire. Our findings are broadly consistent with a snippet we published earlier this year: lower-frequency latents tend to be more interpretable. Figure 3 shows the distribution of interpretability scores for the latents in residual stream SAEs trained on Gemma V3 1B PT, at four different layers.

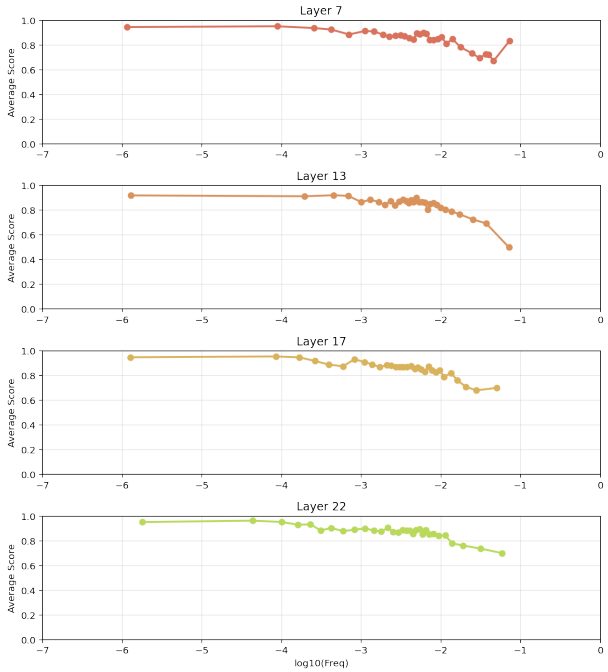

Figure 3 | Automated interpretability scores as a function of feature activation frequency across model sizes and depths, for Gemma V3 1B PT SAEs trained on the residual stream. Higher-frequency features are slightly less interpretable.

4.4. Affine skip connections in transcoders

By giving us more learnable parameters and the ability to model the linear parts of MLP layers without dedicating transcoder latents to it, affine skip connections can improve our performance in both the single and multi-layer setting. This is shown in Figure 4, where we compare the effect of adding affine skip connections to transcoders and CLTs respectively on the model FVU.

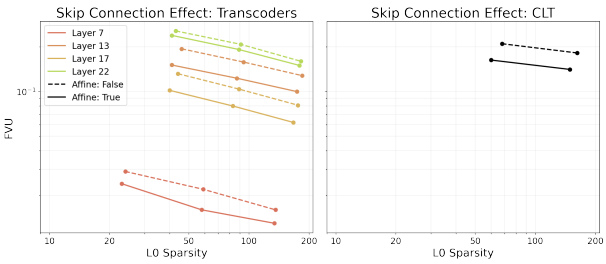

Figure 4 | Effect of affine skip connections on reconstruction quality: FVU versus L0 for transcoders and CLTs, showing improved trade-offs with skip connections.

We can also measure the usefulness of affine skip connections another way. In (Lindsey et al., 2025), the authors show that the circuit-tracing algorithm can be applied to cross-layer transcoders (CLTs) to generate graphs of latents which fire on a particular prompt, and then prune that graph to leave only latents which are important for a particular token prediction. The authors also compare this to the graphs generated from a suite of single-layer transcoders. We compare our trained CLTs and transcoders by generating attribution graphs for each of them, and generally find the same results as the authors: CLTs generate graphs with higher sparsity, as measured both by the number of nodes and edges in the graph. Figure 5 visualizes this by showing the number of latent nodes required to reach a given fraction of total circuit influence (measured using Anthropic's influence metric). Not only do we see CLTs outperforming transcoders (since any given prefix of nodes leads to a greater total influence), but we also see affine skip connections outperform for both transcoders and CLTs.

