Compute Optimal Inference and Provable Amortisation Gap in Sparse Autoencoders

Charles O'Neill [1] **Alim Gumran** [2] **David Klindt** [3]

Abstract

A recent line of work has shown promise in using sparse autoencoders (SAEs) to uncover interpretable features in neural network representations. However, the simple linear-nonlinear encoding mechanism in SAEs limits their ability to perform accurate sparse inference. Using compressed sensing theory, we prove that an SAE encoder is inherently insufficient for accurate sparse inference, even in solvable cases. We then decouple encoding and decoding processes to empirically explore conditions where more sophisticated sparse inference methods outperform traditional SAE encoders. Our results reveal substantial performance gains with minimal compute increases in correct inference of sparse codes. We demonstrate this generalises to SAEs applied to large language models, where more expressive encoders achieve greater interpretability. This work opens new avenues for understanding neural network representations and analysing large language model activations.

1. Introduction

Understanding the inner workings of neural networks has become a critical task since these models are increasingly employed in high-stakes decision-making scenarios (Fan et al., 2021; Shahroudnejad, 2021; Räuker et al., 2023). As the complexity and scale of neural networks continue to grow, so does the importance of developing robust methods for interpreting their internal representations. This paper compares sparse autoencoders (SAEs) and sparse coding techniques, aiming to advance our ability to extract interpretable features from neural network activations.

Recent work has investigated the "superposition hypothesis" (Elhage et al., 2022), which posits that neural networks represent interpretable features in a linear manner using non-orthogonal directions in their latent spaces. Building on this idea, researchers have shown that individual features can be recovered from these superposed representations using sparse autoencoders (Bricken et al., 2023; Cunningham et al., 2023). These models learn sparse and overcomplete representations of neural activations, with the resulting sparse codes often proving to be more interpretable than the original dense representations (Cunningham et al., 2023; Elhage et al., 2022; Gao et al., 2024).

The mathematical foundation of SAEs aligns closely with that of sparse coding. Both approaches assume that a large number of sparse codes are linearly projected into a lower-dimensional space, forming the neural representation. However, while sparse coding typically involves solving an optimisation problem for each input, SAEs learn an efficient encoding function through gradient descent, potentially sacrificing optimal sparsity for computational efficiency. This trade-off introduces what statistical inference literature calls the "amortisation gap" – the disparity between the best sparse code predicted by an SAE encoder and the optimal sparse codes that an unconstrained sparse inference algorithm might produce (Marino et al., 2018).

In this paper, we explore this amortisation gap and investigate whether more sophisticated sparse inference methods can outperform traditional SAE encoders. Our key contribution is decoupling the encoding and decoding processes, allowing for a comparison of various sparse encoding strategies. We evaluate four types of encoding methods on synthetic datasets with known ground-truth features. We evaluate these methods on two dimensions: alignment with true underlying sparse features and inference of the correct sparse codes, while accounting for computational costs during both training and inference. To demonstrate real-world applicability, we also train models on GPT-2 activations (Radford et al., 2019), showing that more complex methods such as MLPs can yield *more* interpretable features than SAEs in large language models.

[1]Australian National University [2]Nazarbayev University [3]Cold Spring Harbor Laboratory. Correspondence to: Charles O'Neill <charles.oneill@anu.edu.au>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

2. Background and Related Work

2.1. Sparse Neural Representations

Sparse representations in neural networks specifically refer to activation patterns where only a small subset of neurons are active for any given input (Olshausen & Field, 1996). These representations have gained attention due to their potential for improved interpretability and efficiency (Lee et al., 2007). Sparse autoencoders (SAEs) are neural network architectures designed to learn sparse representations of input data (Ng et al., 2011; Makhzani & Frey, 2013). An SAE consists of an encoder that maps input data to a sparse latent space and a decoder that reconstructs the input from this latent representation. Sparse coding, on the other hand, is a technique that aims to represent input data as a sparse linear combination of basis vectors (Olshausen & Field, 1997). The objective of sparse coding is to find both the optimal basis (dictionary) and the sparse coefficients that minimise reconstruction error while maintaining sparsity. While both SAEs and sparse coding seek to find sparse representations, they differ in their approach. SAEs learn an efficient encoding function through gradient descent, allowing for fast inference but potentially sacrificing optimal sparsity. Sparse coding, in contrast, solves an optimisation problem for each input, potentially achieving better sparsity at the cost of increased computational complexity during inference.

2.2. Superposition in Neural Representations

The superposition hypothesis suggests that neural networks can represent more features than they have dimensions, particularly when these features are sparse (Elhage et al., 2022). Features are often defined as interpretable properties of the input that a sufficiently large neural network would reliably dedicate a neuron to representing (Olah et al., 2020). Formally, let us consider a neural representation $y \in \mathbb{R}^M$ and a set of N features, where typically $N > M$. In a linear representation framework, each feature f_i is associated with a direction $w_i \in \mathbb{R}^M$. The presence of multiple features is represented by $y = \sum_{i=1}^{N} x_i w_i$ where $x_i \in \mathbb{R}$ represents the activation or intensity of feature i.

In an M-dimensional vector space, only M orthogonal vectors can fit. However, the Johnson-Lindenstrauss Lemma states that if we permit small deviations from orthogonality, we can fit exponentially more vectors into that space. More formally, for any set of N points in a high-dimensional space, there exists a linear map to a lower-dimensional space of $\mathcal{O}(\log N/\epsilon^2)$ dimensions that preserves pairwise distances up to a factor of $(1 \pm \epsilon)$. This lemma supports the hypothesis that LLMs might be leveraging a similar principle in superposition, representing many more features than dimensions by allowing small deviations from orthogonality.

Superposition occurs when the matrix $W = [w_1, ..., w_N] \in \mathbb{R}^{M \times N}$ has more columns than rows (i.e., $N > M$), making $W^T W$ non-invertible. Superposition relies on the sparsity of feature activations. Let $s = ||x||_0$ be the number of non-zero elements in $x = [x_1, ..., x_N]^T$. When $s \ll N$, the model can tolerate some level of interference between features, as the probability of many features being active simultaneously (and thus interfering) is low.

2.3. Compressed Sensing and Sparse Coding

Compressed sensing theory provides a framework for understanding how sparse signals can be recovered from lower-dimensional measurements (Donoho, 2006). This theory suggests that under certain conditions, we can perfectly recover a sparse signal from fewer measurements than traditionally required by the Nyquist-Shannon sampling theorem. Let $s \in \mathbb{R}^N$ be a sparse signal with at most K non-zero components. If we make M linear measurements of this signal, represented as $y = Ws$ where $W \in \mathbb{R}^{M \times N}$, compressed sensing theory states that we can recover s from y with high probability if:

$$M > \mathcal{O}\left(K \log\left(\frac{N}{K}\right)\right) \tag{1}$$

This result holds under certain assumptions about the measurement matrix W, such as the Restricted Isometry Property (RIP) (Candes, 2008).[1] Sparse coding is one approach to recovering such sparse representations. The objective function for sparse coding (Olshausen & Field, 1996) is:

$$\mathcal{L}(D, \alpha) := \sum_{i}^{N} ||x_i - D\alpha_i||_2^2 + \lambda ||\alpha_i||_1 \tag{2}$$

where $D \in \mathbb{R}^{K \times M}$ is the dictionary, $\alpha_i \in \mathbb{R}^M$ are the sparse codes for data point $x_i \in \mathbb{R}^K$, and λ is a hyperparameter controlling sparsity. Optimisation of this objective typically alternates between two steps. First is sparse inference: $\min_{\alpha} \sum_i^N ||x_i - D\alpha_i||_2^2 + \lambda ||\alpha_i||_1$. Then dictionary learning: $\min_{D} \sum_i^N ||x_i - D\alpha_i||_2^2 \quad \text{s.t.} \quad \forall i \in 1, ..., M : |D{:}, i| = 1$. These techniques allow extraction of interpretable, sparse representations from high-dimensional neural data.

2.4. Sparse Autoencoders

Sparse autoencoders (SAEs) offer an alternative approach to extracting sparse representations, using amortised inference instead of the iterative optimisation used in sparse coding. SAEs learn to reconstruct inputs using a sparse set of features in a higher-dimensional space, *potentially disentangling superposed features* (Elhage et al., 2022; Olshausen

[1] This property is readily satisfied by many common measurement matrices, including random Gaussian and Bernoulli matrices (Baraniuk et al., 2008).

& Field, 1997). The architecture of an SAE consists of an encoder network that maps the input to a hidden, sparse representation of latent coefficients, and a decoder network that reconstructs the input as a linear combination of vectors, with the coefficients defined by the sparse representation. Let $x_i \in \mathbb{R}^K$ be an input vector (as in our sparse coding formulation), and $\alpha_i \in \mathbb{R}^M$ be the hidden representation (analogous to the sparse codes in sparse coding), where typically $M > K$. The encoder and decoder functions are:

$$\text{Encoder}: \quad \alpha_i = f_\theta(x_i) = \sigma(W_e x_i + b_e) \tag{3}$$

$$\text{Decoder}: \quad \hat{x}_i = g_\phi(\alpha_i) = W_d \alpha_i + b_d \tag{4}$$

where $W_e \in \mathbb{R}^{M \times K}$ and $W_d \in \mathbb{R}^{K \times M}$ are the encoding and decoding weight matrices, $b_e \in \mathbb{R}^M$ and $b_d \in \mathbb{R}^K$ are bias vectors, and $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU). The parameters $\theta = W_e, b_e$ and $\phi = W_d, b_d$ are learned during training.

The training objective of an SAE maintains the same form as Equation 2, minimising reconstruction error while promoting sparsity. However, SAEs differ from sparse coding in how they perform inference. In sparse coding, finding the codes α_i for a new input requires solving an iterative optimisation problem that alternates between updating the codes and the dictionary. In contrast, SAEs learn an encoder function f_θ during training that directly computes sparse codes in a single forward pass. This amortised inference trades off some precision in the optimisation for computational savings at inference time – while sparse coding must solve a new optimisation problem for each input, an SAE can instantly generate codes through its learned encoder.

SAE with Inference-Time Optimisation (SAE+ITO) (SAE+ITO) is an extension of the standard SAE approach that combines the learned dictionary from SAEs with inference-time optimisation for sparse code inference (Nanda et al., 2024). In this method, the decoder weights W_d learned during SAE training are retained, but the encoder function f_θ is replaced with an optimisation procedure at inference time. For each input x_i, SAE+ITO solves the optimisation problem outlined in Equation 2, except only optimising the latent codes with the decoder weights fixed.

