

Article

FISTA-Refined Dictionary Learning with Learned TopK Initialization for Sparse Decomposition of Transformer Residual Streams

Kyung-In Joung and Jang Pyo Bae * 

Division of Healthcare Science, College of Future Convergence, CHA University,
Pocheon 11160, Republic of Korea; jki0515@cha.ac.kr

* Correspondence: haizzzi@cha.ac.kr; Tel.: +82-31-850-8956

Abstract

Sparse autoencoders (SAEs) decompose transformer activations into sparse feature dictionaries, but one-shot support selection can leave many features unused and become difficult to optimize at very small active budgets. We study an SAE-like dictionary-learning procedure that combines a learned TopK encoder with a few unrolled Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) steps for coefficient refinement. On GPT-2 Small layer-11 residual streams, a 12,288-feature TopK256 model with five FISTA steps achieves $\Delta\text{CE} = 0.0745$ and a 3.09% dead-feature ratio, compared with 0.0859 and 50.76% for a matched TopK SAE. Ablations show that initialization and learned TopK feature selection explain most of the reduction in dead features, while FISTA further improves fidelity. In a 32,768-feature regime, direct exact-TopK32 training is unstable. Training becomes feasible by retaining a TopK128 candidate support while reducing the final active budget from TopK128 to TopK64 and then TopK32. Five refinement steps reach $\Delta\text{CE} = 0.2442$ and 17.84% dead features after 264 million token presentations. A residual-gain-ranked loss over candidates at ranks 33–48 improves near-boundary feature recruitment with a modest fidelity cost. These results show that lightweight iterative correction can improve fixed-budget sparse decomposition without replacing one-shot SAE inference with a full optimization solver.

Keywords: sparse autoencoder; dictionary learning; FISTA; TopK sparsity; mechanistic interpretability



Academic Editor: Roberto Montemanni

Received: 18 July 2026

Revised: 17 August 2026

Accepted: 19 August 2026

Published: 20 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Mechanistic interpretability seeks to explain model behavior in terms of internal computational elements rather than only input–output statistics. Transformer residual streams are a particularly useful substrate because they aggregate contributions from attention and MLP sublayers and can be cached, reconstructed, and causally patched through hook-based tooling such as TransformerLens [1,2].

A central difficulty is superposition: more potentially meaningful features may be represented than there are model dimensions, so individual neurons or basis directions can respond to multiple unrelated patterns. Sparse dictionary learning and sparse autoencoders (SAEs) address this problem by reconstructing dense activations from a sparse combination of atoms in an overcomplete dictionary. Classical sparse coding established that learned sparse dictionaries can recover structured factors, and recent work has applied the same

principle to transformer activations, reporting semantically coherent features and finer causal localization than neuron-level analyses [3–6].

Modern SAE research has therefore focused not only on reconstruction error, but also on exact sparsity control and dictionary utilization. TopK autoencoders, including the original k-sparse autoencoder formulation, directly enforce a fixed active-feature budget [7], while large-scale studies emphasize decoder normalization, initialization, auxiliary losses, and dead-latent mitigation [8]. Gated SAEs separate support selection from activation magnitude [9], JumpReLU learns activation thresholds [10], BatchTopK reallocates sparse capacity across a batch [11], and end-to-end dictionary learning optimizes features for downstream functional objectives [12]. At the same time, benchmark and critique work shows that reconstruction, sparsity, automated interpretability, and downstream usefulness are related but non-equivalent properties [13–17].

Most practical SAEs nevertheless infer their sparse code in a single encoder pass followed by TopK selection, gating, or thresholding. This amortized inference is efficient, but the resulting code is an encoder prediction rather than the solution of a reconstruction problem for the current activation and dictionary. Classical dictionary learning instead separates dictionary optimization from per-example coefficient inference [18,19]. Recent work has formalized the discrepancy between SAE encoder outputs and more directly optimized sparse codes as an amortization gap [20].

