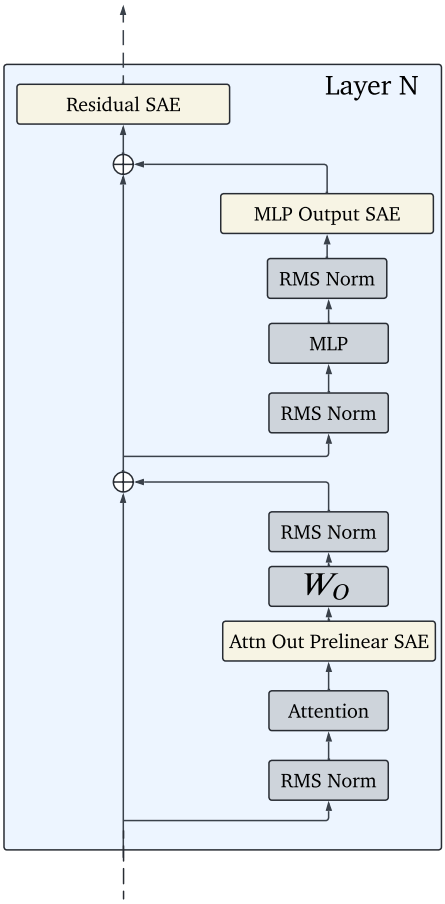

Figure 1 | Locations of sparse autoencoders inside a transformer block of Gemma 2. Note that Gemma 2 has RMS Norm at the start and end of each attention and MLP block.

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 linear warmup for the sparsity coefficient from 0 to λ over the first 10,000 training steps.

During training, we parameterise 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. See Appendix A for details.

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 initalized 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.

3.3. Infrastructure

3.3.1. Accelerators

Topology We train most of our SAEs using TPUv3 in a 4x2 configuration. Some SAEs, especially the most wide ones, were trained using TPUv5p in either a 2x2x1 or 2x2x4 configuration.

Sharding We train SAEs with 16.4K latents with maximum amount of data parallelism, while using maximal amounts of tensor parallelism using Megatron sharding (Shoeybi et al., 2020) for all other configurations. We find that as one goes to small SAEs and correspondingly small update step time, the time spent on host-to-device (H2D) transfers outgrows the time spent on the update step, favoring data sharding. For larger SAEs on the other hand, larger batch sizes enable higher arithmetic intensity by reducing transfers between HBM and VMEM of the TPU. Furthermore, the specific architecture of SAEs means that when using Megatron sharding, device-to-device (D2D) communication is minimal, while data parallelism causes a costly all-reduce of the full gradients. Thus we recommend choosing the smallest degree of data sharding such that the H2D transfer takes slightly less time than the update step.

As an example, with proper step time optimization this enables us to process one batch for a

Gemma 2 Model	SAE Width	Attention	MLP	Residual	# Tokens
2.6B PT	$2^{14} \approx 16.4$K	All	All	All+	4B
(26 layers)	2^{15}	✗	✗	{12}	8B
	2^{16}	All	All	All	8B
	2^{17}	✗	✗	{12}	8B
	2^{18}	✗	✗	{12}	8B
	2^{19}	✗	✗	{12}	8B
	$2^{20} \approx 1$M	✗	✗	{5, 12, 19}	16B
9B PT	2^{14}	All	All	All	4B
(42 layers)	2^{15}	✗	✗	{20}	8B
	2^{16}	✗	✗	{20}	8B
	2^{17}	All	All	All	8B
	2^{18}	✗	✗	{20}	8B
	2^{19}	✗	✗	{20}	8B
	2^{20}	✗	✗	{9, 20, 31}	16B
27B PT (46 layers)	2^{17}	✗	✗	{10, 22, 34}	8B
9B IT	2^{14}	✗	✗	{9, 20, 31}	4B
(42 layers)	2^{17}	✗	✗	{9, 20, 31}	8B

Table 1 | Overview of the SAEs that were trained for which sites and layers. For each model, width, site and layer, we release multiple SAEs with differing levels of sparsity (L0).
All+: We also train one suite of transcoders on the MLP sublayers on Gemma 2.6B PT (Appendix B).

