# SPIKE-AND-SLAB AUTOENCODERS WITH ADAPTIVE SPARSITY

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Sparse Autoencoders (SAEs) learn sparse representations of deep neural activations, but fixed Top-$k$ architectures impose the same feature budget on every input. This aspect limits their ability to adapt representational capacity to inputs of different complexity. In this paper, we introduce Spike-and-Slab Sparse Autoencoders ($S^2$-SAE), a Bayesian framework in which the sparsity level $k$ can vary using a global learned threshold. By casting sparse encoding as a MAP inference problem, we derive an objective that balances reconstruction fidelity and expected sparsity. We additionally introduce an optional post-hoc refinement that selects among candidate code lengths without retraining. We evaluate our framework on representations from image classification models and Large Language Models, including Pythia-160M and Gemma-2-2B. Results show that $S^2$-SAE improves the observed sparsity–reconstruction Pareto frontier across vision and language benchmarks. These findings connect Bayesian sparse coding with practical adaptive sparsity, with potential applications to concept discovery in neural representations. Code is available at `https://anonymous.4open.science/r/s2sae`.

## 1 INTRODUCTION

A central challenge in deep learning, and especially in the field of mechanistic interpretability research is to connect high-dimensional neural representations with structured abstractions that support interpretation, debugging, intervention, and reasoning (Bereska & Gavves, 2024). Deep learning models rely on distributed representations to achieve strong downstream performance, but these representations are difficult to inspect and manipulate. In the literature, neurons are considered *polysemantic*, i.e., they activate under a large set of understandable concepts.

Within this field, Concept Bottleneck Models (CBMs) provide a prime example of one explicit interface between neural and symbolic representations: they first predict a set of human-defined concepts and then use those concepts to solve a downstream task (Koh et al., 2020). Their intermediate state is understandable and supports direct intervention, but training normally requires concept annotations. Such labels are expensive, incomplete, noisy, and potentially unable to capture all abstractions learned by a neural model (Bergamin et al., 2026).

In recent years, an alternative route emerged, which involves discovering concept-like features directly from neural activations, moving from a supervised strategy to an unsupervised (or self-supervised) objective. Sparse Autoencoders (SAEs) train an overcomplete autoencoder to reconstruct the internal activations of a pretrained model while enforcing sparsity in the latent representation. These sparse features can capture recurring activation patterns of the model, making SAEs a promising mechanism for unsupervised feature discovery (Elhage et al., 2022; Bricken et al., 2023; Cunningham et al., 2023; Templeton et al., 2024). After training, it is possible to inspect examples that strongly activate a feature and manually give a natural-language interpretation of its behavior (Bricken et al., 2023; Templeton et al., 2024). Since manual inspection does not scale to a high number of features learned by modern SAEs, language models can be prompted to generate candidate explanations from a set of examples (Paulo et al., 2025).

However, popular SAE architectures rely on rigid, sample-independent sparsity mechanisms. A common design choice is to employ a simple 1-layer autoencoder with fixed Top-$k$ selection, where only the $k$ largest latent features are retained for every input (Gao et al., 2025). This mechanism imposes the same representational budget on all examples. This can be problematic: simple inputs



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

may require only a small number of active concept-like features, whereas more complex inputs may require a richer description. A fixed-length sparse code may therefore over-represent simple cases and under-represent complex ones, limiting its usefulness as an adaptive concept interface. Existing approaches already relax fixed per-input sparsity. JumpReLU and BatchTopK allow variable feature counts through learned thresholds and batch-level allocation, respectively (Rajamanoharan et al., 2024b; Bussmann et al., 2024). AdaptiveK explicitly selects an input-dependent $k$ using predicted text complexity (Yao et al., 2026). These approaches motivate our investigation of whether a Bayesian derivation can guide the design of adaptive sparsity mechanisms.

We use a spike-and-slab model to motivate a shared selection threshold, which yields exact MAP inference in the idealized case of orthogonal decoder vectors (i.e., when all decoder's feature vectors are perpendicular, each accounting for a separate part of the input). For practical, overcomplete dictionaries, a learned encoder and threshold approximate this selection rule, while an average sparsity constraint controls the overall feature budget.

Building on this formulation, we introduce Spike-and-Slab Sparse Autoencoders ($S^2$-SAE). Our contributions are threefold:

- We establish the theoretical connection between a shared activation cost and a common threshold rule for coordinate-wise inference.
- We develop a practical autoencoder that learns this threshold under an average sparsity constraint, together with an optional post-hoc refinement that selects among candidate code lengths without retraining.
- We benchmark $S^2$-SAE on ConvNeXt-Tiny representations from Caltech-UCSD Birds-200-2011 and Oxford-IIIT Pet, and on Pythia-160M and Gemma-2-2B representations using the SAEBench evaluation framework (Karvonen et al., 2025).

Our experiments assess reconstruction quality, variation in feature allocation, preservation of downstream behavior, and the trade-off between inference cost and reconstruction fidelity. Results show improvements in the sparsity–reconstruction trade-off across most evaluated settings, with stronger gains on vision tasks and more mixed results on language benchmarks. Our findings provide practical insights into the design and implementation of SAEs with adaptive sparsity, showing how a learned global threshold enables input-dependent feature counts.

## 2 Related Work

**Sparsity in sparse autoencoders.** Sparse autoencoders (SAEs) learn an overcomplete dictionary of latent features by jointly optimizing reconstruction fidelity and sparsity. Early SAE formulations commonly encourage sparse representations through an $\ell_1$ penalty on latent activations (Cunningham et al., 2023; Bricken et al., 2023). While simple and differentiable, $\ell_1$ regularization acts on activation magnitudes rather than directly controlling the number of active features, and can introduce shrinkage.

Several recent approaches therefore target sparsity more directly. TopK SAEs (Gao et al., 2025) retain the $k$ largest latent activations for each input, providing explicit control over the number of active features and strong reconstruction performance. However, this imposes the same representational budget on every input, independently of how many features may be useful for its reconstruction. Recent methods relax this fixed per-example cardinality in different ways. JumpReLU SAEs (Rajamanoharan et al., 2024b) replace the standard ReLU with a discontinuous thresholded activation and optimize the resulting $\ell_0$ sparsity using a straight-through estimator, naturally allowing the number of active features to vary between inputs. BatchTopK (Bussmann et al., 2024) instead moves the TopK constraint from individual samples to the batch level: a total activation budget is shared across the batch, allowing different samples to receive different numbers of active latents while directly controlling average sparsity.

These approaches provide strong evidence that relaxing fixed per-example sparsity can improve the reconstruction–sparsity trade-off. Like JumpReLU and BatchTopK, $S^2$-SAE permits the number of active features to vary across inputs. JumpReLU learns feature-specific thresholds, whereas our method learns a single shared threshold under a dual average-sparsity constraint. BatchTopK allocates a batch-level activation budget during training and derives a threshold for inference. Our

contribution combines a spike-and-slab derivation of the shared activation criterion with a practical training procedure and an optional reconstruction-based refinement of code length. AdaptiveK (Yao et al., 2026) adapts Top-$k$ selection using a linear probe trained on text-complexity annotations. The probe predicts a complexity score from model activations, which determines the number of features retained. $S^2$-SAE shares the goal of input-dependent feature allocation, but learns a global activation threshold under an average sparsity constraint. Its feature count therefore emerges from thresholding learned scores, without complexity annotations or a separate complexity predictor.

**$\ell_0$ regularization and probabilistic sparse representations.** Directly optimizing cardinality is challenging because the $\ell_0$ pseudo-norm is discrete and non-differentiable. A substantial literature has therefore developed differentiable or stochastic approximations to discrete feature selection. In particular, Louizos et al. (2018) formulate neural-network sparsification through stochastic gates and introduce the hard-concrete distribution, making the expected $\ell_0$ objective amenable to gradient-based optimization. Similar works are appreciated in the interpretable machine learning community as well, e.g., for mitigating the complexity of interpretable rule set representations (Bergamin et al., 2025). Related gating and continuous-relaxation approaches provide general mechanisms for learning sparse structures despite discrete selection decisions. In the SAE setting, JumpReLU similarly uses straight-through estimation to optimize an $\ell_0$ activation objective directly (Rajamanoharan et al., 2024b). Our setting differs in that the $\ell_0$ term is not introduced primarily as a regularizer to be approximated: it arises directly from MAP estimation under the spike-and-slab model. Moreover, rather than relaxing each discrete activation decision through an independent stochastic gate, we exploit the resulting structure to obtain a common activation criterion and use a dual variable to control the average cardinality. The theoretical connection to $\ell_0$ regularization is therefore used to motivate the allocation rule, while differentiability during training is handled through the straight-through approximation described in Section 3.