4.5. Initializing multi-layer models from trained single-layer models

We initialize multi-layer models using weights from trained single-layer models and gradually decay the target L0. This reduces wall-clock training time because single-layer models train

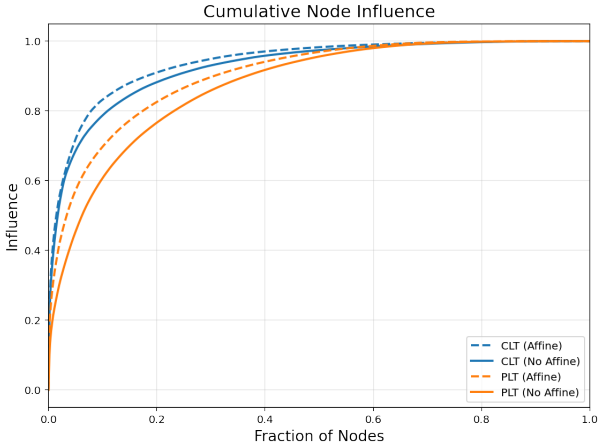

Figure 5 | Cumulative influence graph for CLTs vs transcoders for Gemma V3 1B IT, on the prompt "The National Data Authority (N".

and parallelize more efficiently than randomly-initialized multi-layer models. Figure 6 shows the average cosine similarity between decoder weights and their initialized values during the course of training, for a crosscoder which was initialized from several single-layer SAEs. Although the cosine similarity does come down by the end of training, it still remains fairly high, suggesting that the initialized features from single-layer models are good approximations to what the multi-layer model eventually needs to learn.

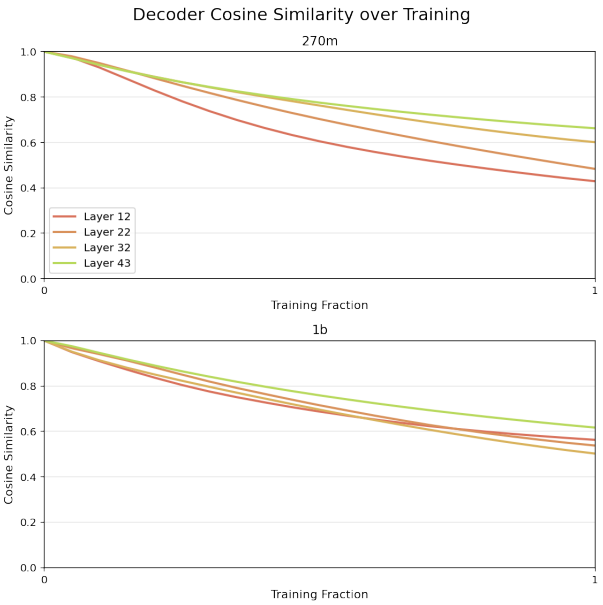

Figure 6 | Training dynamics for weakly causal crosscoders initialized from single-layer SAEs.