This formulation allows for potentially more accurate sparse codes by directly minimising reconstruction error, rather than relying on the learned encoder approximation, despite incurring higher computational costs at inference time. The optimisation problem can be solved using algorithms such as matching pursuit (Blumensath & Davies, 2008) and gradient pursuit (Nanda et al., 2024).

2.5. Applications in Neural Network Models

Sparse autoencoders (SAEs) have emerged as a promising tool for enhancing the interpretability of large language models (LLMs) by extracting interpretable features from their dense representations. Early work by Cunningham et al. (2023) and Bricken et al. (2023) demonstrated the potential of sparse dictionary learning to disentangle features, lifting them out of superposition in transformer MLPs. This approach was extended to attention heads by Kissane et al. (2024), who scaled it to GPT-2 (Radford et al., 2019). These studies have shown that SAEs can extract highly abstract, multilingual, and multimodal features from LLMs, including potentially safety-relevant features related to deception, bias, and dangerous content (Templeton, 2024). In vision models, Gorton (2024) and Klindt et al. (2023) trained SAEs on convolutional neural network activations. The latter found that K-means (which is equivalent to one-hot sparse coding) outperformed SAEs (Fig.12) in quantitative interpretability metrics (Zimmermann et al., 2024).

The scaling of SAEs to larger models has been a focus of recent research, with significant progress made in applying them to state-of-the-art LLMs. Gao et al. (2024) proposed using k-sparse autoencoders (Makhzani & Frey, 2013) to simplify tuning and improve the reconstruction-sparsity frontier, demonstrating clean scaling laws with respect to autoencoder size and sparsity. They successfully trained a 16 million latent autoencoder on GPT-4 activations. Similarly, Templeton (2024) reported extracting high-quality features from Claude 3 Sonnet, while Lieberum et al. (2024) released a comprehensive suite of SAEs trained on all layers of Gemma 2 models. These advancements underscore the importance of developing efficient and accurate SAE techniques, especially as applications to larger models become more prevalent. The growing body of work on SAEs in LLMs suggests that they may play a crucial role in future interpretability research.

3. Methods

This section outlines our approach to comparing sparse encoding strategies. We begin by presenting a theoretical foundation for the suboptimality of sparse autoencoders (SAEs), followed by our data generation process, encoding schemes, evaluation metrics, and experimental scenarios.

3.1. Theory: Provable Suboptimality of SAEs

Theorem 3.1 (SAE Amortisation Gap)**.** *Let $K \geq 2$ and P_K be a sparse distribution over $\mathbb{R}^N$, i.e., $\forall s \in \mathbb{R}^N$: $s \in supp(P_K) \iff \|s\|_0 \leq K$. This means that any sample has at most K non-zero entries or, equivalently, the support of P_K is a union over K dimensional subspaces. The sources are linearly projected into an M-dimensional space, satisfying the restricted isometry property, where $K \log \frac{N}{K} \leq M < N$. A sparse autoencoder (SAE) with a linear-nonlinear (L-NL) encoder must have a non-zero amortisation gap.*

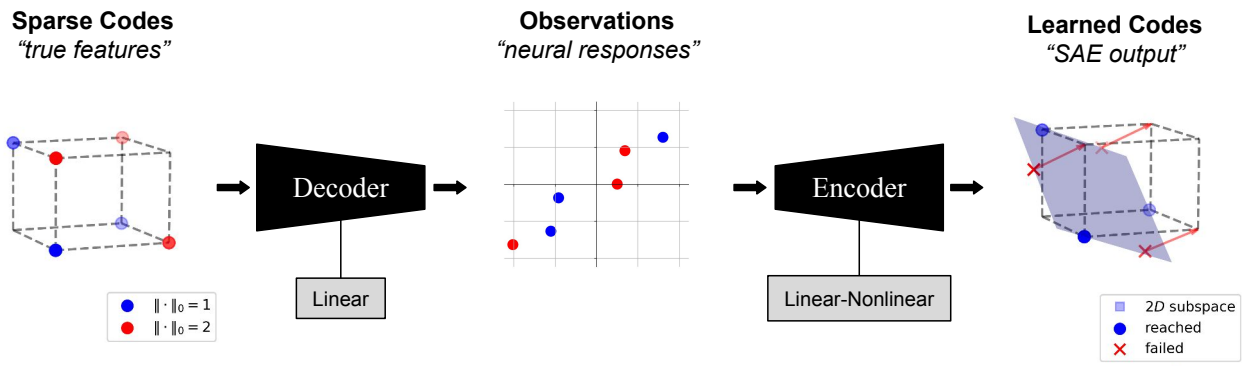


Figure 1: **Illustration of SAE Amortisation Gap. Left**, shows sparse sources in an $N = 3$ dimensional space with at most $\|s\| \leq K = 2$ non-zero entries. Both blue and red points are valid sources, by contrast, the top right corner $s = (1, 1, 1)$ is not. **Middle**, shows the sources as they are linearly *decoded* into observation space. This is, in most applications, the activation space of a neural network that we are trying to lift out of superposition. **Right**, shows how using a linear-nonlinear encoder, a SAE is tasked to project the points back onto their correct positions. This is not possible, because the pre-activations are at most $M = 2$ dimensional (see proof in Appendix A).

The complete proof of Theorem 3.1 is provided in Appendix A. The theorem considers a setting where sparse signals $s \in \mathbb{R}^N$ with at most K non-zero entries are projected into an M-dimensional space ($M < N$). Compressed sensing theory guarantees that unique recovery of these sparse signals is possible when $M \geq K \log(N/K)$, up to sign ambiguities (Donoho, 2006). However, we prove that SAEs fail to achieve this optimal recovery, resulting in a non-zero amortisation gap. The core of this limitation lies in the architectural constraints of the SAE's encoder. The linear-nonlinear (L-NL) structure of the encoder lacks the computational complexity required to fully recover the high-dimensional (N) sparse representation from its lower-dimensional (M) projection. Figure 1 illustrates this concept geometrically.

For completeness, we compare our amortisation-gap argument with previous local and distribution-specific recovery results (e.g., (Rangamani et al., 2018; Nguyen et al., 2019)) in Appendix B. In particular, we clarify why local convergence guarantees for ReLU-based autoencoders do not contradict our global impossibility result when addressing all K-sparse signals in $\mathbb{R}^N$.

3.2. Synthetic data

To evaluate our sparse encoding strategies, we generate synthetic datasets with known ground-truth latent representations and dictionary vectors. We first construct a dictionary matrix $\mathbf{D} \in \mathbb{R}^{M \times N}$, where each column represents a dictionary element. We then generate latent representations $\mathbf{s}_i \in \mathbb{R}^N$ with exactly K non-zero entries ($K \ll N$), drawn from a standard normal distribution. This allows us to create observed data points as $\mathbf{x}_i = \mathbf{D}\mathbf{s}_i$. This process yields a dataset $\mathcal{D} = (\mathbf{x}_i, \mathbf{s}_i)_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^M$ and $\mathbf{s}_i \in \mathbb{R}^N$. In Appendix D, we explore an alternative data generation process that incorporates a Zipf distribution over feature activations, motivated by recent observations that latent representations in large models often follow heavy-tailed distributions (Engels et al., 2024; Park et al., 2024)

3.3. Sparse Encoding Schemes

We compare four sparse encoding strategies:

1. **Sparse Autoencoder (SAE)**: $f(x) := \sigma(Wx)$, where σ is a nonlinear activation function.
2. **Multilayer Perceptron (MLP)**: $f(x) := \sigma(W_n\sigma(W_{n-1}\dots\sigma(W_1x)))$, with the same decoder as the SAE.
3. **Sparse Coding (SC)**: $f(x) = \text{argmin}_{\hat{s}}|x - D\hat{s}|_2^2 + \lambda||\hat{s}||_1$, solved iteratively with $s_{t+1} = s_t + \eta\nabla\mathcal{L}$, where $\mathcal{L}$ is the MSE loss with L1 penalty.
4. **SAE with Inference-Time Optimisation (SAE+ITO)**: Uses the learned SAE dictionary, optimising sparse coefficients at inference time.

For all methods, we normalise the column vectors of the decoder matrix to have unit norm, preventing the decoder from reducing the sparsity loss $||\hat{s}||_1$ by increasing feature vector magnitudes.

3.4. Measuring the quality of the encoder and decoder

For any given x, how do we measure the quality of (1) the encoding (i.e. the sparse coefficients); and (2) the decoding (i.e. the actual reconstruction, given the coefficients)?

We employ the Mean Correlation Coefficient (MCC) to evaluate both encoder and dictionary quality:

$$\text{MCC} = \frac{1}{d} \sum_{(i,j)\in M} |c_{ij}| \tag{5}$$

where c_{ij} is the Pearson correlation coefficient between the i-th true feature and the j-th learned feature, and M is the set of matched pairs determined by the Hungarian algorithm (or a greedy approximation when dimensions differ). This metric quantifies alignment between learned sparse coefficients and true underlying sparse features (encoder quality), and learned dictionary vectors and true dictionary vectors (dictionary quality).

3.5. Disentangling Dictionary Learning and Sparse Inference

Our study decomposes the sparse coding problem into two interrelated tasks: dictionary learning and sparse inference. Dictionary learning involves finding an appropriate sparse dictionary $D \in \mathbb{R}^{M\times N}$ from data, while sparse inference focuses on reconstructing a signal $x \in \mathbb{R}^M$ using a sparse combination of dictionary elements, solving for $s \in \mathbb{R}^N$ in $x \approx Ds$ where s is sparse. These tasks are intrinsically linked: dictionary learning often involves sparse inference in its inner loop, while sparse inference requires a dictionary.

Known Sparse Codes. In this scenario, we assume knowledge of the true sparse codes s^* and focus solely on the encoder's ability to predict these latents, effectively reducing the problem to latent regression. We define the objective as minimising $\mathcal{L}(f(x), s^*) = 1 - \cos(f(x), s^*)$, where f is the encoding function and cos denotes cosine similarity.[2] In this setting, only the SAE encoder and MLP are applicable, as they directly learn mappings from input to latent space. The SAE encoder learns an amortised inference function, while the MLP learns a similar but more complex mapping. Conversely, SAE+ITO and sparse coding are not suitable for this task. SAE+ITO focuses on optimising reconstruction using a fixed dictionary, which is irrelevant when true latents are known. Similarly, sparse coding alternates between latent and dictionary optimisation, which reduces to encoder training when the dictionary is disregarded.