## 3 METHODOLOGY

Let $\boldsymbol{x} \in \mathbb{R}^d$ be an activation from a pretrained model. It serves as both the input to the SAE and its self-supervised reconstruction target. The encoder maps $\boldsymbol{x}$ to a nonnegative code $\boldsymbol{z} \in \mathbb{R}^m_{\geq 0}$, and the decoder reconstructs it as $\hat{\boldsymbol{x}} = \boldsymbol{D}\boldsymbol{z}$, where $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_m]$. We use an overcomplete dictionary ($m > d$) with unit-norm decoder vectors, i.e., $\|\boldsymbol{d}_j\|_2 = 1$ (Cunningham et al., 2023).

Our methodology consists of three main steps:

1. We derive a **Bayesian objective** from a spike-and-slab model, showing how the Bayesian formulation naturally assigns a cost to each active feature, promoting sparsity.
2. We use the resulting MAP structure to motivate two practical **inference methods**: a fast global-threshold rule and a more computationally expensive post-hoc refinement.
3. We propose a single **training method** under an average sparsity constraint, allowing the number of active features to vary across inputs while controlling the overall feature budget.

**Bayesian objective.** A sparse code specifies which features are active and their magnitude. For each feature $j$, we introduce an inclusion binary variable $s_j$ and a nonnegative magnitude $u_j$:

$$s_j \sim \text{Bernoulli}(\pi), \qquad u_j \sim \text{HalfNormal}(\alpha), \qquad z_j = s_j u_j.$$

where the HalfNormal distribution can be seen as a zero-centered normal distribution restricted to its nonnegative half: equivalently, if $v_j$ follows a zero-centered Normal distribution, then $u_j = |v_j|$ follows a HalfNormal distribution. $\alpha$ is its precision parameter. We draw $s_j$ and $u_j$ independently; consequently, $z_j$ follows a spike-and slab distribution (Goodfellow et al., 2012): it has probability $1 - \pi$ of being exactly zero and otherwise it follows a positive half-normal distribution.

We model SAE reconstruction error as an additional Gaussian noise component with variance $\sigma^2$ for each coordinate. After applying Bayes' theorem and maximizing the joint posterior $p(\boldsymbol{s}, \boldsymbol{u} \mid \boldsymbol{x}, \boldsymbol{D})$, maximum a posteriori (MAP) inference prescribes as best the most probable feature choices $\boldsymbol{s}$ and magnitude $\boldsymbol{u}$. With $\pi < 1/2$ promoting sparsity, this corresponds to:

$$\hat{\boldsymbol{z}}_{\mathrm{MAP}}(\boldsymbol{x}) = \arg\min_{\boldsymbol{z}\geq 0} \left\{ \underbrace{\frac{\|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{z}\|_2^2}{2\sigma^2}}_{\text{recon. error}} + \underbrace{\frac{\alpha}{2}\|\boldsymbol{z}\|_2^2}_{\text{magnitude}} + \underbrace{\rho\|\boldsymbol{z}\|_0}_{\text{sparsity}} \right\}, \qquad \rho = \log\frac{1-\pi}{\pi}. \tag{1}$$

The first term favors a good reconstruction, the second limits the magnitude of the latent features, and the third promotes few active features, where each additional feature has a positive cost $\rho$. The cost $\rho$ suggests a simple procedure to select a feature: a feature is worth activating when the reduction in the continuous reconstruction-and-magnitude terms exceeds its inclusion cost $\rho$.

This objective characterizes inference under the generative model. For a unit-normalized dictionary, the exact coordinate-wise solution is

$$z_j = \frac{c_j}{1+\alpha\sigma^2}\,\mathbf{1}\left[c_j > \sqrt{2\sigma^2(1+\alpha\sigma^2)\rho}\right], \qquad c_j = (\boldsymbol{d}_j^\top \boldsymbol{r}_{-j})_+,$$

where $\boldsymbol{r}_{-j} = \boldsymbol{x} - \sum_{\ell\neq j}\boldsymbol{d}_\ell z_\ell$ is the residual excluding feature $j$. Thus, a feature is activated when its residual-dependent evidence exceeds a shared threshold. For an orthogonal dictionary, these decisions decouple and yield the global MAP solution. For an overcomplete dictionary, feature interactions make inference coupled; simultaneous thresholding is generally not an exact MAP solution. The whole derivation is provided in Appendix A.

**Inference methods.** When using an overcomplete dictionary, some encoded features must explain overlapping parts of $\boldsymbol{x}$. Therefore, choosing to activate a feature may affect the reconstruction performance of another one. Finding the best MAP coding for every activation requires iterative, expensive optimization. Instead, we resort to a fast approximation, learned by the SAE encoder, to produce feature scores $\boldsymbol{a}_\phi(\boldsymbol{x})$ in one pass. Our fast inference rule checks the score vector $\boldsymbol{a}_\phi(\boldsymbol{x})$ against a global learned threshold $\tau > 0$, implemented as $e^{\theta_{\text{global}}}$, to implement a variable sparsity level according to the extracted scores from $\boldsymbol{x}$:

$$\boldsymbol{a}_\phi(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{W}_e\boldsymbol{x} + \boldsymbol{b}_e), \qquad z_j(\boldsymbol{x};\tau) = a_{\phi,j}(\boldsymbol{x})\mathbf{1}[a_{\phi,j}(\boldsymbol{x}) > \tau]. \tag{2}$$

We also introduce an optional post-hoc refinement that uses the same trained encoder and decoder. For each input, we construct candidate codes $\boldsymbol{z}^{(k)}(\boldsymbol{x})$ by retaining the largest $k$ encoder scores and setting the remaining entries to zero, with $\boldsymbol{z}^{(0)} = \mathbf{0}$. We then select

$$k^\star(\boldsymbol{x}) = \arg\min_{0\leq k\leq K_{\max}} \left\{ \underbrace{\|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{z}^{(k)}(\boldsymbol{x})\|_2^2/d}_{\text{reconstruction error}} + \eta k \right\}. \tag{3}$$

Here, $K_{\max}$ is the largest candidate code length and $\eta$ is the cost per retained feature. We select $\eta$ on validation data so that refined inference has a mean $\ell_0$ as close as possible to that of fast inference. This procedure minimizes the reconstruction-cardinality objective over the tested prefixes with fixed encoder scores. It does not optimize over all feature subsets. Unlike the fast rule, it evaluates multiple reconstructions for every input, giving a sensible tradeoff between inference time and fidelity.

**Training method.** We use the MAP derivation to motivate the architecture, while training the encoder and decoder under an average sparsity constraint. The practical objective omits the magnitude penalty, following the broad-slab approximation, and replaces a fixed activation cost with a dual variable that adapts to a prescribed average feature budget. The global threshold is learned directly rather than computed from the prior parameters. Consequently, training does not guarantee that the encoder recovers the exact MAP code.