FISTA provides a natural way to test whether a small amount of optimization can correct this one-shot prediction. FISTA is an accelerated proximal-gradient method for smooth reconstruction objectives combined with nonsmooth sparsity penalties [18], and it has previously been used to infer nonnegative sparse coefficients over transformer factors [4]. Our aim is not to replace the encoder with a long-running sparse-coding solver. Instead, the learned TopK encoder proposes a candidate support, and only a few unrolled FISTA steps refine the coefficient magnitudes before an exact final TopK projection. When $\lambda = 0$, these steps act as reconstruction-driven nonnegative coefficient correction without additional ℓ_1 shrinkage. We use the term FISTA-style refinement because the updates retain the accelerated momentum and step-size structure of FISTA; when $\lambda = 0$, the shrinkage operator reduces to nonnegative projection.

The ultra-sparse TopK32 setting raises a second problem. When the candidate and final supports are both restricted to 32 features, only a very small set of coefficients contributes directly to each reconstruction, which can repeatedly reinforce already selected features and make it difficult for nearby alternatives to enter the support. We therefore separate candidate generation from final sparsity: a TopK128 candidate set is refined and then projected to an exact TopK32 output through a shared TopK128→TopK64→TopK32 curriculum. This design allows the model to consider a wider near-boundary feature set while preserving the final active budget.

We further introduce a near-miss ranking objective for candidates immediately below the TopK32 boundary. The loss compares weak active features with positive-gain candidates at ranks 33–48 and promotes either the four candidates with the largest detached residual gain or four randomly selected candidates from the same positive-gain pool. To the best of our knowledge, prior SAE work has not used a residual-gain-ranked pairwise objective to train the one-shot support selector specifically at the exact TopK boundary. This loss differs from AuxK-style dead-latent reconstruction losses [8,21]. AuxK trains dead latents through an auxiliary reconstruction of the main residual, thereby providing gradients to their encoder and decoder parameters. In contrast, our auxiliary gradients flow only through the selected initializer-score entries, while the dictionary and FISTA refinement path are detached. Like AuxK, the proposed loss does not directly modify the exact TopK32 reconstruction produced by the main branch.

The paper makes three empirical contributions. First, it separates the effects of initialization and learned support selection from the effects of iterative coefficient refinement in a matched 12,288-feature TopK256 benchmark. Second, it shows that candidate-to-target training and a small number of $\lambda = 0$ FISTA steps substantially accelerate learning in a 32,768-feature exact TopK32 regime. Third, it demonstrates that convergence is sensitive to a narrow group of near-boundary features and evaluates a gain-ranked loss for recruiting those features. All models are evaluated by activation patching with Δ CE, logit-difference preservation, effective L0, dead-feature ratio, and feature-usage entropy.

2. Materials and Methods

2.1. Model, Data, and Activation Source

All main experiments use GPT-2 Small loaded through TransformerLens as a Hooked-Transformer, with the target hidden representation taken from the residual stream at blocks.11.hook_resid_post. GPT-2 is a standard decoder-only transformer language model, and TransformerLens provides the hookable forward interface used for both training-time activation extraction and evaluation-time causal patching [1,2,22].

Two experimental regimes are reported. The original 12,288-feature benchmark retains its one-epoch protocol over 24 million GPT-2 tokens, using sequence length 128 and batch size 8. The ultra-sparse experiments use $K = 32,768$ and a cached set of residual-stream activations corresponding to 24 million training tokens per epoch. The same cached token set is reshuffled and reused across epochs, with sequence length 128 and batch size 160. For the cached 32,768-feature experiments, the activation corpus was reshuffled using a fixed cache seed of 0 and an epoch-dependent deterministic generator, so the data order was reproducible across runs. In all ultra-sparse runs, the learned initializer forms a candidate TopK128 code, whereas the target active budget follows a shared curriculum: TopK128 in epoch 1, TopK64 in epoch 2, and TopK32 from epoch 3 onward. Evaluation uses 200,000 WikiText-103 validation tokens at sequence length 128. For trajectory analyses, each saved model is evaluated using the target TopK active at its corresponding training epoch: TopK128 at epoch 1, TopK64 at epoch 2, and TopK32 thereafter. The project uses Hugging Face's wikitext-103-raw-v1 representation of WikiText [23]. All experiments were implemented in Python 3.12.13 using PyTorch 2.7.1+cu126 and TransformerLens 3.4.0. The 12,288-feature experiments were run on an NVIDIA A100-SXM4 GPU with 40 GB of memory, and the cached 32,768-feature experiments were run on an NVIDIA RTX PRO 6000 Blackwell GPU with 96 GB of memory. For the 32,768-feature experiments, the batch size of 160 was selected conservatively after preliminary GPU-memory stress testing so that the largest refinement-step configurations used in the study could be executed without memory failure.