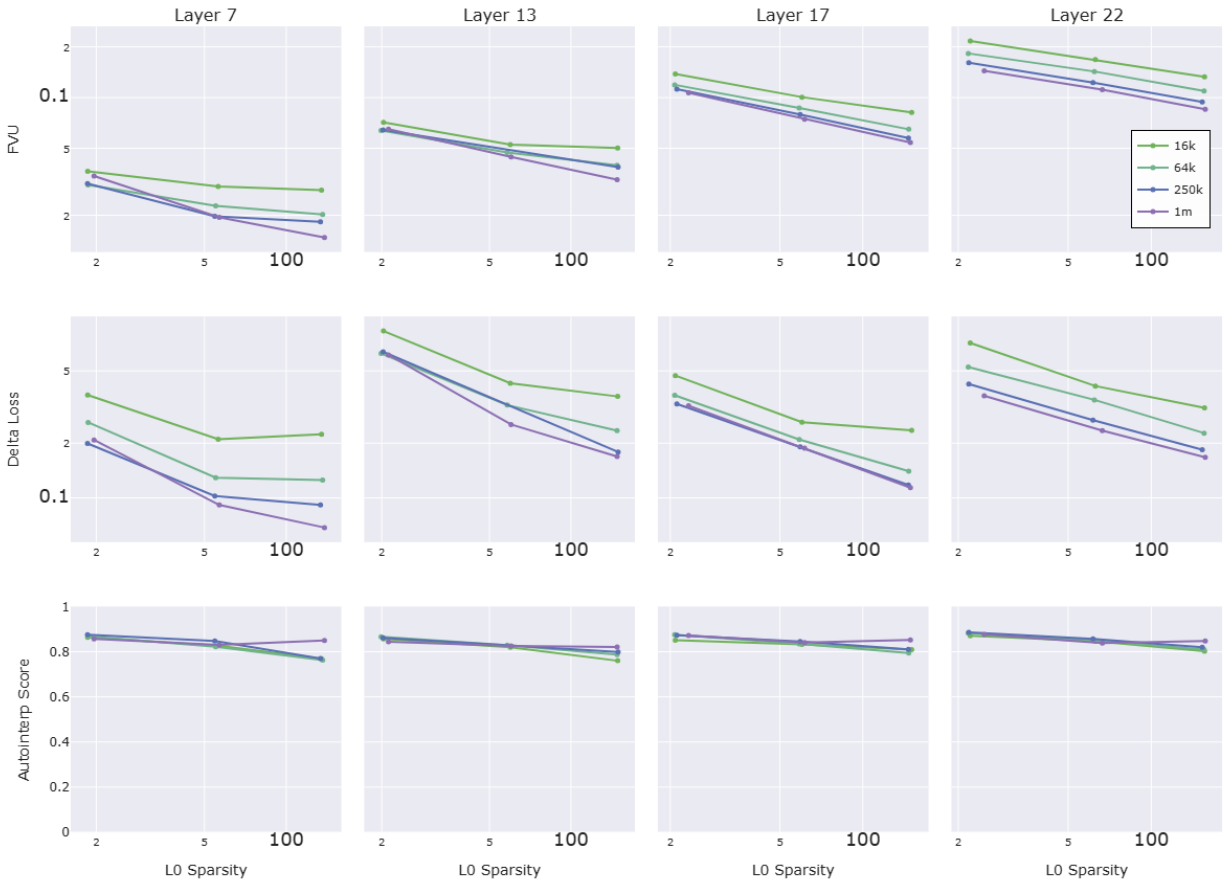

Figure 7 | Sparsity–fidelity trade-off for Gemma 3 1B resid-post SAEs, and autointerpretability scores. Higher L0s (and wider SAEs) lead to better performance, without having a significant impact on latent interpretability.

5. Open problems that Gemma Scope 2 may help tackle

As for the original Gemma Scope release, we're excited to help the broader safety and interpretability communities advance our understanding of interpretability, and how it can be used to make models safer. In this section we provide a list of open problems we're particularly excited to see on. The list reflects how our own views on sparse autoencoder research (and interpretability more broadly) have changed over the past year as well as the kinds of research this release is especially well suited for. For example, we're interested in large-scale circuit analysis as well as using sparse autoencoders for real-world tasks such as debugging strange model behaviors, but we're less excited about fundamental research into new SAE architectures.

Deepening our understanding of SAEs

1. Comparisons of residual stream SAE features across layers, e.g. are there persistent features that one can "match up" across adjacent layers? How can multi-layer models help us understand this?
2. Better understanding the phenomenon of "feature splitting" (Bricken et al., 2023) where high-level features in a small SAE break apart into several finer-grained features in a wider SAE. Do Matryoshka SAEs help resolve this?
3. We know that SAEs introduce error, and completely miss out on some features that are captured by wider SAEs ((Bussmann et al., 2024); (Templeton et al., 2024)). Can we quantify and easily measure "how much" they miss and how much this matters in practice?
4. How are circuits connecting up superposed features represented in the weights? How do models deal with the interference between features (Nanda et al., 2023b)?

Using SAEs for real-world applications and understanding model behavior

1. Detecting or fixing jailbreaks, and understanding the mechanisms by which jailbreaks succeed or fail.
2. Helping find new jailbreaks/red-teaming models (Ziegler et al., 2022).
3. Understanding real-world failures in model reasoning and alignment, such as hallucinations, unfaithful chain of thought, and emergent misalignment in finetuned or in-context learning scenarios.
4. Comparing steering vectors (Turner et al., 2024) to SAE feature steering (Conmy and Nanda, 2024) or clamping (Templeton et al., 2024) for controlling model behavior.
5. Can SAEs be used to improve interpretability techniques, like steering vectors, such as by removing irrelevant features (Conmy and Nanda, 2024)?
6. Using SAEs to identify and remove spurious correlations or discover causal structures in model reasoning (Marks et al., 2024).
7. Auditing games: can we use SAEs to verify whether models are reasoning faithfully, planning deceptively, or pursuing hidden goals?