Let $\ell_0(\boldsymbol{x};\tau) = \|\boldsymbol{z}(\boldsymbol{x};\tau)\|_0$ denote the number of active features under a given global threshold $\tau$. We aim to keep its average $\ell_0$ sparsity at most $K_{\text{target}}$, while allowing individual inputs to use a variable number of features. Since the hard threshold operation in Eq. 2 has zero derivative almost everywhere, we use a straight-through estimator for the gates and the resulting cardinality estimate $\tilde{\ell}_0(\boldsymbol{x})$ (Bengio et al., 2013; Rajamanoharan et al., 2024b). The actual loss is:

$$\mathcal{L}_{\text{train}} = \mathbb{E}_{\boldsymbol{x}}\left[\underbrace{\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}(\boldsymbol{x};\tau)\|_2^2/d}_{\text{reconstruction error}}\right] + \theta_{\text{sparsity}}\left(\underbrace{\frac{\mathbb{E}_{\boldsymbol{x}}[\tilde{\ell}_0(\boldsymbol{x};\tau)]}{K_{\text{target}}} - 1}_{\text{sparsity}}\right) + \gamma\,\underbrace{\mathcal{L}_{\text{AuxK}}}_{\text{activity}}. \tag{4}$$

The nonnegative weight $\theta_{\text{sparsity}}$ is a dual variable updated through projected dual ascent that controls the average $\ell_0$ sparsity (Elenter et al., 2024). After each training step, we update it using the measured batch count: $\theta_{\text{sparsity}} \leftarrow \left[\theta_{\text{sparsity}} + \eta_{\text{dual}}\left(\frac{\mathbb{E}_{\text{batch}}[\ell_0(\boldsymbol{x})]}{K_{\text{target}}} - 1\right)\right]_+$. In practice, it is increased when the observed batch exceeds the target $\ell_0$ sparsity, discouraging the final solution to exceed the prescribed budget $K_{\text{target}}$. Finally, $\mathcal{L}_{\text{AuxK}}$ supplies a training signal to inactive features through auxiliary residual reconstruction (Gao et al., 2025). It is an optimization aid and is not derived from the probabilistic model. Further implementation details are provided in Appendices B.1 and B.2.

# 4 Evaluation

The evaluation is organized around four questions. RQ1 evaluates whether S$^2$-SAE improves reconstruction at a measured sparsity level. RQ2 checks how the number of active features varies across inputs and across the evaluated models. RQ3 investigates if the reconstructed activations preserve the behavior of the frozen models. RQ4 asks whether the faster methodology is effective in providing a tradeoff between reconstruction quality and inference latency.

## 4.1 Experimental Setup

**Datasets and models.** For vision, we consider two datasets, namely Caltech-UCSD Birds-200-2011 (CUB) (Welinder et al., 2010) and Oxford-IIIT Pet (Parkhi et al., 2012). For each dataset, we fine-tune an ImageNet-pretrained ConvNeXt-Tiny classifier (Liu et al., 2022), using class labels, without concept annotations, and then freeze its parameters. The SAE reconstructs the $768 \times 7 \times 7$ feature map of the classifier (i.e., before global pooling). Each of its 49 spatial positions constitutes a separate activation vector with $d = 768$. We use dictionaries with $m = 3,072$ features and train all vision SAEs on the same classifier activations and fixed data splits. For language, we use pretrained Pythia-160M (Biderman et al., 2023) and Gemma-2-2B (Gemma Team, 2024), evaluated using the SAEBench (Karvonen et al., 2025). The inputs of the SAE are residual-stream activation vectors at token-level granularity, located at the 8th and 12th layer, respectively, with dimension $d = 768$ and $d = 2,304$. Each S$^2$-SAE configuration is trained on 500 million token activations from the Pile Uncopyrighted dataset (Gao et al., 2020).

**Compared models and inference protocols.** For vision, we compare S$^2$-SAE with fixed Top-$k$, BatchTopK, $\ell_1$-regularized SAEs, and JumpReLU. For language, we compare against published SAEBench checkpoints, with additional baselines including Gated SAEs (Rajamanoharan et al., 2024a), $p$-annealing (Karvonen et al., 2024), and Matryoshka BatchTopK (Bussmann et al., 2025). We evaluate fast and refined S$^2$-SAE inference on the same trained checkpoints. The refinement penalty $\eta$ is selected on validation activations to approximately match the mean $\ell_0$ of fast inference and is held fixed during test evaluation. Since test sparsities can differ, we report the measured $\ell_0$ of each variant. Refinement introduces additional inference computation and is not applied to the baseline checkpoints. Training and model-selection details are provided in Appendix D. Our empirical comparison focuses on SAE methods trained without external complexity annotations. AdaptiveK (Yao et al., 2026), which trains a complexity predictor using LLM-generated annotations, falls outside this scope and is discussed in Related Work.

**Evaluation metrics.** For RQ1, we report test reconstruction error together with measured mean $\ell_0$, the number of nonzero latent coefficients per SAE input. Vision reconstruction is evaluated using standardized activation MSE per spatial position; language reconstruction uses the SAEBench reconstruction metric per token (Karvonen et al., 2025). Comparisons use measured sparsity rather than nominal training targets. Within each benchmark, an evaluated checkpoint is Pareto-optimal if no other evaluated checkpoint achieves both equal or lower reconstruction error and equal or lower

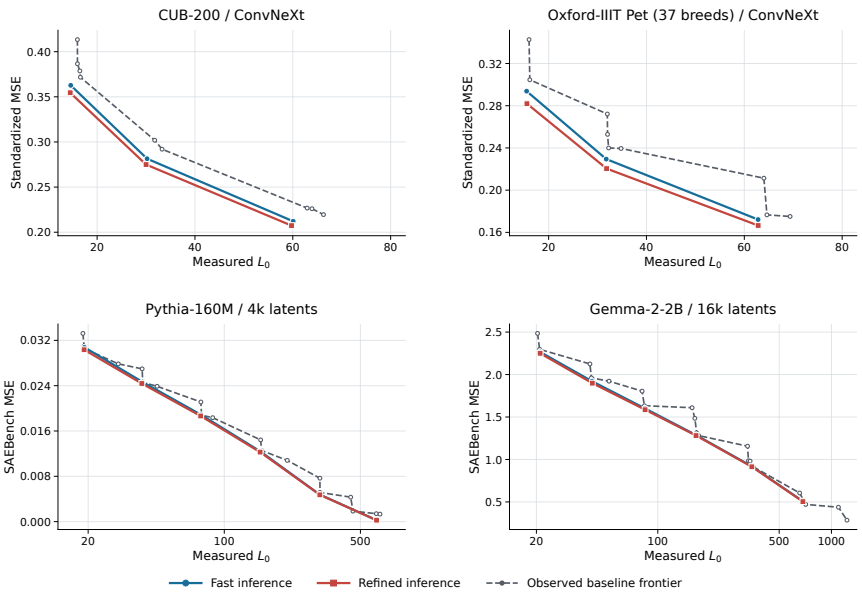


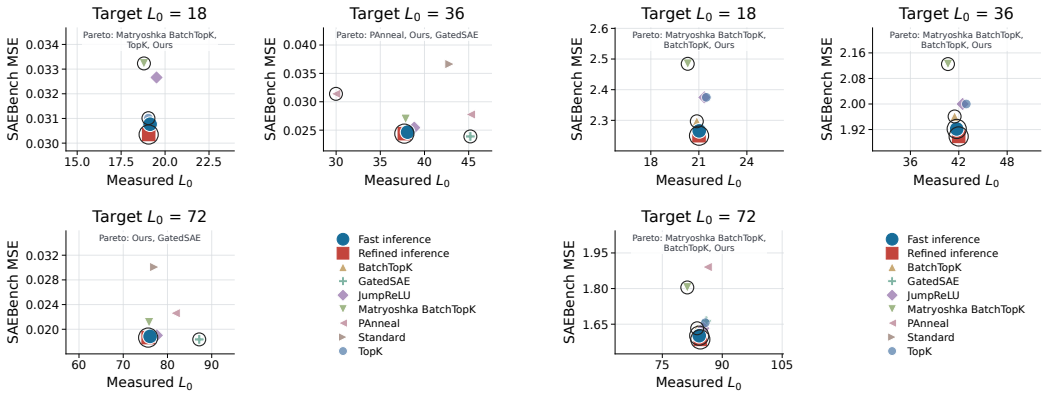

Figure 1: Test MSE reconstruction error versus measured mean $\ell_0$ (log scale for language) for vision (top) and language (bottom) tasks. Vision reports standardized activation MSE per spatial position; language reports the SAEBench reconstruction score per token. Fast and refined points share trained checkpoints. The dashed gray line connects the nondominated *observed* test values; they are not evaluated sparsity settings. Vision points are means over three SAE seeds with negligible standard deviation. Language points use one seed for our method and individual published baseline checkpoints.

mean $\ell_0$, with at least one strict improvement. For visual clarity, we aggregate nondominated baseline checkpoints across methods into a single observed frontier. Lines connecting these checkpoints are visual guides and do not represent additional measured operating points. Comparisons against interpolated baseline errors are reported separately from comparisons against observed checkpoints. For RQ2, we examine the distribution of active feature counts across spatial positions or tokens, including their means and percentiles. These measurements characterize input-dependent allocation. For RQ3, we replace the original activations with their SAE reconstructions and evaluate the remaining frozen model. For vision, we report top-1 agreement between predictions obtained from the original and reconstructed feature maps. For language, we report SAEBench cross-entropy loss recovery (Karvonen et al., 2025), which measures the fraction of the loss increase caused by ablating the original activations that is recovered by the SAE reconstruction. For RQ4, we compare reconstruction error, measured $\ell_0$, and SAE-only inference latency between fast and refined inference on the same frozen checkpoints.