Known Dictionary. When the true dictionary D^* is known, we focus on optimising the encoder or inference process while keeping the dictionary fixed. This scenario is applicable to SAE, MLP, and SAE+ITO methods. For SAE and MLP, we optimise $\min_\theta \mathbb{E}_x[|x - D^* f_\theta(x)|_2^2]$, where f_θ represents the encoder function with parameters θ. SAE+ITO, in contrast, performs gradient-based optimisation at inference time: $\min_s |x - D^* s|_2^2 + \lambda|z|_1$ for each input x, incurring zero training FLOPs but higher inference-time costs. This differs from SAE and MLP by directly optimising latent coefficients rather than learning an encoding function. Sparse coding is not applicable in this scenario, as it reduces to SAE+ITO when the dictionary is known.

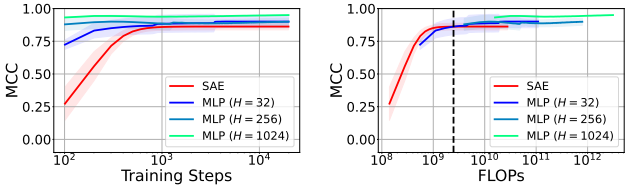
[2]We use cosine similarity rather than MSE loss in this setting because we found training to be more stable.

Unknown Sparse Codes and Dictionary. This scenario represents the standard setup in sparse coding, where both the sparse codes s and the dictionary D are unknown and must be learned simultaneously. All four methods — SAE, MLP, SAE+ITO, and sparse coding — are applicable here. SAE and MLP learn both an encoder function $f_\theta(x)$ and a dictionary D simultaneously. SAE+ITO and sparse coding learn a dictionary during training and optimises latents at inference time.

4. Synthetic Sparse Inference Experiments

We present the results of our experiments comparing different sparse encoding strategies across various scenarios. All experiments were conducted using synthetic data with $N = 16$ sparse sources, $M = 8$ measurements, and $K = 3$ active components per timestep, unless otherwise specified (more settings in Sec. 4.4, with larger values in App. B and App. C).

4.1. Known Sparse Codes



(a) MCC vs. training steps (b) MCC vs. total FLOPs

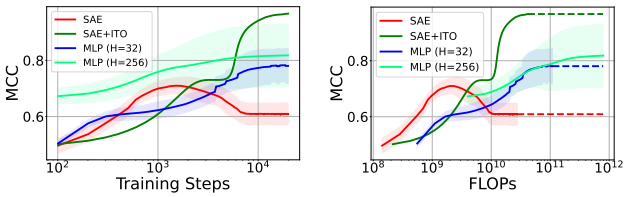
Figure 2: Performance comparison of SAE and MLPs in predicting known latent representations. The black dashed line in (b) indicates the average FLOPs at which MLPs surpass SAE performance.

We first compare the performance of sparse autoencoders (SAEs) and multilayer perceptrons (MLPs) in predicting known latent representations. Figure 2 illustrates the performance of SAEs and MLPs with varying hidden layer widths. MLPs consistently outperform SAEs in terms of Mean Correlation Coefficient (MCC), with wider hidden layers achieving higher performance (Figure 2a). The MLP with $H = 1024$ reaches an MCC approximately 0.1 higher than the SAE at convergence. While MLPs converge faster in terms of training steps, this comes at the cost of increased computational complexity (Figure 2b). All MLPs surpass the SAE's plateau performance at approximately the same total FLOPs, suggesting a consistent computational thresh-

old beyond which MLPs become more effective, regardless of hidden layer width.

We also validated our findings at larger scales that better match real-world applications ($N = 1000, M = 200, K = 20$, and $500,000$ data points), finding that the amortisation gap becomes even more pronounced (see Appendix C).

4.2. Known Dictionary



(a) MCC vs. training steps (b) MCC vs. total FLOPs

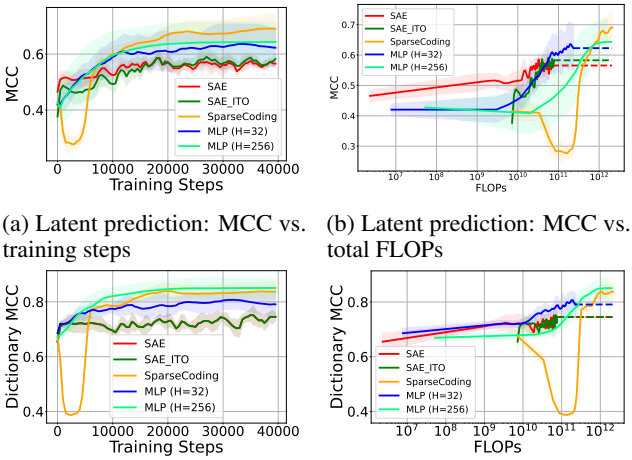
Figure 3: Performance comparison of SAE, SAE with inference-time optimisation (SAE+ITO), and MLPs in predicting latent representations with a known dictionary. Dashed lines in (b) indicate extrapolated performance beyond the measured range.

Next, we examine the performance of different encoding strategies when the true dictionary D^* is known. Figure 3 shows the performance of SAE, SAE+ITO, and MLPs. MLPs consistently outperform the standard SAE, achieving an MCC nearly 10% higher at convergence (Figure 3a). Both MLP configurations ($H = 32$ and $H = 256$) converge to similar performance levels, with the wider network showing faster initial progress. When plotted against total FLOPs, the MLP curves overlap, suggesting a consistent computational cost-to-performance ratio across different hidden layer widths (Figure 3b). SAE+ITO initialised with SAE latents exhibits distinct, stepwise improvements throughout training, ultimately achieving the highest MCC.

4.3. Unknown Sparse Codes and Dictionary

Finally, we evaluate all four methods when both latent representations and dictionary are unknown. We use a dataset of 2048 samples, evenly split between training and testing sets, and conduct 5 independent runs of $100,000$ steps.

Figures 4 illustrates the performance in latent prediction and dictionary learning, respectively. For latent prediction, SAE, SAE+ITO, and MLPs converge to similar MCC, with MLPs showing a slight advantage. Sparse coding demonstrates superior performance, achieving an MCC over 10% higher than other methods, despite an initial decrease in performance. Sparse coding reaches this higher performance while using comparable FLOPs to the MLP with $H = 256$. For dictionary learning, both MLPs and sparse coding outperform SAE by a margin of approximately 10%. Sparse coding again exhibits an initial decrease in dictionary MCC before surpassing other methods.



(a) Latent prediction: MCC vs. training steps (b) Latent prediction: MCC vs. total FLOPs

(c) Dictionary learning: MCC vs. training steps (d) Dictionary learning: MCC vs. total FLOPs

Figure 4: Dictionary learning performance comparison when both s^* and D^* are unknown.

4.4. Performance Across Varying Data Regimes

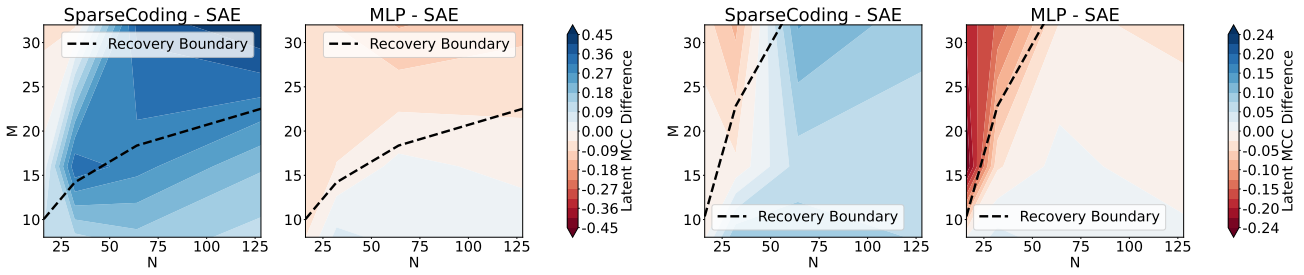
To understand how performance varies with changes in data characteristics, we trained models under varying N, M, and K, holding other hyperparameters constant.

Figure 5 shows the difference in final latent MCC between methods. Sparse coding outperforms SAE in essentially all data-generation regimes, for both $K = 3$ and $K = 9$. MLP and SAE perform roughly equivalently, with MLP slightly better as M (number of measurements) increases. The performance advantage of sparse coding is more pronounced in regimes where compressed sensing theory predicts recoverability (above and to the left of the black dashed line).

Sparsity-Performance Trade-off We also investigated the trade-off between sparsity and performance for each method in Figure 6. Sparse coding achieves slightly lower reconstruction error for each L0 level, barring some very small active latents. Sparse coding shows a Pareto improvement at each L0 level in terms of MCC, even with very small active latents. The improvement is more evident when plotting against L1 rather than L0, as L1 accounts for the magnitude of non-zero values. The presence of very small non-zero latents in sparse coding motivates the exploration of top-k sparse coding, detailed in Appendix H.2.

5. Interpretability of Sparse Coding Schemes

A common concern about more powerful encoding approaches is that they might learn unnatural features that



(a) $K = 3$ (b) $K = 9$

Figure 5: Difference in final latent MCC between methods across varying N and M, for $K = 3$ and $K = 9$. **Left:** Sparse coding vs. SAE. **Right:** MLP vs. SAE. The black dashed line indicates the theoretical recovery boundary.

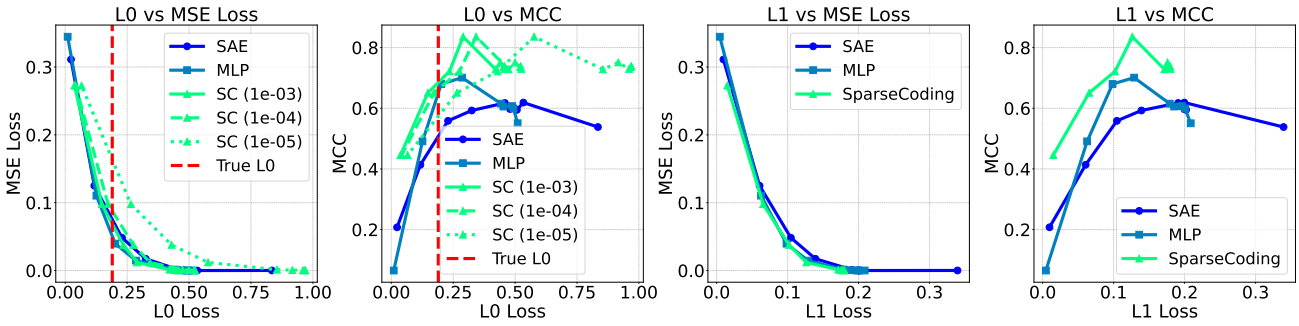


Figure 6: Pareto curves showing sparsity (L0 or L1 loss) against performance (MSE loss or latent MCC) for models trained with varying L1 penalty coefficients λ. The red dashed line in the top row shows the true L0 of the sparse sources. Multiple thresholds for active features are shown for sparse coding due to the presence of very small non-zero values.

are not interpretable. To investigate the interpretability of more complex encoding techniques, we trained three distinct methods on 406 million tokens from OpenWebText: a sparse autoencoder with a single linear encoder layer and ReLU activation, a multilayer perceptron encoder with one hidden layer of width 8448, and a locally competitive algorithm following the approach of Olshausen & Field (1997) and Blumensath & Davies (2008). Each method learned an overcomplete dictionary of size $16,896$ for the residual-stream pre-activations at Layer 9 of GPT-2 Small (Radford et al., 2019), which have dimension 768.