Red-teaming SAEs

1. Can we find downstream tasks where SAEs can be measured against simple baselines (either black-box or simpler white-box methods)? How do they perform?
2. How robust are claims about the interpretability of SAE features (Huang et al., 2023)?
3. Can we find the "dark matter" of truly non-linear features?
4. Do SAEs learn spurious compositions of independent features to improve sparsity as has been shown to happen in toy models (Anders et al., 2024), and can we fix this?

Scalable circuit analysis: What interesting circuits can we find in these models?

1. What's the learned algorithm for addition (Stolfo et al., 2023) in Gemma 3 4B? Does it resemble that found by (Lindsey et al., 2025) ?
2. How can we practically extend the SAE feature circuit finding algorithm in (Marks et al., 2024) to larger models?
3. Can we use single-layer transcoders (Dunefsky et al., 2024) to find input independent, weights-based circuits?

Using SAEs as a tool to answer existing questions in interpretability

1. What does finetuning do to a model's internals (Jain et al., 2024)? Can SAEs detect the traces left by finetuning (Minder et al., 2025)?
2. What is actually going on when a model uses chain of thought? What changes when the chain of thought is unfaithful?
3. Is in-context learning true learning, or just promoting existing circuits ((Hendel et al., 2023); (Todd et al., 2024))?
4. Can we find any "macroscopic structure" in language models, e.g. families of features that work together to perform specialised roles, like organs in biological organisms?

Acknowledgements

We are incredibly grateful to Joseph Bloom, Johnny Lin and Curt Tigges for their help supporting more interactive demos of Gemma Scope on Neuronpedia (Lin and Bloom, 2023), creating tooling for researchers like feature dashboards, and help making educational materials. Their work extended beyond the original feature dashboards used in Gemma Scope, and included visualizations of the circuit tracing methodology applied to our transcoder models.

Author contributions

Callum McDougall (CM) led the writing of the report and the bulk of this project, but it wouldn't have been possible without much supporting work such as the implementation and running of evaluations. Tom Lieberum and Vikrant Varma primarily designed the origin sparse autoencoder training codebase which was adapted for this work, with significant contributions from Arthur Conmy (AC). Lewis Smith (LS) wrote the original Gemma Scope tutorial, which was adapted by CM. Senthooran Rajamanoharan (SR) developed the JumpReLU architecture which was primarily used in this work. CM led the early access and open sourcing of code and weights. Neel Nanda (NN) provided advice and mentorship throughout the project. Sparse autoencoder visualization and autointerpretability evaluations were written and implemented by CM.

References

E. Anders, C. Neo, J. Hoelscher-Obermaier, and J. N. Howard. Sparse autoencoders find composed features in small toy models. LessWrong, 2024.

Y. Belinkov. Probing classifiers: Promises, shortcomings, and advances. Computational Linguistics, 48(1):207–219, 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7.

S. Bills, N. Cammarata, D. Mossing, H. Tillman, L. Gao, G. Goh, I. Sutskever, J. Leike, J. Wu, and W. Saunders. Language models can explain neurons in language models. OpenAI, 2023. URL https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html.

T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language

models with dictionary learning. Transformer Circuits Thread, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

B. Bussmann, P. Leask, J. Bloom, C. Tigges, and N. Nanda. Stitching SAEs of different sizes. Alignment Forum, 2024. URL https://www.alignmentforum.org/posts/baJyjpktzmcmRfosq/stitching-saes-of-different-sizes.

T. Conerly, A. Templeton, T. Bricken, J. Marcus, and T. Henighan. Update on how we train SAEs. Transformer Circuits Thread, 2024. URL https://transformer-circuits.pub/2024/april-update/index.html#training-saes.

A. Conmy and N. Nanda. Activation steering with SAEs. Alignment Forum, 2024. URL https://www.alignmentforum.org/posts/C5KAZQib3bzzpeyrg/progress-update-1.

H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey. Sparse autoencoders find highly interpretable features in language models. 2023.