### 4.2 Results

**Reconstruction at measured sparsity (RQ1).** Fig. 1 shows test reconstruction error against the observed mean number of active features. On the vision benchmarks, $S^2$-SAE improves the observed sparsity-reconstruction trade-off across the evaluated operating points. The local comparisons in the Appendix (Fig. 6a and 6b) show lower reconstruction errors than nearby baseline checkpoints at comparable measured sparsity. Both fast and refined inference perform well in these comparisons, indicating that the vision gains do not depend exclusively on post-hoc refinement.

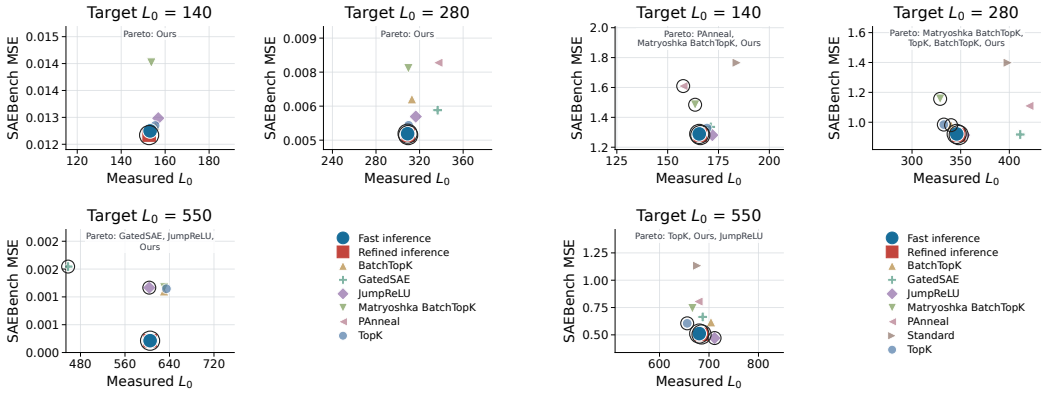
On the language benchmarks (Figures 2a, 2b, 2c, and 2d), the reconstruction gains depend on the model and sparsity level. Refined $S^2$-SAE contributes nondominated points to the observed sparsity-reconstruction frontier. However, nondominance among evaluated checkpoints does not establish superiority at exactly matched sparsity. The interpolated baseline comparisons in Appendix Tables 1 and 2 favor refined inference at all six Pythia settings and five of the six Gemma-2-2B settings. At




(a) Pythia (low $\ell_0$)




(b) Gemma-2-2B (low $\ell_0$)




(c) Pythia (high $\ell_0$)




(d) Gemma-2-2B (high $\ell_0$)

Figure 2: Local reconstruction comparisons around each trained sparsity target. Each panel shows the fast and refined $S^2$-SAE together with every observed baseline checkpoint in close range of the local $\ell_0$ level. Vision tasks show average values across three runs. Language tasks use one seed each.

the highest Gemma-2-2B target, refined inference achieves MSE $0.50391$ at mean $\ell_0 = 684.96$, compared with an interpolated baseline MSE of $0.49465$. Fast inference also trails the interpolated baseline at the three lowest Pythia targets (Table 3). These results indicate improvements across many evaluated settings rather than uniform superiority. At the highest Pythia target, refined inference achieves MSE $2.55 \times 10^{-4}$ at mean $\ell_0 = 605.12$. The estimated gain at this operating point is particularly sensitive to the spacing of baseline checkpoints and the interpolation procedure.

**Variation in allocated code length (RQ2).** Figure 3 shows how the number of active features varies across individual inputs. For the illustrated CUB fast-inference configuration with mean $\ell_0 = 30.2$ per spatial position, the fifth and 95th percentiles are $1$ and $108$, respectively. The average budget therefore permits substantial variation in the number of features used for individual activation vectors. Thus, the SAE is not constrained to assign the same count to every input. Interestingly, the distribution assumes different shapes depending on the task and model, e.g., CUB has a long-tail distribution, Pythia resembles a unimodal distribution, while Gemma-2-2B has a bimodal structure. These observations demonstrate that a shared threshold can produce different allocation distributions depending on the learned scores and input activations.

**Preservation of model behavior (RQ3).** Fig. 4 evaluates whether reconstructed activations preserve the behavior of the original frozen models. On CUB, top-1 agreement increases as more features are retained. On Oxford-IIIT Pet, agreement is already approximately $0.99$, leaving limited room for further improvement.

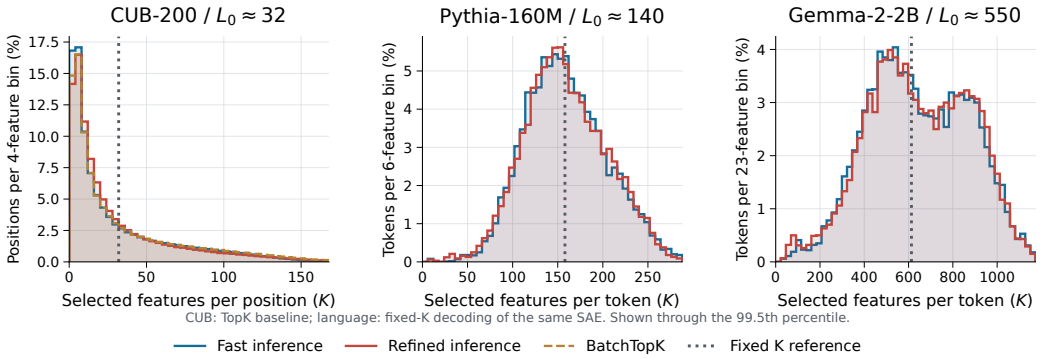



Figure 3: Distribution of the selected feature count $K$ at three representative sparsity levels. CUB counts features per spatial position; Pythia-160M and Gemma-2-2B count features per token. Fast and refined inference use the same trained SAE. The dotted line marks fixed TopK for CUB and fixed-$K$ decoding of the same SAE for language. Histograms are normalized over all evaluation samples, with the horizontal range ending at the 99.5th percentile.

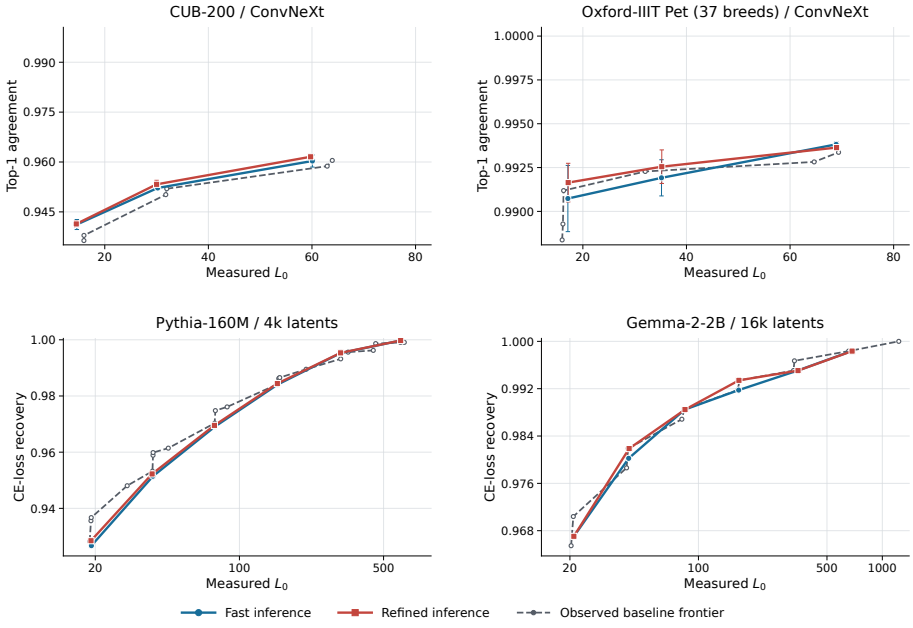



Figure 4: Behavior preservation versus measured mean $\ell_0$. Vision panels show top-1 agreement between predictions from original and reconstructed ConvNeXt feature maps; language panels show SAEBench cross-entropy loss recovery. Higher is better within each panel. The dashed line joins the best nondominated observed baseline checkpoints for each metric. Vision error bars show one standard deviation across three SAE seeds; language points use one seed for our method.