All methods were trained using Adam with a learning rate of $3 \cdot 10^{-4}$ and an L_1 penalty of $1 \cdot 10^{-4}$. Following Bricken et al. (2023) and Cunningham et al. (2023), we resampled dead neurons every $15,000$ steps by setting columns with no activity to new random directions. The final results across methods were: the SAE achieved a normalised MSE of 0.061 with a mean L_0 of 35.66 and 11% dead neurons, while the MLP reached a normalised MSE of 0.055 with a mean L_0 of 31.13 and 22% dead neurons. The LCA approach, with 100 gradient-based sparse-inference steps per batch, achieved a normalised MSE of 0.070. While technically none of the LCA codes were exactly zero, most were extremely small, and thresholding values below 10^{-5} yielded an effective L_0 of approximately 18.56. Notably, the LCA dictionary maintained no strictly dead columns.

To assess the interpretability of the learned features, we randomly selected 500 features from each method and employed an automated interpretability classification approach using GPT-4o (full details in Appendix J). For each feature, we identified its top 10 most highly activating tokens in our 13.1 million-token test set and computed logit effects through the path expansion $W_U \cdot f$, where W_U represents the model's unembedding matrix and f denotes the feature vector. We provided both the activating examples and the top and bottom 10 tokens by logit effect to GPT-4o, which generated a concise explanation of the feature's function. To validate these interpretations, we presented them to a second instance of GPT-4o along with at least five new activating examples and five non-activating examples, labelling the activating tokens. The model predicted which examples should activate the feature based on the first instance's explanation, allowing us to compute an F1-score against the ground truth. This automated interpretability approach is considered standard in the literature and relies on a base prompt from Juang et al. (2024).

Figure 7 displays the distributions of F1-scores across the evaluated features. The results indicate that SAE and LCA

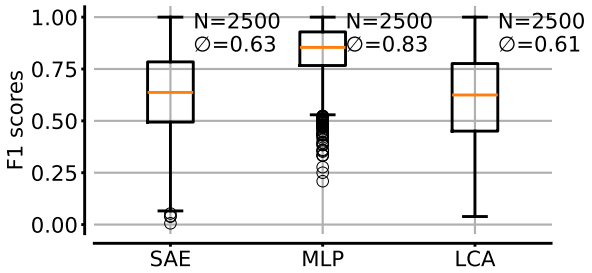


Figure 7: Distribution of F1 scores for feature interpretability across three methods (SAE, MLP, and LCA) trained on residual stream activations of Layer 9 in GPT-2. Each distribution represents 500 randomly selected features evaluated using GPT-4o for explanation generation and validation.

features demonstrate comparable interpretability, with median F1-scores around 0.6. Most notably, the MLP features achieve substantially higher interpretability scores, with a median F1-score of 0.83 and a tighter distribution. A Kruskal-Wallis test revealed significant differences between the methods ($H = 1856.33$, $p < 0.001$), and subsequent Dunn's tests with Bonferroni correction confirmed that both SAE and LCA were significantly less interpretable than MLP features ($p < 0.001$). See Appendix J for examples of feature interpretations.

6. Discussion

Our study provides theoretical and empirical evidence for an inherent amortisation gap in sparse autoencoders (SAEs) when applied to neural network interpretability. We prove that SAEs with linear-nonlinear encoders cannot achieve optimal sparse recovery in settings where such recovery is theoretically possible. This limitation is supported by experimental results showing that sparse coding, and sometimes MLPs, outperform SAEs across synthetic data scenarios. Our investigation of GPT-2 activations demonstrates that MLP-based features achieve higher interpretability scores than both SAE and LCA features. These findings refute the assumption that simpler encoders are necessary for maintaining interpretability. The results carry implications for neural network interpretability, suggesting that more sophisticated encoding techniques can improve feature extraction without compromising feature validity, though at increased cost.

The use of linear-nonlinear encoders in SAEs for language model interpretability stems from concerns that more powerful methods might extract features not used by the transformer (Bricken et al., 2023). This approach appears overly restrictive given the complexity of transformer layer representations, which emerge from multiple rounds of attention and feed-forward computations. The superior performance of MLPs suggests that matching the computational complexity of the underlying representations improves feature extraction. Better encoders aligns with recent work on inference-time optimisation (Nanda et al., 2024), and will be validated as we improve encoding evaluation (Makelov et al., 2024). Regardless, SAEs are sensitive to hyperparameters and fragile (Cunningham et al., 2023), so exploring more powerful encoders is warranted.

The computational cost of complex encoders must be evaluated against gains in feature extraction and interpretability. Projects like Gemma Scope (Lieberum et al., 2024) demonstrate substantial resource investment in feature extraction, suggesting that additional compute for improved representation quality may be justified. Complex encoders can maintain the linear decoder needed for downstream tasks such as steering while providing better features. Future work should systematically compare feature quality across encoder architectures and address non-zero-centered representations (Hobbhahn, 2023).

Limitations Our study has several limitations. Our LCA implementation was not optimised for the scale of experiments, requiring investigation of sparse coding methods, sparsity levels, optimisation iterations, and thresholding of near-zero activations. The gap between MLP and SAE/LCA interpretability scores warrants examination – while LCA's lower performance likely stems from suboptimal training, the MLP's superior interpretability relative to SAEs requires investigation. Our analysis also focused on scenarios with constant sparsity and uncorrelated channels, which may not capture real-world data complexity. Our synthetic data generation process did not account for varying feature importance described in Elhage et al. (2022)'s framework, although we did begin to explore this in Appendix D.

Future work should incorporate recent SAE variants like top-k SAEs (Makhzani & Frey, 2013; Gao et al., 2024) and JumpReLU SAEs (Rajamanoharan et al., 2024b) to measure the amortisation gap with modern architectures. Our SAE+ITO implementation did not use advanced techniques like matched pursuit, potentially underestimating its performance. The traditional dictionary learning approaches in Appendix H indicate room for improvement. Finally, we should explore sampling feature activations from different parts of the activation spectrum when doing automated interpretability, because features may exhibit different levels of specificity at different activation strengths, and examining only top activations could miss important collaborative behaviors between features and edge cases that help validate feature interpretations (Bricken et al., 2023). Addressing these limitations would advance understanding of sparse encoding strategies for complex neural representations.

Impact Statement

This work advances neural network interpretability by improving methods for extracting understandable features from complex AI systems, with primarily positive societal implications for building trustworthy AI in high-stakes applications. By demonstrating that more sophisticated encoding methods achieve superior interpretability while maintaining computational efficiency, this research could accelerate transparent AI development across domains like healthcare and autonomous systems, and democratise access to interpretability tools for smaller research groups. However, we acknowledge potential risks including misuse for adversarial attacks, exploitation of model vulnerabilities, and false confidence in model safety if users over-rely on feature interpretations. The computational improvements may also lower barriers for both beneficial AI safety research and potentially harmful applications. Overall, we believe the benefits of advancing interpretability research significantly outweigh the risks, as transparent and understandable AI systems are fundamental to responsible AI deployment.

References

Baraniuk, R., Davenport, M., DeVore, R., and Wakin, M. A simple proof of the restricted isometry property for random matrices. *Constructive approximation*, 28:253–263, 2008.

Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J., and Saunders, W. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)*, 2, 2023.

Blumensath, T. and Davies, M. E. Gradient pursuits. *IEEE Transactions on Signal Processing*, 56(6):2370–2382, 2008.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2, 2023.

Candes, E. J. The restricted isometry property and its implications for compressed sensing. *Comptes rendus. Mathematique*, 346(9-10):589–592, 2008.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Donoho, D. L. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. 2004.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

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

Fan, F.-L., Xiong, J., Li, M., and Wang, G. On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(6): 741–760, 2021.

Foote, A., Nanda, N., Kran, E., Konstas, I., and Barez, F. N2g: A scalable approach for quantifying interpretable neuron representation in llms. In *ICLR 2023 Workshop on Trustworthy and Reliable Large-Scale Machine Learning Models*.

Gao, L., la Tour, T. D., Tillman, H., Goh, G., Troll, R., Radford, A., Sutskever, I., Leike, J., and Wu, J. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Gorton, L. The missing curve detectors of inceptionv1: Applying sparse autoencoders to inceptionv1 early vision. *arXiv preprint arXiv:2406.03662*, 2024.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pp. 399–406, 2010.

Hobbhahn, M. More findings on memorization and double descent, 2023. URL https://www.alignmentforum.org/posts/KzwB4ovzrZ8DYWgpw/more-findings-on-memorization-and-double-descent. [Accessed 29-09-2024].

Juang, C., Paulo, G., Drori, J., and Belrose, N. Open source automated interpretability for sparse autoencoder features, 2024. URL https://blog.eleuther.ai/autointerp/. [Accessed 29-09-2024].

Kissane, C., Krzyzanowski, R., Bloom, J. I., Conmy, A., and Nanda, N. Interpreting attention layer outputs with sparse autoencoders. *arXiv preprint arXiv:2406.17759*, 2024.

Klindt, D., Sanborn, S., Acosta, F., Poitevin, F., and Miolane, N. Identifying interpretable visual features in artificial and biological neural systems. *arXiv preprint arXiv:2310.11431*, 2023.

Lee, H., Ekanadham, C., and Ng, A. Sparse deep belief net model for visual area v2. *Advances in neural information processing systems*, 20, 2007.

Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A., Shah, R., and Nanda, N. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2, 2024. URL https://arxiv.org/abs/2408.05147.

Makelov, A., Lange, G., and Nanda, N. Towards principled evaluations of sparse autoencoders for interpretability and control, 2024. URL https://arxiv.org/abs/2405.08366.

Makhzani, A. and Frey, B. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Marino, J., Yue, Y., and Mandt, S. Iterative amortized inference. In *International Conference on Machine Learning*, pp. 3403–3412. PMLR, 2018.