J. Dunefsky, P. Chlenski, and N. Nanda. Transcoders find interpretable LM feature circuits. 2024. URL https://arxiv.org/abs/2406.11944.

N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah. Toy models of superposition. Transformer Circuits Thread, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

J. Engels, I. Liao, E. J. Michaud, W. Gurnee, and M. Tegmark. Not all language model features are linear. 2024. URL https://arxiv.org/abs/2405.14860.

L. Gao, T. D. la Tour, H. Tillman, G. Goh, R. Troll, A. Radford, I. Sutskever, J. Leike, and J. Wu. Scaling and evaluating sparse autoencoders. 2024. URL https://arxiv.org/abs/2406.04093.

Gemma Team. Gemma 3: Open language models. 2025. URL https://arxiv.org/abs/2503.19786.

W. Gurnee, N. Nanda, M. Pauly, K. Harvey, D. Troitskii, and D. Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. Transactions on Machine Learning Research, 2023. URL https://openreview.net/forum?id=JYs1R9IMJr.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. 2015. URL https://arxiv.org/abs/1502.01852.

J. Minder, C. Dumas, S. Slocum, H. Casademunt, C. Holmes, R. West, and N. Nanda. Narrow finetuning leaves clearly readable traces in activation differences. URL https://arxiv.org/abs/2510.13900.

R. Hendel, M. Geva, and A. Globerson. In-context learning creates task vectors. In EMNLP 2023. URL https://openreview.net/forum?id=QYvFUlF19n.

J. Huang, A. Geiger, K. D'Oosterlinck, Z. Wu, and C. Potts. Rigorously assessing natural language explanations of neurons. 2023. URL https://arxiv.org/abs/2309.10312.

E. Hubinger. A transparency and interpretability tech tree. Alignment Forum, 2022.

S. Jain, E. S. Lubana, K. Oksuz, T. Joy, P. H. S. Torr, A. Sanyal, and P. K. Dokania. What makes and breaks safety fine-tuning? A mechanistic study. 2024. URL https://arxiv.org/abs/2407.10264.

A. Jermyn, C. Olah, and T. Henighan. Attention head superposition. Transformer Circuits Thread, 2023. URL https://transformer-circuits.pub/2023/may-update/index.html#attention-superposition.

A. Karvonen, B. Wright, C. Rager, R. Angell, J. Brinkmann, L. R. Smith, C. M. Verdun, D.

Bau, and S. Marks. Measuring progress in dictionary learning for language model interpretability with board game models. In ICML 2024 Workshop on Mechanistic Interpretability, 2024. URL https://openreview.net/forum?id=qzsDKwGJyB.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2017. URL https://arxiv.org/abs/1412.6980.

C. Kissane, R. Krzyzanowski, J. I. Bloom, A. Conmy, and N. Nanda. Interpreting attention layer outputs with sparse autoencoders. 2024. URL https://arxiv.org/abs/2406.17759.

D. Braun, J. Taylor, N. Goldowsky-Dill, and L. Sharkey. Identifying functionally important features with end-to-end sparse dictionary learning. arXiv preprint arXiv:2405.12241, 2024. :contentReference[oaicite:0]index=0

A. Karvonen. Revisiting end-to-end sparse autoencoder training: A short finetune is all you need. arXiv preprint arXiv:2503.17272, 2025. :contentReference[oaicite:1]index=1

C. Kissane, R. Krzyzanowski, A. Conmy, and N. Nanda. SAEs (usually) transfer between base and chat models. Alignment Forum, 2024. URL https://www.alignmentforum.org/posts/fmwk6qxrpW8d4jvbd/saes-usually-transfer.

A. Köpf, E. Kilincer, I. Kutlu, Y. Sawada, J. Long, O. Basturk, R. Rychkova, A. Hartmann, V. Nguyen, N. Matti, J. Wei, K. Wong, M. Wolff, J. Wang, E. Abbott, K. Nguyen, A. Presland, F. Münch, C. Liu, C. Panait, D. Hay, R. Paynter, A. Pelakh, B. Chen, D. Reddy, J. Jain, T. Mann, M. Schmidt, T. Georgiou, Z. Hu, V. Silkin, N. Vogt, J. Cantara, H. Prince, M. Balint, and A. Meemken. OpenAssistant conversations – democratizing large language model alignment. NeurIPS 2023 Datasets and Benchmarks Track, 2023. URL https://arxiv.org/abs/2304.07327.