In language tasks, a lower reconstruction MSE does not necessarily match with better loss recovery. For example, even refined inference on Pythia, which achieved good MSE scores on low $\ell_0$, gets a lower loss recovery ($0.9285$ versus $0.9356$ at the lowest sparsity data point). The scores reach similar performance for high $\ell_0$, and generally competitive results on Gemma-2-2B, with some variation possibly attributable to individual run variance; nonetheless, we opted for following the SAEBench procedure strictly to ensure fair experimental conditions.

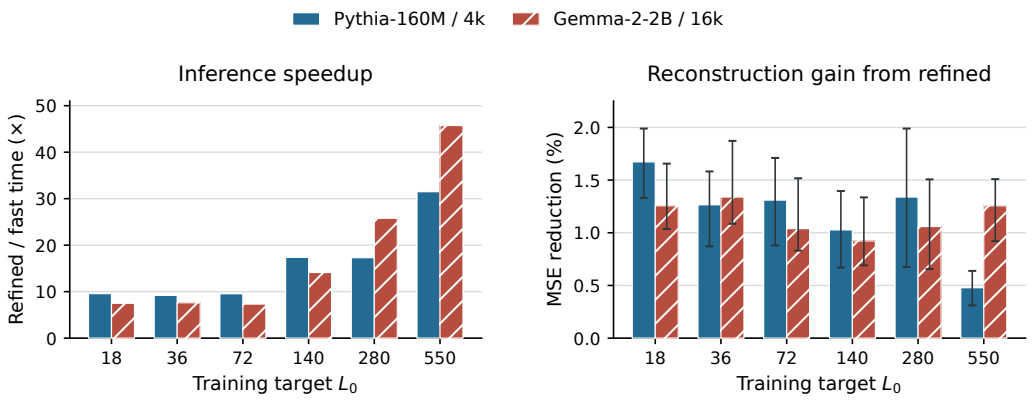



Figure 5: Fast versus refined inference on the same frozen SAE checkpoints for Pythia-160M and Gemma-2-2B. Left: SAE-only latency of refined inference relative to fast threshold inference. Right: percentage reduction in test reconstruction MSE from using refined inference; error bars show the variability of MSE across the test data. The two variants have close measured $\ell_0$ values.

**Fast versus refined inference (RQ4).** Figure 5 compares the two inference procedures on the same frozen SAE checkpoints. At approximately matched measured sparsity, refinement reduces reconstruction MSE by approximately $0.2$–$1.5\%$ across the evaluated language configurations. Small differences in test $\ell_0$ remain, so these improvements should not be interpreted as gains at exactly equal feature budgets. For example, at the highest Gemma-2-2B target, refinement uses approximately $4.70$ additional features per token on average. The additional reconstruction fidelity comes at a substantial inference cost. Refined inference is approximately 9–32 times slower than fast inference on Pythia and 7–46 times slower on Gemma-2-2B, with the largest overheads occurring at higher sparsity levels. These are SAE-only latency comparisons, not end-to-end model speedups. Detailed values are reported in Appendix Tables 5 and 6. Fast inference therefore provides most of the reconstruction quality achieved by the trained model, while refinement offers a smaller additional improvement when increased inference cost is acceptable.

## 5 Conclusion and Future Work

We introduced S$^2$-SAE, a framework connecting spike-and-slab sparse coding with adaptive sparsity in autoencoders. The Bayesian derivation assigns a shared cost to active features and establishes a threshold rule for coordinate-wise inference. This principle motivates a practical encoder with a learned global threshold and control over average sparsity, together with an optional post-hoc refinement of code length.

Experiments show improvements in the sparsity-reconstruction trade-off across many evaluated settings, with stronger gains on vision tasks and more mixed results on language benchmarks. Fast inference achieves most of the reconstruction quality available from the trained checkpoints, while refinement provides modest additional gains at substantially higher inference cost. However, lower reconstruction error does not consistently improve preservation of downstream model behavior. The language experiments use a single training seed per configuration; repeated runs are needed to establish the robustness of the observed gains.

Future work could examine how closely learned encoder scores approximate residual-dependent evidence and develop refinement procedures with lower computational overhead. The implications for concept discovery also remain open: our results demonstrate variable feature allocation and reconstruction benefits, but do not establish that the learned features are more semantically meaningful or interpretable. Direct evaluations of feature meaning and concept-level interventions would help assess whether adaptive sparsity supports understanding and manipulating neural representations.

## AI USE STATEMENT

Generative AI tools were used during the development of this work to assist with refinement of theoretical arguments, literature search, code development and debugging, and editing of the manuscript for clarity. All research ideas, theoretical claims, experimental designs, implementations, and results were reviewed and validated by the authors. The authors take full responsibility for the content of the paper.

## REPRODUCIBILITY STATEMENT

We release the implementation of $S^2$-SAE and the training and evaluation pipelines for our CUB and SAEBench experiments at `https://anonymous.4open.science/r/s2sae/`. The repository documents the experimental configurations, random seeds, data preprocessing, optimization settings, and commands needed to run the experiments. We use published benchmarks (SAEBench) and official train-test splits.

## REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. URL https://arxiv.org/abs/1308.3432.

Leonard Bereska and Efstratios Gavves. Mechanistic interpretability for AI safety – a review. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ePUVetPKu6.

Luca Bergamin, Roberto Confalonieri, and Fabio Aiolli. (sometimes) less is more: Mitigating the complexity of rule-based representation for interpretable classification. In *International Joint Conference on Neural Networks*, pp. 1–8, 2025. doi: 10.1109/IJCNN64981.2025.11228944.

Luca Bergamin, Shakiba Farjood Fashalami, and Fabio Aiolli. Addressing noise in annotations to preserve interpretability of concept-based models. In *Artificial Neural Networks and Machine Learning – ICANN 2026*, volume 17092 of *Lecture Notes in Computer Science*, pp. 415–427, Cham, 2026. Springer. doi: 10.1007/978-3-032-38407-2_33.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Trenton Bricken, Adly Templeton, Joshua Batson, et al. Towards monosemanticity: Decomposing language models with dictionary learning. Transformer Circuits Thread, 2023.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. In *arXiv preprint arXiv:2412.06410*, 2024. URL https://arxiv.org/abs/2412.06410.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6077–6101. PMLR, 2025.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023. doi: 10.48550/arXiv.2309.08600.

Juan Elenter, Luiz F. O. Chamon, and Alejandro Ribeiro. Near-optimal solutions of constrained learning problems. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2403.11844.

Nelson Elhage, Tristan Hume, Catherine Olsson, et al. Toy models of superposition. Transformer Circuits Thread, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Leo Gao, Tom Dupré la Tour, Henk Tillman, et al. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tcsZt9ZNKD.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL https://arxiv.org/abs/2408.00118.

Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Large-scale feature learning with spike-and-slab sparse coding. In *Proceedings of the 29th International Conference on Machine Learning*, 2012. URL https://icml.cc/2012/papers/718.pdf.

Adam Karvonen, Benjamin Wright, Can Rager, Rico Angell, Jannik Brinkmann, Logan Smith, Claudio Mayrink Verdun, David Bau, and Samuel Marks. Measuring progress in dictionary learning for language model interpretability with board game models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 83091–83118, 2024. doi: 10.52202/079017-2644.

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Isaac Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum Stuart Mcdougall, Kola Ayonrinde, Demian Till, Matthew Wearden, Arthur Conmy, Samuel Marks, and Neel Nanda. SAEBench: A comprehensive benchmark for sparse autoencoders in language model interpretability. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29223–29264. PMLR, 2025. URL https://proceedings.mlr.press/v267/karvonen25a.html.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International Conference on Machine Learning*, pp. 5338–5348. PMLR, 2020.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through $l_0$ regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012. URL https://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/.