Nanda, N., Conmy, A., Smith, L., Rajamanoharan, S., Lieberum, T., Kramár, J., and Varma, V. Progress update from the gdm mech interp team, 2024. [Accessed 01-09-2024].

Ng, A. et al. Sparse autoencoder. *CS294A Lecture notes*, 72 (2011):1–19, 2011.

Nguyen, T. V., Wong, R. K., and Hegde, C. On the dynamics of gradient descent for autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2858–2867. PMLR, 2019.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Olshausen, B. A. and Field, D. J. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Park, K., Choe, Y. J., Jiang, Y., and Veitch, V. The geometry of categorical and hierarchical concepts in large language models. *arXiv preprint arXiv:2406.01506*, 2024.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. S. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers*, pp. 40–44. IEEE, 1993.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Rajamanoharan, S., Lieberum, T., Sonnerat, N., Conmy, A., Varma, V., Kramár, J., and Nanda, N. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024b.

Rangamani, A., Mukherjee, A., Basu, A., Arora, A., Ganapathi, T., Chin, S., and Tran, T. D. Sparse coding and autoencoders. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 36–40. IEEE, 2018.

Räuker, T., Ho, A., Casper, S., and Hadfield-Menell, D. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 ieee conference on secure and trustworthy machine learning (satml)*, pp. 464–483. IEEE, 2023.

Shahroudnejad, A. A survey on understanding, visualizations, and explanation of deep neural networks. *arXiv preprint arXiv:2102.01792*, 2021.

Taggart, G. Profilu: A nonlinearity for sparse autoencoders. In *AI Alignment Forum*, 2024.

Templeton, A. *Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet*. Anthropic, 2024.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Wright, B. and Sharkey, L. Addressing feature suppression in saes. In *AI Alignment Forum*, pp. 16, 2024.

Zimmermann, R. S., Klindt, D. A., and Brendel, W. Measuring mechanistic interpretability at scale without humans. volume 38, 2024.

Contents

A. Amortisation gap proof

Theorem A.1 (SAE Amortisation Gap)**.** *Let $K \geq 2$ and P_K be a sparse distribution over $\mathbb{R}^N$, i.e., $\forall s \in \mathbb{R}^N : s \in supp(P_K) \iff \|s\|_0 \leq K$. This means that any sample has at most K non-zero entries or, equivalently, the support of P_K is a union over K dimensional subspaces. The sources are linearly projected into an M-dimensional space, satisfying the restricted isometry property, where $K \log \frac{N}{K} \leq M < N$. A sparse autoencoder (SAE) with a linear-nonlinear (L-NL) encoder must have a non-zero amortisation gap.*

This setting is solvable according to compressed sensing theory (Donoho, 2006), meaning that it is possible to uniquely recover the true S up to sign flips – we cannot resolve the ambiguity between the sign of any code element and the corresponding row in the decoding matrix. If a SAE fails to achieve the same recovery, then there must be a non-zero amortisation gap, meaning that the SAE cannot solve the sparse inference problem of recovering all sparse sources from their M-dimensional projection. The problem is the low computational complexity of the L-NL encoder as we see by looking at its functional mapping. Essentially, the SAE is not able, not even after the nonlinear activation function, to recover the high dimensionality (N) of the data after a projection into a lower (M) dimensional space Figure 1.

Proof. Let $S = \text{diag}(s_{11}, ..., s_{NN})$ be a diagonal matrix with non-zero diagonal elements $s_{ii} \neq 0, \forall i \in \{1, ..., N\}$. Ever row s_i is a valid source signal because it has non-zero support under P_K since, $\|s_i\|_0 = 1 \leq K, \forall i \in \{1, ..., N\}$. Since the support of P_K includes all 1-sparse vectors (as $1 \leq K$), selecting S as the diagonal matrix of 1-sparse signals is without loss of generality. Let $W_d \in \mathbb{R}^{N \times M}$ be the unknown projection matrix from N down to M dimensions and $W_e \in \mathbb{R}^{M \times N}$ be the learned encoding matrix of the SAE. Define $W := W_d W_e \in \mathbb{R}^{N \times N}$ and

$$S' := SW \tag{6}$$

the pre-activation matrix from the encoder of the SAE. Since W_d projects down into M dimensions,

$$\text{rank}(W) = \text{rank}(W_d W_e) \leq M. \tag{7}$$

It follows that

$$\text{rank}(S') = \text{rank}(SW) \leq M. \tag{8}$$

As an intermediate results, we conclude that the pre-activations S' of the SAE encoder cannot recover the sources $S' \neq |S|$ since $\text{rank}(|S|) = N$, because S is a diagonal matrix.

The next step is to see whether the nonlinear activation function might help to map back to the sources. The SAE must learn an encoding matrix W_e such that

$$|S| = \max(0, SW_d W_e) = \max(0, SW) = \max(0, S') \tag{9}$$

where $\max(0, \cdot)$ is the ReLU activation function. Thus, for the SAE to correctly reconstruct the sparse signals up to sign flips, for any source code $\sigma \in \text{supp}(P_K)$, we require

$$(\sigma W)_i = \begin{cases} |\sigma_i| & \text{if } \sigma_i \neq 0 \\ \leq 0 & \text{otherwise} \end{cases} \tag{10}$$

specifically, S' must be non-positive off the diagonal and identical to $|S|$ on the diagonal.

Approach: Show that a matrix S' cannot simultaneously satisfy conditions (eq. 8) and (eq. 10).

According to (eq. 6) and condition (eq. 10), we require that

$$s_1 W = (s'_{11}, s'_{12}, s'_{13}, ..., s'_{1N}) = (|s_{11}|, s'_{12}, s'_{13}, ..., s'_{1N}) \tag{11}$$

with $s'_{1i} \leq 0$ for all $i \in \{2, ..., N\}$. Analogously,

$$s_2 W = (s'_{21}, s'_{22}, s'_{23}, ..., s'_{2N}) = (s'_{21}, |s_{22}|, s'_{23}, ..., s'_{2N}) \tag{12}$$

with $s'_{2i} \leq 0$ for all $i \in \{1, 3, ..., N\}$. Moreover, since $\|s_1 + s_2\|_0 = 2 < K$ we know that $s_1 + s_2$ has non-zero support under P_K, so condition (eq. 10) must also hold for it. Thus, we need that

$$\begin{aligned}(s_1 + s_2)W &= (|s_{11} + s_{21}|, |s_{12} + s_{22}|, \gamma_1, ..., \gamma_{N-2}) \\ &= (|s_{11} + 0|, |0 + s_{22}|, \gamma_1, ..., \gamma_{N-2}) \\ &= (|s_{11}|, |s_{22}|, \gamma_1, ..., \gamma_{N-2})\end{aligned} \tag{13}$$

with some non-positive $\gamma_i \leq 0$ for all $i \in \{1, ..., N-2\}$. However, because of linearity,

$$\begin{aligned}(|s_{11}|, |s_{22}|, \gamma_1, ..., \gamma_{N-2}) &= (s_1 + s_2)W \\ &= s_1 W + s_2 W \\ &= (|s_{11}|, s'_{12}, s'_{13}, ..., s'_{1N}) \\ &+ (s'_{21}, |s_{22}|, s'_{23}, \ldots, s'_{2N}) \\ &= (|s_{11}| + s'_{21}, s'_{12} \\ &+ |s_{22}|, s'_{13} + s'_{23}, ..., s'_{1N} + s'_{2N})\end{aligned} \tag{14}$$

Thus, $|s_{11}| = |s_{11}| + s'_{21}$ and $|s_{22}| = s'_{12} + |s_{22}|$. From which it follows that $s'_{21} = 0$ and $s'_{12} = 0$. By repeating this for all s_i, s_j combinations, we obtain that all off-diagonal elements in S' must be zero. However, that means $S' = \text{diag}(|s_{11}|, ..., |s_{NN}|)$ must be diagonal. This leads to a contradiction, since it would imply that $\text{rank}(S') = N$, violating condition (eq. 8). □

Notes: We can generalise the result to L_1 sparse distributions P_k with $\forall s \in \mathbb{R}^N : s \in \text{supp}(P_k) \iff \|s\|_1 \leq k$ for some $k > 0$. In this case, we would choose $\|s_1\| < \frac{k}{2}$ and $\|s_2\| < \frac{k}{2}$. Thus, again we would have $(s_1 + s_2) \in \text{supp}(P_k)$ since $\|s_1 + s_2\| < k$, allowing the same reasoning.

B. Relating Our Amortisation Gap to Prior Results in Sparse Autoencoders

In this appendix, we clarify how our amortisation-gap theorem aligns with prior work on shallow autoencoders in the sparse coding literature, including references such as (Rangamani et al., 2018) and (Nguyen et al., 2019). While these earlier results may appear to contradict our statement that a single feedforward linear-nonlinear encoder cannot globally recover all sparse codes from fewer measurements ($M < N$), we show that these works rely on *local* or *probabilistic* assumptions. By contrast, our theorem provides a *global*, *worst-case* statement.

Our work presents a *global* impossibility claim: a single-layer linear+σ map cannot perfectly invert *every* K-sparse code if $M < N$. This argument is rank-based and does not rely on training initialisation or a specific data distribution. In contrast, many prior theorems establish *local* (or *near-dictionary*) results: they assume the encoder's weights start sufficiently close to the true dictionary, then show that a ReLU (or threshold) gating can maintain or refine correct sparse recovery for typical data.

The distinction between uniform and distribution-specific recovery is also important. Our proof deals with uniform, adversarially chosen K-sparse codes. If the model must handle *all* codes in $\mathbb{R}^N$ with $\|s\|_0 \leq K$, a single feedforward pass will inevitably fail for some codes. By contrast, much of the prior literature – including (Rangamani et al., 2018; Nguyen et al., 2019) –requires that codes are drawn from a *random* distribution (e.g., sub-Gaussian or mixture-of-Gaussians). This assumption enables high-probability success on *most* sampled codes, but does not guarantee recovery of *all* codes.

Another key distinction lies in single-pass versus multi-pass inference. Our amortisation-gap statement explicitly concerns a *single-layer* feedforward autoencoder. Iterative or unrolled algorithms (e.g., LISTA (Gregor & LeCun, 2010), or multi-layer ReLU stacks) circumvent the rank restriction by repeatedly refining the estimate. Thus, a multi-iteration or multi-layer approach *can* approach near-optimal sparse recovery; but this does not contradict our statement about a *one-pass* linear-nonlinear encoder's inability to decode *every* sparse signal.