K. Li, O. Patel, F. Viégas, H. Pfister, and M. Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. NeurIPS 2023. URL https://openreview.net/forum?id=aLLuYpn83y.

T. Lieberum, V. Varma, A. Conmy, S. Rajamanoharan, and N. Nanda. Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2. 2024. URL https://arxiv.org/abs/2408.05147.

J. Lin and J. Bloom. Analyzing neural networks with dictionary learning. 2023. URL https://www.neuronpedia.org.

J. Lindsey, A. Templeton, J. Marcus, T. Conerly, J. Batson, and C. Olah. Sparse crosscoders for cross-layer features and model diffing. Transformer Circuits Thread, 2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html.

J. Lindsey, W. Gurnee, E. Ameisen, B. Chen, A. Pearce, N. L. Turner, C. Citro, D. Abrahams, S. Carter, B. Hosmer, J. Marcus, M. Sklar, A. Templeton, T. Bricken, C. McDougall, H. Cunningham, T. Henighan, A. Jermyn, A. Jones, A. Persic, Z. Qi, T. B. Thompson, S. Zimmerman, K. Rivoire, T. Conerly, C. Olah, and J. Batson. On the biology of a large language model. Transformer Circuits Thread, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

A. Makelov, G. Lange, and N. Nanda. Towards principled evaluations of sparse autoencoders for interpretability and control. In ICLR 2024 Workshop on Secure and Trustworthy Large Language Models, 2024. URL https://openreview.net/forum?id=MHIX9H8aYF.

S. Marks, C. Rager, E. J. Michaud, Y. Belinkov, D. Bau, and A. Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. 2024.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. 2013. URL https://arxiv.org/abs/1301.3781.

N. Nanda. A longlist of theories of impact for interpretability. Alignment Forum, 2022.

N. Nanda and J. Bloom. TransformerLens. 2022. URL https://github.com/TransformerLensOrg/TransformerLens.

N. Nanda, A. Lee, and M. Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In BlackboxNLP 2023, pages 16–30, 2023. URL https://aclanthology.org/2023.blackboxnlp-1.2.

N. Nanda, S. Rajamanoharan, J. Kramar, and R. Shah. Fact finding: Attempting to reverse-engineer factual recall on the neuron level. 2023.

C. Olah. Interpretability. Alignment Forum, 2021.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. Distill, 2020. doi: 10.23915/distill.00024.001. URL https://distill.pub/2020/circuits/zoom-in.

K. Park, Y. J. Choe, and V. Veitch. The linear representation hypothesis and the geometry of large language models. 2023.

S. Rajamanoharan, A. Conmy, L. Smith, T. Lieberum, V. Varma, J. Kramár, R. Shah, and N. Nanda. Improving dictionary learning with gated sparse autoencoders. arXiv preprint arXiv:2404.16014, 2024.

S. Rajamanoharan, T. Lieberum, N. Sonnerat, A. Conmy, V. Varma, J. Kramár, and N. Nanda. Jumping ahead: Improving reconstruction fidelity with JumpReLU sparse autoencoders. 2024. URL https://arxiv.org/abs/2407.14435.

A. Stolfo, Y. Belinkov, and M. Sachan. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In EMNLP 2023, 2023. URL https://openreview.net/forum?id=aB3Hwh4UzP.

A. Templeton, T. Conerly, J. Marcus, J. Lindsey, T. Bricken, B. Chen, A. Pearce, C. Citro, E. Ameisen, A. Jones, H. Cunningham, N. L. Turner, C. McDougall, M. MacDiarmid, C. D. Freeman, T. R. Sumers, E. Rees, J. Batson, A. Jermyn, S. Carter, C. Olah, and T. Henighan. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. Transformer Circuits Thread, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

E. Todd, M. Li, A. S. Sharma, A. Mueller, B. C. Wallace, and D. Bau. Function vectors in large language models. In ICLR 2024, 2024. URL https://openreview.net/forum?id=AwyxtyMwaG.