Gonçalo Santos Paulo, Alex Troy Mallen, Caden Juang, and Nora Belrose. Automatically interpreting millions of features in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48393–48421. PMLR, 2025. URL https://proceedings.mlr.press/v267/paulo25a.html.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Krámár, Rohin Shah, and Neel Nanda. Improving sparse decomposition of language model activations with gated sparse autoencoders. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. doi: 10.52202/079017-0024.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with JumpReLU sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024b. doi: 10.48550/arXiv.2407.14435.

Adly Templeton, Tom Conerly, Jonathan Marcus, et al. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. Transformer Circuits Thread, 2024.

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Yifei Yao, Hanrong Zhang, and Mengnan Du. AdaptiveK: Complexity-driven sparse autoencoders for interpretable language model representations. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23702–23728, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1187. URL https://aclanthology.org/2026.findings-acl.1187/.

# A THEORETICAL DERIVATION OF THE TRAINING OBJECTIVE

## A.1 OBSERVATION MODEL

Let $\boldsymbol{x} \in \mathbb{R}^d$ be a neural representation and let $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_m] \in \mathbb{R}^{d \times m}$ be a decoder dictionary. We consider nonnegative coefficients $z \in \mathbb{R}^m_{\geq 0}$ and assume unit-norm atoms, $\|\boldsymbol{d}_j\|_2 = 1$. Given a code $\boldsymbol{z}$, the observation model is

$$\boldsymbol{x} \mid \boldsymbol{z}, D \sim \mathcal{N}(\boldsymbol{D}\boldsymbol{z}, \sigma^2 \boldsymbol{I}_d). \tag{5}$$

The unit-norm convention matters: without it, rescaling a dictionary atom and its coefficient would change the apparent size of a threshold without changing the reconstruction.

## A.2 SPIKE-AND-SLAB LATENT CODE

For each feature $j$, independently draw a binary inclusion variable and a nonnegative magnitude,

$$s_j \sim \text{Bernoulli}(\pi), \qquad u_j \sim \text{HalfNormal}(\alpha), \qquad z_j = s_j u_j, \tag{6}$$

where $0 < \pi < 1$ and $\alpha > 0$. To make the parameterization explicit, the half-normal density is

$$p(u_j) = c_\alpha \exp\left(-\frac{\alpha u_j^2}{2}\right) \mathbf{1}[u_j \geq 0], \tag{7}$$

where $c_\alpha$ is the normalizing constant. We treat $\alpha$ as fixed in the derivation. The vector $\boldsymbol{u}$ is drawn for every feature, independently of $\boldsymbol{s}$, including features for which $s_j = 0$. Marginally, $z_j$ has a spike at zero and a positive continuous slab. This construction is useful because the normalization constant of the half-normal density appears once for every coordinate and is therefore independent.

## A.3 FROM THE POSTERIOR TO A PENALIZED OBJECTIVE

We obtain the MAP objective by writing the negative log of each factor. First, substituting $\boldsymbol{z} = \boldsymbol{s} \odot \boldsymbol{u}$ into the Gaussian observation model gives

$$-\log p(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{D}) = \frac{\|\boldsymbol{x} - \boldsymbol{D}(\boldsymbol{s} \odot \boldsymbol{u})\|_2^2}{2\sigma^2} + C_x. \tag{8}$$

Second, independence of the half-normal magnitudes gives

$$-\log p(\boldsymbol{u}) = \frac{\alpha}{2}\|\boldsymbol{u}\|_2^2 + C_u. \tag{9}$$

Third, the negative log Bernoulli prior is

$$-\log p(\boldsymbol{s}) = -\sum_{j=1}^{m} \big[s_j \log \pi + (1 - s_j)\log(1 - \pi)\big] = \rho \sum_{j=1}^{m} s_j + C_s, \qquad \rho = \log \frac{1-\pi}{\pi}. \tag{10}$$

Adding these three terms and discarding $C_x + C_u + C_s$ gives the joint MAP problem

$$\min_{\substack{\boldsymbol{s} \in \{0,1\}^m \\ \boldsymbol{u} \geq \boldsymbol{0}}} \frac{\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2}{2\sigma^2} + \frac{\alpha}{2}\|\boldsymbol{u}\|_2^2 + \rho \sum_{j=1}^{m} s_j, \qquad \boldsymbol{z} = \boldsymbol{s} \odot \boldsymbol{u}, \qquad \rho = \log \frac{1-\pi}{\pi}. \tag{11}$$

The slab normalization does not enter $\rho$ because $u_j$ is drawn whether or not feature $j$ is active.

Assume $\rho > 0$, equivalently $\pi < 1/2$, as expected under a sparse prior. At a MAP solution, an inactive feature has $u_j = 0$: when $s_j = 0$, its value does not affect the reconstruction, so the quadratic prior is minimized at zero. Conversely, the state $s_j = 1, u_j = 0$ is dominated by $s_j = 0, u_j = 0$, which has the same reconstruction and a lower activation cost. Thus $\sum_j s_j = \|\boldsymbol{z}\|_0$ and $\|\boldsymbol{u}\|_2^2 = \|\boldsymbol{z}\|_2^2$ at the optimum. The joint problem can therefore be written directly in terms of the code:

$$\hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z} \geq 0} \left\{ \frac{\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{z}\|_2^2}{2\sigma^2} + \frac{\alpha}{2}\|\boldsymbol{z}\|_2^2 + \rho\|\boldsymbol{z}\|_0 \right\}, \tag{12}$$

where $\|\boldsymbol{z}\|_0$ counts nonzero coefficients. This is the theoretical MAP objective. The implementation does not estimate $\pi$, $\sigma^2$, or $\alpha$; it uses the resulting decision structure to motivate a trainable threshold, as described below.

### A.4 BROAD-SLAB LIMIT

When $\alpha$ is small enough that the magnitude penalty is weak on the scale of the reconstruction term, the corresponding penalized problem is approximately

$$\min_{\boldsymbol{z}\geq 0}\left\{\frac{\|\boldsymbol{x}-\boldsymbol{D}\boldsymbol{z}\|_2^2}{2\sigma^2}+\rho\|\boldsymbol{z}\|_0\right\}. \tag{13}$$

This is a useful sparse-coding reference objective. In the masked construction above, taking a broad slab weakens the coefficient penalty without adding a slab-normalization term to $\rho$.

### A.5 THE EXACT COORDINATE-WISE DECISION

Fix all coefficients except $z_j$ and define the residual excluding feature $j$ as

$$\boldsymbol{r}_{-j}=\boldsymbol{x}-\sum_{\ell\neq j}\boldsymbol{d}_\ell \boldsymbol{z}_\ell. \tag{14}$$

For this coordinate, the objective is

$$F_j(t)=\frac{\|\boldsymbol{r}_{-j}-\boldsymbol{d}_j t\|_2^2}{2\sigma^2}+\frac{\alpha t^2}{2}+\rho\mathbf{1}[t>0],\qquad t\geq 0. \tag{15}$$

Let $b_j=\boldsymbol{d}_j^\top \boldsymbol{r}_{-j}$ and $c_j=(b_j)_+$. If $b_j\leq 0$, a positive coefficient cannot reduce the continuous objective. If $b_j>0$, the optimal active magnitude and its improvement over $t=0$ are

$$z_j^{\mathrm{on}}=\frac{c_j}{1+\alpha\sigma^2},\qquad \Delta_j=\frac{c_j^2}{2\sigma^2(1+\alpha\sigma^2)}. \tag{16}$$

The feature is activated precisely when $\Delta_j>\rho$. Consequently,

$$z_j=\frac{c_j}{1+\alpha\sigma^2}\mathbf{1}[c_j>\tau],\qquad c_j=(\boldsymbol{d}_j^\top \boldsymbol{r}_{-j})_+, \tag{17}$$

where

$$\tau=\sqrt{2\sigma^2(1+\alpha\sigma^2)\rho}. \tag{18}$$

For the broad-slab objective in Eq. 13, the rule reduces to

$$z_j=c_j\mathbf{1}[c_j>\sqrt{2\sigma^2\rho}]. \tag{19}$$

Eq. 17 holds for any unit-normalized dictionary, including an overcomplete one. The threshold is shared because every active feature pays the same cost $\rho$. What makes inference difficult are the dependencies: $c_j$ depends on $\boldsymbol{r}_{-j}$, which depends on the other active features and their magnitudes. Custom inference algorithms could seek a coordinate-wise solution, but the $\ell_0$ objective remains nonconvex and hard to optimize.