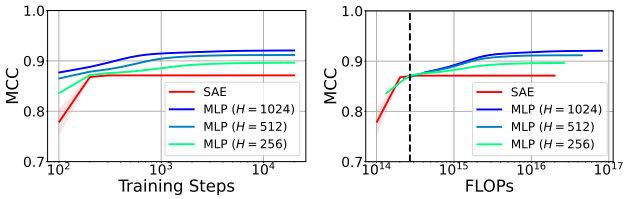
Finally, our result demands *exact* (or perfect) inversion of all feasible codes, while prior analyses often accept *approximate* or *high-probability* correctness. They conclude that, given *some* distribution on codes and an adequately trained near-dictionary encoder, one recovers the support with probability $> 1 - \delta$. This does not conflict with a global impossibility statement.

C. Large-Scale Experiments

To validate that our findings generalise to larger scales more representative of real-world applications, we conducted additional experiments with substantially increased dimensionality. We scaled up our synthetic experiments for the known Z case to $N = 1000$ sparse sources, $M = 200$ measurements, and $K = 20$ active components, training on $500,000$ samples for $20,000$ steps. This represents a significant increase from our base experiments (which used $N = 16$, $M = 8$, $K = 3$), bringing us closer to the scale of actual SAE applications.

For these experiments, we modified our training procedure to use minibatch processing (batch size 1024) to handle the increased data scale efficiently. We evaluated MLPs with hidden layer widths of $H = \{256, 512, 1024\}$ against a standard SAE. The results, shown in Figures 8a and 8b, demonstrate that our key findings about the amortisation

gap not only hold but become more pronounced at larger scales.



(a) MCC vs. training steps (b) MCC vs. total FLOPs

Figure 8: **(Larger N, M and K)** Performance comparison of SAE and MLPs in predicting known latent representations. The black dashed line in (b) indicates the average FLOPs at which MLPs surpass SAE performance.

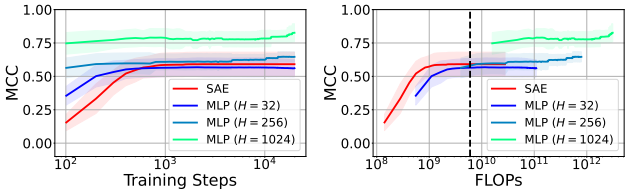
Specifically, the performance gap is slightly more substantial than in our smaller-scale experiments, suggesting that the limitations of linear-nonlinear encoders become more significant as the problem dimensionality increases. This aligns with our theoretical predictions, as the higher-dimensional setting creates more opportunities for interference between features that the simple SAE encoder struggles to disentangle.

The FLOP analysis (Figure 8b) reveals that all MLPs surpass the SAE's performance at approximately 3×10^{14} FLOPs, regardless of hidden layer width. This consistent computational threshold, despite varying model capacities, suggests a fundamental limitation in the SAE's architecture rather than a simple capacity constraint.

D. A Different Distribution of Codes

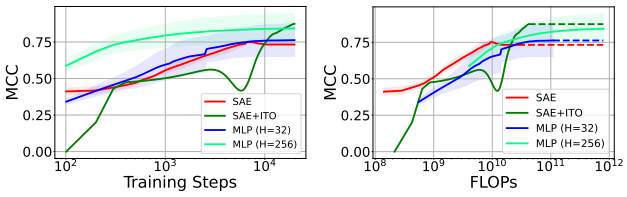
In this appendix, we explore an alternative data generation process that better reflects the distributional properties observed in real-world latent representations. While our main experiments use uniformly sampled sparse codes, recent work has shown that latent features in large models often follow heavy-tailed distributions (e.g., power laws) with varying activation frequencies (Engels et al., 2024; Park et al., 2024). To investigate the robustness of our findings, we modify our synthetic data generator to incorporate a Zipf distribution (parameterised by α) over feature activations. This creates a hierarchical structure where certain features are consistently more likely to be active and have larger magnitudes, while others are more rarely activated. The modified generator maintains the core sparsity constraint of K active dimensions, but weighs both the selection probability and magnitude of each dimension according to its position in the Zipf distribution.

We reproduced all experiments from Section 4.4 using this modified data generation process, with $\alpha = 1.0$. The results reveal several interesting differences while broadly supporting our main conclusions. In the known sparse codes scenario (Figure 9), all methods achieve higher absolute performance, with MLPs reaching MCC values of approximately 0.8 compared to 0.6 in the uniform case. The advantage of wider hidden layers becomes more pronounced under the Zipfian distribution, though the computational threshold at which MLPs surpass SAE performance remains consistent with our original findings.



(a) MCC vs. training steps (b) MCC vs. total FLOPs

Figure 9: **(Zipfian)** Performance comparison of SAE and MLPs in predicting known latent representations. The black dashed line in (b) indicates the average FLOPs at which MLPs surpass SAE performance.

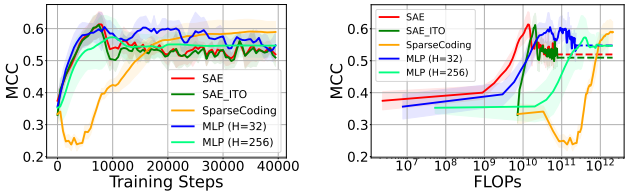


(a) MCC vs. training steps (b) MCC vs. total FLOPs

Figure 10: **(Zipfian)** Performance comparison of SAE, SAE with inference-time optimisation (SAE+ITO), and MLPs in predicting latent representations with a known dictionary. Dashed lines in (b) indicate extrapolated performance beyond the measured range.

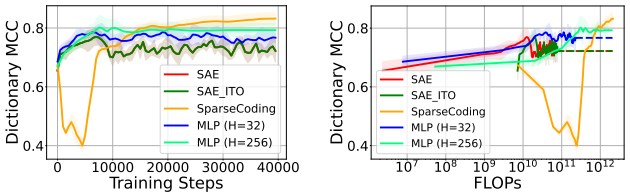
When the dictionary is known but sparse codes are unknown (Figure 10), we observe similar relative performance patterns but with higher peak MCC values (around 0.85 compared to 0.75 in the uniform case). The SAE with inference-time optimisation (SAE+ITO) exhibits more volatile training dynamics under the Zipfian distribution, showing a characteristic performance drop around 10^4 training steps before recovery. This suggests that optimisation becomes more challenging when dealing with hierarchically structured features, though the method ultimately achieves strong performance.

The most substantial differences emerge in the fully unsupervised setting, where both dictionary and sparse codes are unknown (Figure 11). Here, the Zipfian distribution leads to lower overall performance (MCC of 0.5-0.6 versus 0.7-0.8 in the uniform case) and creates clearer separation between different methods. While sparse coding still outperforms



(a) Latent prediction: MCC vs. training steps

(b) Latent prediction: MCC vs. total FLOPs



(c) Dictionary learning: MCC vs. training steps

(d) Dictionary learning: MCC vs. total FLOPs

Figure 11: **(Zipfian)** Dictionary learning performance comparison when both s^* and D^* are unknown.

other approaches, its advantage is less pronounced than in the uniform setting. Dictionary learning under the Zipfian distribution shows increased volatility across all methods, particularly for sparse coding, though the relative ordering of performance remains consistent with our original results.

These findings suggest that while our conclusions about the relative merits of different approaches hold under more realistic distributional assumptions, the absolute difficulty of the sparse inference problem increases when dealing with hierarchically structured features.

E. Decoder weight analysis

A useful method for gaining insight into the behaviour of our models is through examining the final weights of the decoder. Specifically, we visualise $W^\top W$, an $N \times N$ matrix, for three scenarios: when N equals the true sparse dimensionality, when N exceeds it, and when N is smaller than the true dimensionality.

In the case where N matches the true sparse dimension, we observe the matrix $D^\top D$ for the learned decoder matrix D after training. Figure 12 illustrates this scenario for $N = 16$ and $M = 8$, without applying decoder column unit normalisation. For sparse coding, the matrix $D^\top D$ is approximately an $N \times N$ identity matrix after softmax normalisation. This means that the model has learned a set of basis vectors where each column of D is nearly orthogonal to all others, indicating that the features are independent.

In contrast, both the sparse autoencoder (SAE) and the multilayer perceptron (MLP) show $D^\top D$ matrices with a mix of diagonal and off-diagonal elements. In these cases, many

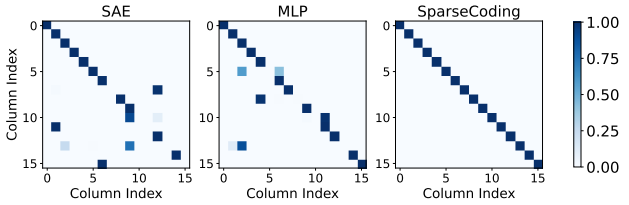


Figure 12: Visualisation of $D^\top D$ when N matches the true sparse dimension. Sparse coding achieves near-identity matrices, while sparse autoencoders (SAE) and multilayer perceptrons (MLP) show significant off-diagonal elements, indicating superposition.

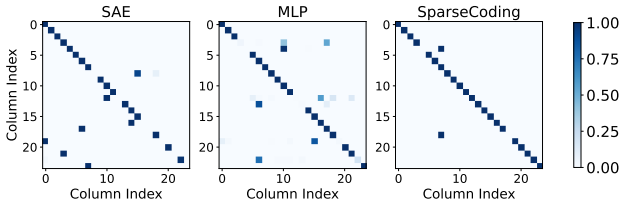


Figure 13: Visualisation of $D^\top D$ when N exceeds the true sparse dimension.

off-diagonal elements are close to 1.0, suggesting that these models utilise superposition, representing more features than there are dimensions. This is suboptimal in this particular scenario because the models have the exact number of dimensions required to represent the feature space effectively. Notably, this superposition effect diminishes when vector normalisation is applied during training.

We observe similar patterns when N is greater than the true sparse dimensionality (Figure 13) and when N is smaller (Figure 14). In cases where N exceeds the required dimensionality, sparse coding still strives to maintain orthogonal feature directions, leading to a near-identity matrix. However, both SAEs and MLPs show stronger correlations between features, as indicated by larger off-diagonal elements, though MLPs exhibit less extreme correlations (e.g., off-diagonal values of around 0.5).

When N is smaller than the true sparse dimension (Figure 14), sparse coding again attempts to maintain orthogonality, though it is constrained by the reduced number of dimensions. The SAE and MLP models, in contrast, continue to exhibit superposition, with off-diagonal elements close to 1.0. MLPs, however, show somewhat weaker correlations between features, as indicated by off-diagonal values around 0.5 in some instances.