A. M. Turner, L. Thiergart, G. Leech, D. Udell, J. J. Vazquez, U. Mini, and M. MacDiarmid. Activation addition: Steering language models without optimization. 2024. URL https://arxiv.org/abs/2308.10248.

M. Wattenberg and F. Viégas. Relational composition in neural networks: A gentle survey and call to action. In ICML 2024 Workshop on Mechanistic Interpretability, 2024. URL https://openreview.net/forum?id=zzCEiUIPk9.

L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, H. Gonzalez, and I. Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. NeurIPS 2023. URL https://arxiv.org/abs/2306.05685.

D. Ziegler, S. Nix, L. Chan, T. Bauman, P. Schmidt-Nielsen, T. Lin, A. Scherlis, N. Nabeshima, B. Weinstein-Raun, D. de Haas, B. Shlegeris, and N. Thomas. Adversarial training for high-stakes reliability. In NeurIPS 2022, volume 35, pages 9274–9286. Curran Associates, Inc., 2022.

A. Matryoshka Frequencies

We can analyze the effect of the Matryoshka loss function on our features. Based on this penalty, we would expect that the features with the smallest indices are generally the more important ones for reconstructing the model's activations. Since a latent's contribution to loss can be decomposed as the product of its firing frequency and average loss

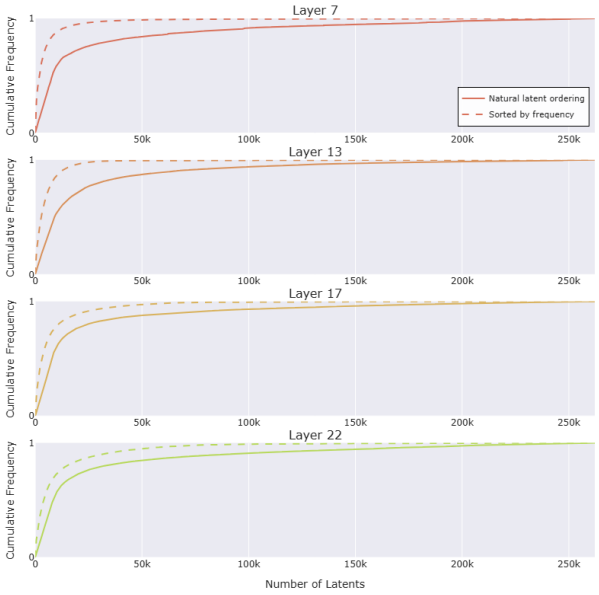

Figure 8 | Matryoshka feature frequencies for SAEs trained on Gemma V3 1B PT, residual stream. Solid line lies above diagonal (indicating earlier features have higher frequency), but below dotted line (indicating that the Matryoshka loss isn't so strong that it makes the SAEs strictly order their latents by frequency).

contribution when it fires, we would also expect early latents to have higher frequencies. This is borne out in Figure 8, which shows the early features having significantly higher frequency than the later features.

B. Sparse Kernels

We experimented with training BatchTopK SAEs, and used sparse kernels to implement this training. Although we ended up including only JumpReLU SAEs in our final release, and didn't find it beneficial to extend the sparse kernel methodology to JumpReLU SAEs at the scale we were training at, we include the theory and implementation details here in case they are of use to anyone who might train their own, particularly if using JAX and not working inside an existing training framwork.

B.1. TopK and BatchTopK SAEs

TopK SAEs enforce sparsity by selecting exactly the K most active latents per token, zeroing all others. Using the same notation as Section 2.1, let

$$\mathbf{f}_{\text{TopK}}(\mathbf{x}) := \text{TopK}_K(\mathbf{W}_{\text{enc}}\,\mathbf{x} + \mathbf{b}_{\text{enc}})\,. \tag{8}$$