### A.6 THE ORTHOGONAL REFERENCE CASE

If $\boldsymbol{D}^\top\boldsymbol{D}=\boldsymbol{I}_m$, then $\boldsymbol{d}_j^\top \boldsymbol{r}_{-j}=\boldsymbol{d}_j^\top \boldsymbol{x}$ for every configuration of the other coefficients. All coordinates decouple, and simultaneous thresholding gives the global MAP solution:

$$\hat{z}_j=\frac{(\boldsymbol{d}_j^\top \boldsymbol{x})_+}{1+\alpha\sigma^2}\mathbf{1}[(\boldsymbol{d}_j^\top \boldsymbol{x})_+>\tau]. \tag{20}$$

This case explains the threshold structure, but it is not an assumption about the trained SAE. In the usual overcomplete setting $m>d$, an orthogonal dictionary is impossible because $\mathrm{rank}(\boldsymbol{D}^\top\boldsymbol{D})\leq d<m$.

### A.7 FROM COUPLED INFERENCE TO AN AMORTIZED ENCODER

For a general dictionary, each exact coordinate decision uses a residual that changes with the other decisions. The SAE replaces this iterative per-example computation with learned scores

$$a_\phi(\boldsymbol{x})=\mathrm{ReLU}(\boldsymbol{W}_e\boldsymbol{x}+\boldsymbol{b}_e). \tag{21}$$

We interpret $a_{\phi,j}(\boldsymbol{x})$ as a score trained to play the role of the residual-aware evidence $(\boldsymbol{d}_j^\top \boldsymbol{r}_{-j})_+$. This is an amortized proxy. The architectural choice retained from MAP inference is to compare feature evidence against a shared absolute threshold.

# B IMPLEMENTATION DETAILS

## B.1 HARD FORWARD PASS AND STRAIGHT-THROUGH GRADIENTS

Because the hard gate has zero derivative almost everywhere, training uses the surrogate

$$p_r = \mathrm{sigmoid}\left(\frac{v_r - \tau}{\delta}\right), \qquad \delta = \max\{\beta \ \mathrm{stopgrad}(\tau), \delta_{\min}\}, \tag{22}$$

and the straight-through gate

$$g_r = h_r + p_r - \mathrm{stopgrad}(p_r). \tag{23}$$

Its forward value is exactly $h_r$; its backward derivative follows $p_r$. Thus reconstruction during training uses a hard, variable-cardinality code while gradients can reach the encoder and learned threshold. The relative bandwidth $\beta$ and floor $\delta_{\min}$ are optimization hyperparameters with no Bayesian interpretation. They affect only the backward surrogate and can be tuned if training is unstable or unusually sensitive; they do not change the hard inference rule.

## B.2 TRAINING OBJECTIVE AND BUDGET CONTROL

Let $\widetilde{\ell}_0(x) = \sum_{r=1}^{C} g_r \mathbf{1}[v_r > 0]$ denote the straight-through estimate of effective cardinality, and let $\ell_0(x)$ count strictly positive coefficients in the hard code. The practical objective is

$$\mathcal{L}_{\mathrm{train}} = \mathcal{L}_{\mathrm{rec}} + \theta_{\text{sparsity}} \left( \frac{\mathbb{E}_{\mathrm{batch}}[\widetilde{\ell}_0(\boldsymbol{x})]}{K_{\text{target}}} - 1 \right) + \gamma \mathcal{L}_{\mathrm{AuxK}}. \tag{24}$$

Here $\mathcal{L}_{\mathrm{rec}}$ is hard-code reconstruction MSE, $\mathcal{L}_{\mathrm{AuxK}}$ is an auxiliary reconstruction term for dead features. After an optimizer step, the nonnegative dual variable is updated using the measured hard cardinality:

$$\theta_{\text{sparsity}} \leftarrow \left[ \theta_{\text{sparsity}} + \eta_{\text{sparsity}} \left( \frac{\mathbb{E}_{\mathrm{batch}}[\ell_0(x)]}{K_{\text{target}}} - 1 \right) \right]_+ . \tag{25}$$

The auxiliary loss is computed only from features considered dead by the training monitor. Let $\mathcal{D}$ be that set, let $A \subseteq \mathcal{D}$ contain the dead features with the largest nonnegative encoder scores for the current input, and let $r = \mathrm{stopgrad}(x - \hat{x})$ be the detached reconstruction residual. The implementation uses

$$\mathcal{L}_{\mathrm{AuxK}} = \frac{1}{d} \left\| \boldsymbol{r} - \sum_{j \in A} \boldsymbol{d}_j a_{\phi,j}(\boldsymbol{x}) \right\|_2^2 . \tag{26}$$

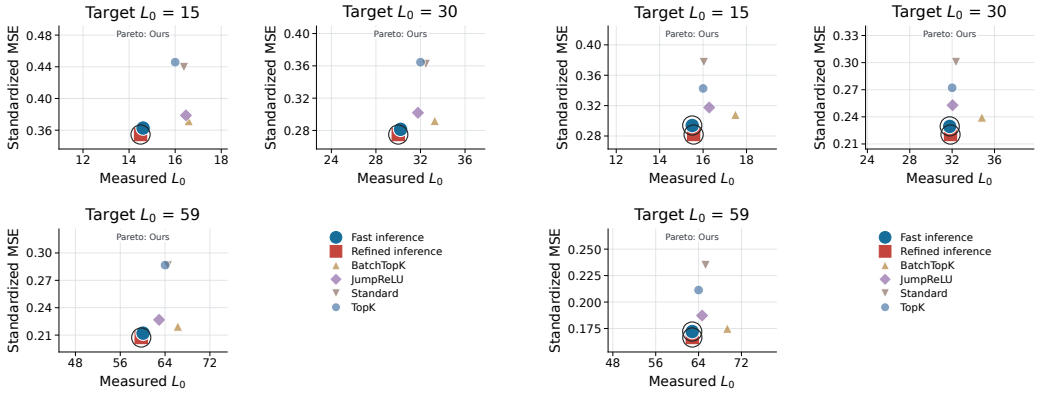
This term gives dead features a training signal. It is an optimization aid and is not derived from the spike-and-slab formulation.

## B.3 UNIT-NORM DECODER PARAMETERIZATION

The decoder is initialized as $\boldsymbol{D} = \mathrm{normalize}(\boldsymbol{W}_e^\top)$, the encoder weight is then set to $\boldsymbol{W}_e = \boldsymbol{D}^\top$, and the encoder bias is initialized to zero. Encoder and decoder parameters are untied thereafter. Before each optimizer step, the decoder gradient $\boldsymbol{G}_j$ is normalized and each $\boldsymbol{d}_j$ is renormalized after the step. This keeps $\|\boldsymbol{d}_j\|_2 = 1$ and removes decoder scale ambiguity.

# C ADDITIONAL RESULTS

## C.1 ADDITIONAL VISUALIZATIONS




(a) CUB (b) Oxford

Figure 6: Local reconstruction comparisons around each trained sparsity target. Each panel shows the fast and refined $S^2$-SAE together with every observed baseline checkpoint in close range of the local $\ell_0$ level. Vision tasks show average values across three runs. Language tasks use one seed each.