F. MLP Ablations

We also wanted to understand in more fine-grained detail how the hidden width of the MLPs affects the key metrics of

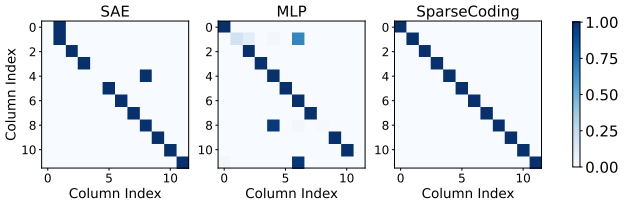


Figure 14: Visualisation of $D^\top D$ when N is smaller than the true sparse dimension.

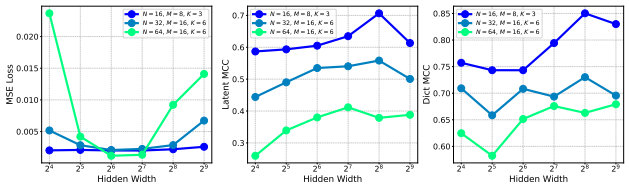


Figure 15: Varying the hidden width of an MLP autoencoder in varying difficulties of dictionary learning regimes. Each data point is an MLP trained for 50,000 iterations with a learning rate of 1e-4.

performance, in different regimes of N, M and K. We show this in Figure 15. We use varying hidden widths and three different combinations of increasingly difficult N, M, K to test this. We train for 50,000 iterations with a learning rate of 1e-4. We see that MCC (both latent and dictionary) increases approximately linearly with hidden width, with a slight drop-off at a hidden width of 512 (most likely due to underfitting). We also see a similar trend in terms of reconstruction loss, with the most difficult case being most sensitive to hidden width.

G. Including a bias parameter

We examine the effect of including a bias parameter in our models in Figure 16. Elhage et al. (2022) noted that a bias allows the model to set features it doesn't represent to their expected value. Further, ReLU in some cases can make "negative interference" (interference when a negative bias pushes activations below zero) between features free. Further, using a negative bias can convert small positive interferences into essentially being negative interferences, which helps deal with noise.

However, Theorem 3.1 doesn't rely on having biases, and although it generalises to the case with biases, we would like to be able to simplify our study by not including them. Thus, we show in Figure 16 that biases have no statistically significant effect on reconstruction loss, latent MCC, dictionary MCC, or L0, for any of the models, except for the L0 and MCC of the MLP, which achieves a higher MCC without bias at the cost of a greater L0.

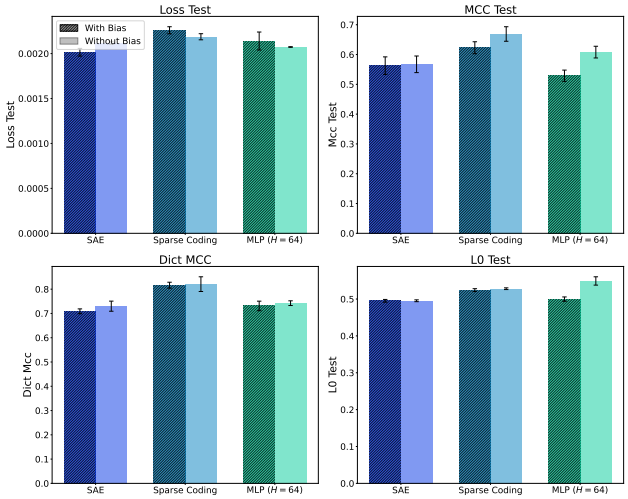


Figure 16: Effects on dictionary learning performance for our three models, with and without a bias. Including a bias has no statistically significant effect on results.

H. Comparison with traditional dictionary learning methods

To provide a comparison with traditional dictionary learning methods, we incorporated the Least Angle Regression (LARS) algorithm to compute the Lasso solution in our experimental framework.

The traditional dictionary learning problem can be formulated as a bi-level optimisation task. Given a set of training samples $X = [x_1, \ldots, x_n] \in \mathbb{R}^{m \times n}$, we aim to find a dictionary $D \in \mathbb{R}^{m \times k}$ and sparse codes $A = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^{k \times n}$ that minimise the reconstruction error while enforcing sparsity constraints:

$$\min_{D,A} \sum_{i=1}^{n} \left(\frac{1}{2} \|x_i - D\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \right)$$

subject to $\|d_j\|_2 \leq 1$ for $j = 1, \ldots, k$, where d_j represents the j-th column of D, and $\lambda > 0$ is a regularisation parameter controlling the trade-off between reconstruction fidelity and sparsity.

In our experiment, we employed the LARS algorithm to solve the Lasso problem for sparse coding, while alternating with dictionary updates to learn the optimal dictionary. Specifically, we used the `scikit-learn` implementation of dictionary learning, which utilises LARS for the sparse coding step. The algorithm alternates between two main steps: (1) sparse coding, where LARS computes the Lasso solution for fixed D, and (2) dictionary update, where D is optimised while keeping the sparse codes fixed.

To evaluate the performance of this traditional approach, we generated synthetic data following the same procedure as in

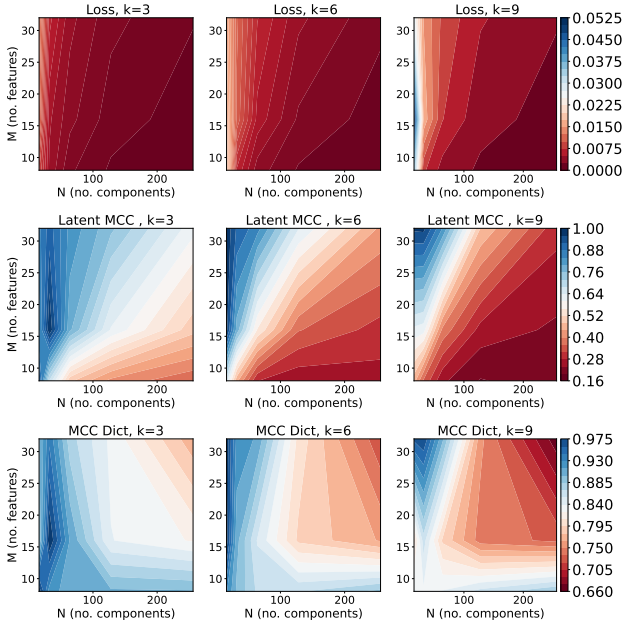


Figure 17: Performance of Least-Angle Regression (LARS) to compute the Lasso solution using our synthetic dictionary learning setup. In general, when comparing to Figure 5, we see an improvement when using LARS over our naïve implementations of SAEs, MLPs and sparse coding, across loss, latent MCC, and dictionary MCC.

our main experiments, with $N = 16$ sparse sources, $M = 8$ measurements, and $K = 3$ active components per timestep. We trained the dictionary learning model on the training set and evaluated its performance on the held-out test set. Performance was measured using the Mean Correlation Coefficient (MCC) between the predicted and true latents, as well as between the learned and true dictionary elements.

The results of this, presented in Figure 17, make clear that traditional sparse coding significantly outperforms our vanilla gradient-based implementations, particularly in terms of latent MCC and dictionary MCC. Whilst our results from the main body show that there does exist a significant amortisation gap between the vanilla implementations of each of the approaches, we should also attempt to understand how the optimised versions of each method compare. We discuss this in the following subsection.

H.1. Optimised Sparse Autoencoders and Sparse Coding

Our initial implementations of sparse autoencoders (SAEs) and sparse coding, while functional, are far from optimal. They represent the minimum computational mechanisms required to solve the problems as we have formulated them. However, more sophisticated approaches can significantly improve performance and address inherent limitations.

H.1.1. Advanced Sparse Autoencoder Techniques

Sparse autoencoders trained with L1 regularisation are susceptible to the *shrinkage problem*. Wright & Sharkey (2024) identified feature suppression in SAEs, analogous to the activation shrinkage first described by Tibshirani (1996) as a property of L1 penalties. The shrinkage problem occurs when L1 regularisation reduces the magnitude of non-zero coefficients to achieve a lower loss, potentially underestimating the true effect sizes of important features.

Several techniques have been proposed to mitigate this issue:

- **ProLU Activation**: Taggart (2024) introduced the ProLU activation function to maintain scale consistency in feature activations.
- **Gated SAEs**: Rajamanoharan et al. (2024a) developed Gated Sparse Autoencoders, which separate the process of determining active directions from estimating their magnitudes. This approach limits the undesirable side effects of L1 penalties and achieves a Pareto improvement over standard methods.
- **JumpReLU SAEs**: Rajamanoharan et al. (2024b) proposed JumpReLU SAEs, which set activations below a certain threshold to zero, effectively creating a non-linear gating mechanism.
- **Top-k SAEs**: Originally proposed by Makhzani & Frey (2013), top-k SAEs were shown by Gao et al. (2024) to prevent activation shrinkage and scale effectively to large language models like GPT-4.

H.1.2. Optimised Sparse Coding Approaches

Our initial sparse coding model, using uniformly initialised latents and concurrent gradient-based optimisation of both sparse codes and the dictionary, is suboptimal. The sparse coding literature offers several more sophisticated approaches:

- **Least Angle Regression (LARS)**: Introduced by Efron et al. (2004), LARS provides an efficient algorithm for computing the entire regularisation path of Lasso. It is particularly effective when the number of predictors is much larger than the number of observations.
- **Orthogonal Matching Pursuit (OMP)**: Pati et al. (1993) proposed OMP as a greedy algorithm that iteratively selects the dictionary element most correlated with the current residual. It offers a computationally efficient alternative to convex optimisation methods.

Future work will involve pitting these against the optimised SAE architectures discussed above.

H.2. Top-k sparse coding

Building on this exploration, we introduced a top-k sparse coding approach. We aimed to determine whether (1) setting very small active latents to zero would improve performance and (2) optimising with a differentiable top-k function, rather than using exponential or ReLU functions, could yield further benefits.

Figure 18 presents the results of these experiments. We first trained the sparse coding model for 20,000 steps on the training data and optimised for an additional 1,000 steps on the test data. During this process, we measured mean squared error (MSE) loss, latent MCC, and the L_0 norm of the latent codes. Due to the presence of very small active latents, all initial setups led to an L_0 value of 1.0, indicating that all latents were active, as shown by the blue star in the figure. We also show a sparse autoencoder trained with different L_1 penalties as a comparison.

Next, we applied a top-k operation to enforce sparsity by setting all but the top-k largest activations to zero. This process resulted in improved L_0 values, but the MSE loss and MCC results indicated that the top-k optimisation itself was hampered by an insufficient learning rate. We hypothesise that with proper tuning of hyperparameters, we could achieve Pareto improvements by using the top-k function directly, rather than applying it to exponentiated codes.