131K-width SAE in 45ms on 8 TPUv3 chips, i.e. a model FLOP utilization (MFU) of about 50.8%.

3.3.2. Data Pipeline

Disk storage We store all collected activations on hard drives as raw bytes in shards of 10-20GiB. We use 32-bit precision in all our experiments. This means that storing 8B worth of activations for a single site and layer takes about 100TiB for Gemma 2 9B, or about 17PiB for all sites and layers of both Gemma 2 2B and 9B. The total amount is somewhat higher still, as we train some SAEs for 16B tokens and also train some SAEs on Gemma 2 27B, as well as having a generous buffer of additional tokens. While this is a significant amount of disk space, it is still cheaper than regenerating the data every time one wishes to train an SAE on it. Concretely, in our calculations we find that storing activations for 10-100 days is typically at least an order of magnitude cheaper than regenerating them one additional time. The exact numbers depend on the model used and the specifics of the infrastructure, but we expect this relationship to hold true in general. If there is a hard limit on the amount of disk space available, however, or if fast disk I/O can not be provided (see next paragraph), then this will favor on-the-fly generation instead. This would also be the case if the exact hyperparameter combinations were known in advance. In practice, we find it advantageous for research iteration speed to be able to sweep sparsity independently from other hyperparameters and to retrain SAEs at will.

Disk reads Since SAEs are very shallow models with short training step times and we train them on activation vectors rather than integer-valued tokens, training them requires high data throughput. For instance, to train a single SAE on Gemma 2 9B without being bottlenecked by data loading requires more than 1 GiB/s of disk read speed. This demand is further amplified when training multiple SAEs on the same site and layer, e.g. with different sparsity coefficients, or while conducting hyperparameter tuning.

To overcome this bottleneck we implement a shared server system, enabling us to amortize disk reads for a single site and layer combination:

- **Shared data buffer:** Multiple training jobs

share access to a single server. The server maintains a buffer containing a predefined number of data batches. Trainers request these batches from the servers as needed.
- **Distributed disk reads:** To enable parallel disk reads, we deploy multiple servers for each site and layer, with each server exclusively responsible for a contiguous slice of the data.
- **Dynamic data fetching:** As trainers request batches, the server continually fetches new data from the dataset, replacing the oldest data within their buffer.
- **Handling speed differences:** To accommodate variations in trainer speeds caused by factors like preemption, crashes and different SAE widths, trainers keep track of the batches they have already processed. If a trainer lags behind, the servers can loop through the dataset again, providing the missed batches. Note that different training speeds result in different trainers not seeing the same data or necessarily in the same order. In practice we found this trade-off well worth the efficiency gains.

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 1.

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 sparsity coefficient λ. 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, we use two metrics:

- Our primary metric is delta LM loss, the increase in the cross-entropy loss experienced by the LM when we splice the SAE into the LM's forward pass.
- As a secondary metric, we also use fraction of variance unexplained (FVU) – also called the normalized loss (Gao et al., 2024) – as a measure of reconstruction fidelity. This is the 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 reconstruction 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 Fig. 2 compares the sparsity-fidelity trade-off for SAEs in the middle of each Gemma model. For the full results see Appendix C. Delta loss is consistently higher for residual stream SAEs compared to MLP and attention SAEs, whereas FVU (Fig. 13) is roughly comparable across sites. We conjecture this is because even small errors in reconstructing the residual stream can have a significant impact on LM loss, whereas relatively large errors in reconstructing a single MLP or attention sub-layer's output have a limited impact on the LM loss.[7]

[7]The residual stream is the bottleneck by which the previous layers communicate with all later layers. Any given MLP or attention layer adds to the residual stream, and is typically only a small fraction of the residual, so even a large error in the layer is a small error to the residual stream and so to the rest of the network's processing. On the other hand, a large error to the residual stream itself will significantly harm loss. For the same reason, mean ablating the residual stream has far higher impact on the loss than mean ablating a single layer.