### C.2 SPARSITY-RECONSTRUCTION TRADE-OFF (REFINED)

| Sparsity target | Measured $\ell_0$ | Refined MSE | Baseline MSE | Best baseline | MSE gain |
|---|---|---|---|---|---|
| 18 | 19.067 | 0.030349 | 0.031005 | TopK | $+2.12\%$ |
| 36 | 37.720 | 0.024384 | 0.024682 | BatchTopK | $+1.21\%$ |
| 72 | 75.634 | 0.018650 | 0.018868 | BatchTopK | $+1.16\%$ |
| 140 | 152.708 | 0.012273 | 0.012714 | TopK | $+3.47\%$ |
| 280 | 309.528 | 0.004724 | 0.005168 | TopK | $+8.59\%$ |
| 550 | 605.117 | $2.5503\times10^{-4}$ | $1.6145\times10^{-3}$ | BatchTopK | $+84.20\%$ |

Table 1: Refined inference against the strongest interpolated published baseline for Pythia-160M with 4,096 SAE latents. Positive MSE gain means that our method has lower reconstruction error.

| Sparsity target | Measured $\ell_0$ | Refined MSE | Baseline MSE | Best baseline | MSE gain |
|---|---|---|---|---|---|
| 18 | 21.012 | 2.25000 | 2.29363 | BatchTopK | $+1.90\%$ |
| 36 | 41.985 | 1.89844 | 1.95532 | BatchTopK | $+2.91\%$ |
| 72 | 84.510 | 1.58594 | 1.62846 | BatchTopK | $+2.61\%$ |
| 140 | 166.044 | 1.28125 | 1.29885 | JumpReLU | $+1.36\%$ |
| 280 | 348.005 | 0.91406 | 0.92236 | JumpReLU | $+0.90\%$ |
| 550 | 684.964 | 0.50391 | 0.49465 | JumpReLU | $-1.87\%$ |

Table 2: Refined inference against the strongest interpolated published baseline for Gemma-2-2B with 16,384 SAE latents. The interpolation and gain definitions match Table 1.

### C.3 SPARSITY-RECONSTRUCTION TRADE-OFF (FAST)

| Sparsity target | Measured $\ell_0$ | Learned-threshold MSE | Baseline MSE | Best baseline | MSE gain |
|---|---|---|---|---|---|
| 18 | 19.149 | 0.030766 | 0.030534 | BatchTopK | $-0.76\%$ |
| 36 | 38.087 | 0.024646 | 0.024599 | BatchTopK | $-0.19\%$ |
| 72 | 76.074 | 0.018849 | 0.018820 | BatchTopK | $-0.15\%$ |
| 140 | 153.153 | 0.012397 | 0.012688 | TopK | $+2.30\%$ |
| 280 | 308.996 | 0.004785 | 0.005186 | TopK | $+7.73\%$ |
| 550 | 605.165 | $2.5565\times10^{-4}$ | $1.6139\times10^{-3}$ | BatchTopK | $+84.16\%$ |

Table 3: Native learned-threshold inference against the strongest interpolated published baseline for Pythia-160M with 4,096 SAE latents. Positive MSE gain means that our method has lower reconstruction error.

| Sparsity target | Measured $\ell_0$ | Learned-threshold MSE | Baseline MSE | Best baseline | MSE gain |
|---|---|---|---|---|---|
| 18 | 21.024 | 2.26562 | 2.29335 | BatchTopK | $+1.21\%$ |
| 36 | 41.744 | 1.92188 | 1.95801 | BatchTopK | $+1.85\%$ |
| 72 | 84.239 | 1.60156 | 1.62995 | BatchTopK | $+1.74\%$ |
| 140 | 165.521 | 1.28906 | 1.30044 | JumpReLU | $+0.87\%$ |
| 280 | 345.924 | 0.92188 | 0.92541 | JumpReLU | $+0.38\%$ |
| 550 | 680.263 | 0.51172 | 0.49902 | JumpReLU | $-2.55\%$ |

Table 4: Native learned-threshold inference against the strongest interpolated published baseline for Gemma-2-2B with 16,384 SAE latents. The interpolation and gain definitions match Table 3.

### C.4 FAST INFERENCE

| Sparsity target | $\ell_0$ | $\ell_0$ (fast) | $\Delta\ell_0$ | MSE | MSE (fast) | $\Delta$ MSE | Fast speedup |
|---|---|---|---|---|---|---|---|
| 18 | 19.067 | 19.149 | $-0.082$ | 0.030349 | 0.030766 | $-1.35\%$ | $9.6\times$ |
| 36 | 37.720 | 38.087 | $-0.367$ | 0.024384 | 0.024646 | $-1.07\%$ | $9.2\times$ |
| 72 | 75.634 | 76.074 | $-0.440$ | 0.018650 | 0.018849 | $-1.05\%$ | $9.5\times$ |
| 140 | 152.708 | 153.153 | $-0.445$ | 0.012273 | 0.012397 | $-1.00\%$ | $17.4\times$ |
| 280 | 309.528 | 308.996 | $+0.532$ | 0.004724 | 0.004785 | $-1.29\%$ | $17.3\times$ |
| 550 | 605.117 | 605.165 | $-0.047$ | $2.5503\times10^{-4}$ | $2.5565\times10^{-4}$ | $-0.24\%$ | $31.5\times$ |

Table 5: Refined versus fast native learned-threshold inference for Pythia-160M. Negative $\Delta$MSE means that refined inference improves reconstruction. Speedup is the median batch-64 SAE-only inference speedup of the fast method, measured on the same checkpoint and GPU.

| Sparsity target | $\ell_0$ | $\ell_0$ (fast) | $\Delta\ell_0$ | MSE | MSE (fast) | $\Delta$ MSE | Fast speedup |
|---|---|---|---|---|---|---|---|
| 18 | 21.012 | 21.024 | $-0.012$ | 2.25000 | 2.26562 | $-0.69\%$ | $7.5\times$ |
| 36 | 41.985 | 41.744 | $+0.241$ | 1.89844 | 1.92188 | $-1.22\%$ | $7.6\times$ |
| 72 | 84.510 | 84.239 | $+0.271$ | 1.58594 | 1.60156 | $-0.98\%$ | $7.3\times$ |
| 140 | 166.044 | 165.521 | $+0.523$ | 1.28125 | 1.28906 | $-0.61\%$ | $14.1\times$ |
| 280 | 348.005 | 345.924 | $+2.081$ | 0.91406 | 0.92188 | $-0.85\%$ | $25.8\times$ |
| 550 | 684.964 | 680.263 | $+4.701$ | 0.50391 | 0.51172 | $-1.53\%$ | $45.7\times$ |

Table 6: Refined versus fast native learned-threshold inference for Gemma-2-2B. Differences and speedup are defined as in Table 5.

## D OPTIMIZATION AND HYPERPARAMETER SWEEP

For each vision dataset, we initialize ConvNeXt-Tiny from ImageNet-pretrained weights, fine-tune it for 60 epochs, and freeze the checkpoint with the highest validation accuracy. We train SAEs on its standardized spatial activations at target sparsities at around $\ell_0 \in \{16, 32, 64\}$, using three different seeds and batches of 128 activation vectors. Vision SAEs use Adam with an initial learning rate of $10^{-3}$, reduced when the validation objective plateaus. Training runs for at most 500 epochs, with validation-based checkpoint selection and early stopping.

For the completed language comparison, we train at target sparsities $\ell_0 \in \{18, 36, 72, 140, 280, 550\}$. Each configuration starts from a fresh initialization, 500 million token activations, and logical batches of 2,048 tokens. Pythia-160M uses a dictionary width of $m = 4{,}096$; Gemma-2-2B is evaluated at $m = 16{,}384$. Language SAEs use Adam with a peak learning rate of $3 \times 10^{-4}$, a 1,000-step warmup, and linear decay near the end of training.

For the normalized learned-threshold SAE, decoder atoms are initialized by normalizing the transposed encoder weights. The encoder is then set to the decoder transpose and its bias to zero; encoder and decoder are trained separately after that. The initial global threshold is estimated from the first 8,192 training activations, and decoder atoms are renormalized after optimizer updates. For the vision $\ell_1$ and JumpReLU baselines, we train separate runs with different sparsity coefficients. Within each run, we retain the checkpoint minimizing validation reconstruction MSE plus its fixed sparsity penalty. We then choose the coefficient whose checkpoint has validation $\ell_0$ closest to the target, breaking ties by validation MSE. For the other vision SAEs, we retain the checkpoint with the lowest validation reconstruction MSE. Language runs use the final checkpoint after the fixed token budget.

For each trained SAE, we select a single $\eta$ on validation activations so that refined inference has a mean $\ell_0$ as close as possible to that of fast inference. We then hold $\eta$ fixed for test evaluation. For all reported $S^2$-SAE runs, we set the STE bandwidth parameters to $\beta = 0.2$ and $\delta_{\min} = 10^{-3}$, considered all $m$ features as threshold candidates ($C = m$), allowed zero active features, and used AuxK weight $\gamma = 1/32$ with 64 auxiliary features for vision, 384 for Pythia-160M, and 1152 for Gemma-2-2B.