BatchTopK extends this across a batch while keeping a fixed total number of active latents. For inputs $\{\mathbf{x}_b\}_{b=1}^{B}$, define pre-activations $\mathbf{z}_b := \mathbf{W}_{\text{enc}}\,\mathbf{x}_b + \mathbf{b}_{\text{enc}}$. BatchTopK selects the $K\,B$ largest entries across $\{\mathbf{z}_b\}_{b=1}^{B}$ and zeros out the rest:

$$\begin{aligned}\{\mathbf{f}_b\}_{b=1}^{B} &:= \text{BatchTopK}_K\big(\{\mathbf{z}_b\}_{b=1}^{B}\big)\,,\\ \mathbf{z}_b &:= \mathbf{W}_{\text{enc}}\,\mathbf{x}_b + \mathbf{b}_{\text{enc}}.\end{aligned} \tag{9}$$

Both approaches use the same linear decoder and reconstruction as Eq. (2), typically optimizing only the reconstruction loss

$$\mathcal{L}_{\text{reconstruct}} := \|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))\|_2^2, \tag{10}$$

since sparsity is enforced by the hard TopK constraints rather than an ℓ_0/ℓ_1 penalty. These methods provide TPU benefits similar to JumpReLU's sparse regimes: knowing the exact number of active latents (per token for TopK, per batch for BatchTopK) allows JIT compilation of sparse kernels with predictable shapes and memory traffic.

Crucially, BatchTopK can be converted to a JumpReLU parameterization at inference by selecting per-latent thresholds $\boldsymbol{\theta}$ that match the empirical activation quantiles observed during training, yielding

$$\mathbf{f}_{\text{JR}}(\mathbf{x}) := \text{JumpReLU}_{\boldsymbol{\theta}}(\mathbf{W}_{\text{enc}}\,\mathbf{x} + \mathbf{b}_{\text{enc}})\,, \tag{11}$$

with thresholds chosen so that active sets closely match those induced by BatchTopK. In practice, we generally use JumpReLU for single-layer SAEs. For multi-layer SAEs and transcoders, we typically train with BatchTopK (to realize TPU efficiency from sparse kernels) and convert to JumpReLU for inference, which is a favorable trade-off for large-scale circuit analyses.

B.2. Sparse Kernel Implementation

We implement sparse decoding in a JAX-friendly way, adapting the sparse kernel ideas of Gao et al. to our multi-layer models. During training we use only model parallelism, sharding along the latent dimension.

Sharding and activation selection Let activations have shape (B, L_{in}, F) for batch size B, input layers L_{in}, and latents per layer F. With S shards over the latent axis, each shard holds $(B, L_{\text{in}}, F/S)$. For TopK (target total K active latents across all layers), each shard independently selects $K/(L_{\text{in}}S)$ per example from its last dimension. For BatchTopK, each shard selects $(K\,B)/S$ across the shard. We return sparse tensors (values and indices), which remain sharded over latents (uniform by construction) but not over batch.

Sparse decoder The decoder has shape $(L_{\text{in}}, F, L_{\text{out}}, d_{\text{model}})$ and is sharded on its latent axis. For each shard we gather decoder vectors at the sparse indices and sum within each $(\text{batch}, \text{layer})$ group, producing per-shard outputs of shape $(B, L_{\text{out}}, d_{\text{model}})$. Summing across shards yields the final output.

Why stack by layer? Enforcing uniformity over the flattened $(B, L_{\text{in}} \cdot F)$ axis would implicitly impose a uniform activation budget across layers, which is undesirable. Multi-layer model training should allocate activations non-uniformly across depth (empirically we see allocations rise through the network and drop near the end). Enforcing uniformity only across the latent axis within a layer is a much weaker constraint. One residual limitation is that latents in different shards never compete in the TopK, so cross-shard suppression is reduced. If this proves costly, an alternative is to broadcast the global sparse set to all shards and apply a per-shard mask before decoding. This restores cross-shard competition at the price of $S\times$ more indexed gathers (costly in JAX), but encoder/other costs may still dominate end-to-end time.

Approximate TopK We use JAX's approximate TopK with a recall parameter $r \in [0, 1]$, which returns a set whose expected overlap with the true TopK is $\geq r$. Values $r \in [0.75, 0.975]$ worked well. We note a minor implementation issue in the reference formula for the internal candidate size, which slightly overestimates it, leading to higher-than-requested recall and modestly higher runtime; this does not affect results materially.