We believe that further adjustments to the optimisation process, including a higher learning rate for top-k functions, could result in better performance. Additionally, applying the top-k function directly, without exponentiating the codes, may offer further gains in performance and sparsity.

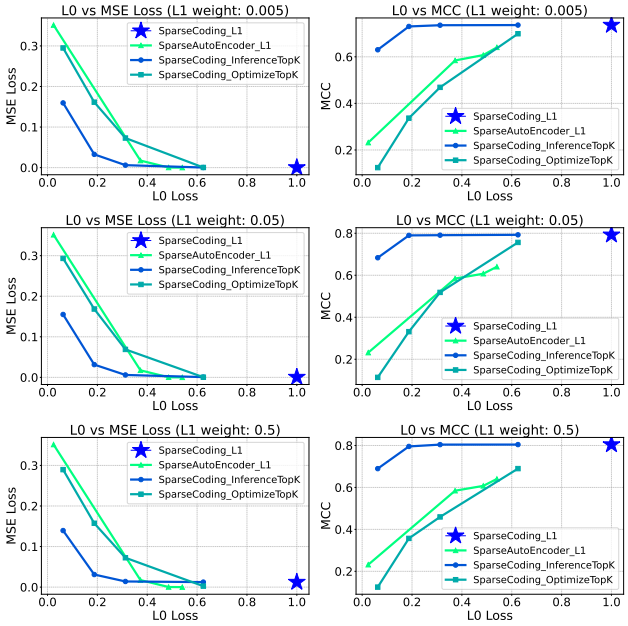


Figure 18: Comparison of L_0 loss vs. MSE loss and L_0 loss vs. MCC for Sparse Coding with L1 regularization, top-k inference, and top-k optimization, alongside results for Sparse Autoencoder. Blue stars represent the initial model's performance, while curves illustrate the results of applying top-k sparsity.

I. Measuring FLOPs

To quantify the computational cost of each method, we calculate the number of floating-point operations (FLOPs) required for both training and inference. This section details our approach to FLOP calculation for each method.

I.1. Sparse Coding

For sparse coding, we calculate FLOPs for both inference and training separately.

Inference: The number of FLOPs for inference in sparse coding is given by:

$$\text{FLOPs}_{\text{SC-inf}} = \begin{cases} 3MN + Nn_s & \text{if learning } D \\ 2MN + Nn_s & \text{otherwise} \end{cases} \tag{15}$$

where M is the number of measurements, N is the number of sparse sources, and n_s is the number of samples. The additional MN term when learning D accounts for the normalisation of the dictionary.

Training: For training, we calculate the FLOPs as:

$$\text{FLOPs}_{\text{SC-train}} = n_{\text{eff}} \cdot (\text{FLOPs}_{\text{forward}} + \text{FLOPs}_{\text{loss}} + \text{FLOPs}_{\text{backward}} + \text{FLOPs}_{\text{update}})$$

where $n_{\text{eff}} = n_{\text{steps}} \cdot \frac{n_b}{n_s}$ is the effective number of iterations, n_{steps} is the number of training steps, n_b is the batch size, and n_s is the total number of samples. The component FLOPs are calculated as:

$$\begin{aligned} \text{FLOPs}_{\text{forward}} &= \text{FLOPs}_{\text{SC-inf}} \\ \text{FLOPs}_{\text{loss}} &= 2Mn_b + Nn_b \\ \text{FLOPs}_{\text{backward}} &\approx 2 \cdot \text{FLOPs}_{\text{forward}} \\ \text{FLOPs}_{\text{update}} &= \begin{cases} Nn_b + MN & \text{if learning } D \\ Nn_b & \text{otherwise} \end{cases} \end{aligned}$$

I.2. Sparse Autoencoder (SAE)

For the sparse autoencoder, we calculate FLOPs for both training and inference.

Training: The total FLOPs for SAE training is given by:

$$\text{FLOPs}_{\text{SAE-train}} = n_{\text{eff}} \cdot (\text{FLOPs}_{\text{forward}} + \text{FLOPs}_{\text{backward}})$$

where n_{eff} is defined as before, and:

$$\text{FLOPs}_{\text{forward}} = \begin{cases} 5MN + N & \text{if learning } D \\ 4MN + N & \text{otherwise} \end{cases}$$

$$\text{FLOPs}_{\text{backward}} = N + (2NM + N) + 2NM + 2(MN + N) + \begin{cases} 2NM & \text{if learning } D \\ 0 & \text{otherwise} \end{cases}$$

Inference: For SAE inference, the FLOPs are calculated as:

$$\text{FLOPs}_{\text{SAE-inf}} = (4MN + N) \cdot n_s$$

I.3. Multilayer Perceptron (MLP)

For the MLP, we calculate FLOPs for both training and inference, considering a single hidden layer of size H.

Training: The total FLOPs for MLP training is given by:

$$\text{FLOPs}_{\text{MLP-train}} = n_{\text{eff}} \cdot (\text{FLOPs}_{\text{forward}} + \text{FLOPs}_{\text{backward}})$$

where:

$$\text{FLOPs}_{\text{forward}} = \begin{cases} 2MH + H + 2HN + N + 2NM + MN & \text{if learning } D \\ 2MH + H + 2HN + N + 2NM & \text{otherwise} \end{cases}$$

$$\text{FLOPs}_{\text{backward}} = N + (2NH + N) + H + (2MH + H) + 2NM + 2(MH + H + HN + N)$$

where we add $2NM$ to $\text{FLOPs}_{\text{backward}}$ if learning D, and not otherwise.

Inference: For MLP inference, the FLOPs are calculated as:

$$\text{FLOPs}_{\text{MLP-inf}} = (2MH + H + 2HN + N + 2NM) \cdot n_s$$

I.4. SAE with Inference-Time Optimisation (SAE+ITO)

For SAE+ITO, we calculate the additional FLOPs required for optimising the codes during inference:

$$\text{FLOPs}_{\text{ITO}} = (MN + N + n_{\text{iter}} \cdot (4MN + 2M + 11N)) \cdot n_s$$

where n_{iter} is the number of optimisation iterations performed during inference.

J. Automated interpretability

In this section, we describe the automated interpretability pipeline used to understand and evaluate the features learned by sparse autoencoders (SAEs) and other models in the context of neuron activations within large language models (LLMs). The pipeline consists of two tasks: feature interpretation and feature scoring. These tasks allow us to generate hypotheses about individual feature activations and to determine whether specific features are likely to activate given particular token contexts.

J.1. Feature Interpreter Prompt

We use a feature interpreter prompt to provide an explanation for a neuron's activation. The interpreter is tasked with analysing a neuron's behaviour, given both text examples and the logits predicted by the neuron. Below is a summary of how the interpreter prompt works:

You are a meticulous AI researcher conducting an investigation into a specific neuron in a language model. Your goal is to provide an explanation that encapsulates the behavior of this neuron. You will be given a list of text examples on which the neuron activates. The specific tokens that cause the neuron to activate will appear between delimiters like `<<this>>`*. If a sequence of consecutive tokens causes the neuron to activate, the entire sequence of tokens will be contained between delimiters* `<<just like this>>`*. Each example will also display the activation value in parentheses following the text. Your task is to produce a concise description of the neuron's behavior by describing the text features that activate it and suggesting what the neuron's role might be based on the tokens it predicts. If the text features or predicted tokens are uninformative, you can omit them from the explanation. The explanation should include an analysis of both the activating tokens and contextual patterns. You will be presented with tokens that the neuron boosts in the next token prediction, referred to as* `Top_logits`*, which may refine your understanding of the neuron's behavior. You should note the relationship between the tokens that activate the neuron and the tokens that appear in the* `Top_logits` *list. Your final response should provide a formatted explanation of what features of text cause the neuron to activate, written as:* `[EXPLANATION]: <your explanation>`*.*

J.2. Feature Scorer Prompt

After generating feature interpretations, we implemented a scoring prompt to predict whether a specific feature is likely to activate on a given token. This ensures that the explanations generated by the interpreter align with actual activations. The scoring prompt tasks the model with evaluating if the tokens marked in the examples are representative of the feature in question.

You are provided with text examples where portions of the sentence strongly represent the feature, with these portions enclosed by `<< and >>`*. Some of these examples might be mislabeled. Your job is to evaluate each example and return a binary response (1 if the tokens are correctly labeled, and 0 if they are mislabeled). The output must be a valid Python list with 1s and 0s, corresponding to the correct or incorrect labeling of each example.*

Table 1: Example interpretations from MLP and SAE neurons, shown with their F1 scores.

Model	*Interpretation*	*F1 Score*
MLP	Activates on the token "to" when used to introduce an infinitive verb indicating purpose or intent, promoting verbs that express actions or goals	0.899
	Activates on concrete and functional nouns or specific actions that are often part of a list or enumeration	0.899
	Activates on the token "than" as part of a comparative structure, aiding in predicting terms used for comparison or establishing norms	1.000
SAE	Activates on parentheses and colons used in structured timestamps, date-time formats, and categorisation notations	1.000
	Activates on tokens within contexts related to font and text styling options, typically presented in a technical or settings menu format	1.000
	Activates on the token "first" within the formulaic expression "first come, first served basis"	1.000

J.3. Evaluation of Automated Interpretability

To evaluate the accuracy of the interpretations generated by the feature interpreter and feature scorer, we compared model-generated explanations against held-out examples. The evaluation involved calculating the F1-score, which was done by presenting the model with a mix of correctly labeled and falsely labeled examples. The model was then tasked with predicting whether each token in the example represented a feature or not, based on the previously generated interpretation. By comparing the model's predictions with ground truth labels, we can assess how accurately the feature interpretation aligns with actual neuron activations. This process helps validate the interpretability of the features learned by SAEs, MLPs, and other models.

This pipeline is based on the work of Juang et al. (2024), which itself builds on the work of others. Bills et al. (2023) used GPT-4 to generate and simulate neuron explanations by analyzing text that strongly activated the neuron. Bricken et al. (2023) and Templeton (2024) applied similar techniques to analyze sparse autoencoder features. Templeton (2024) also introduced a specificity analysis to rate explanations by using another LLM to predict activations based on the LLM-generated interpretation. This provides a quantification of how interpretable a given neuron or feature actually is. Gao et al. (2024) demonstrated that cheaper methods, such as Neuron to Graph (Foote et al.), which uses n-gram based explanations, allow for a scalable feature labeling mechanism that does not rely on expensive LLM computations.

Table 1 presents illustrative examples of interpretations from both MLP and SAE neurons, showing how our automated pipeline can identify specific linguistic patterns and assign quantitative reliability scores.