2.2. Candidate-to-Target SAE-like Dictionary-Learning Architecture

The proposed SAE-like dictionary-learning model decomposes a transformer residual-stream activation $\mathbf{h} \in \mathbb{R}^d$ into sparse nonnegative coefficients over a learned dictionary $D \in \mathbb{R}^{d \times K}$. Here $d = 768$ and K is either 12,288 in the original benchmark or 32,768 in the ultra-sparse experiments. The learned encoder first proposes a wider candidate code and optional FISTA iterations refine its coefficients. A second exact TopK projection then enforces the smaller target active budget used for reconstruction and downstream activation patching. Thus, candidate support generation and final sparsity enforcement are treated as separate operations. The reconstruction is $\hat{\mathbf{h}} = D\mathbf{a}$, and the dictionary columns are renormalized after every optimizer step. The decoder dictionary is initialized elementwise from a Gaussian distribution with standard deviation $1/\sqrt{d}$ [4,18,19].

A standard TopK SAE uses a learned encoder to produce latent scores, retains the TopK entries as the sparse code, and reconstructs the input through a learned linear decoder. Our model retains this encoder–dictionary structure but uses the encoder output as the starting point for optional reconstruction-driven coefficient refinement. The learned encoder first determines a candidate support, FISTA-style iterations adjust coefficient magnitudes on that support using the current dictionary and input activation, and a final exact TopK projection determines the code used for reconstruction. When the number of refinement steps is zero, the iterative refinement is absent and inference reduces to the model’s one-shot TopK encoder–decoder path. This zero-step condition is not identical to the separately trained standard SAE baseline because the proposed model uses different encoder/dictionary initialization and dictionary normalization during training. These differences are examined empirically in the 12,288-feature benchmark and the SAE initialization controls.

Let $\mathbf{z}$ denote the nonnegative scores produced by the learned initializer:

$$\mathbf{z} = \text{ReLU}(W_{\text{enc}}\mathbf{h} + \mathbf{b}_{\text{enc}}) \quad (1)$$

The candidate code is $\mathbf{a}^{(c)} = \text{TopK}_c(\mathbf{z})$, where c is the candidate budget. In the 32,768-feature experiments, $c = 128$ throughout training. The corresponding mask $\mathbf{m}_c = 1[\mathbf{a}^{(c)} > 0]$ restricts the iterative coefficient updates to this candidate support. The encoder is initialized from the decoder transpose, $W_{\text{enc}} \leftarrow D^\top$, with zero bias [8,21]. After optional refinement, a separate post-FISTA projection retains the largest k coefficients, where k is the target budget supplied by the curriculum.

Given the initializer $\mathbf{a}_0$, the code can be refined by solving a nonnegative sparse-coding problem of the form

$$\min_{\mathbf{a} \geq 0} \frac{1}{2} \|\mathbf{h} - D\mathbf{a}\|_2^2 + \lambda \|\mathbf{a}\|_1 \quad (2)$$

using a fixed number of FISTA iterations [18]. Let $f(\mathbf{a}) = 0.5\|\mathbf{h} - D\mathbf{a}\|_2^2$ denote the smooth reconstruction term, and let $\mathbf{y}^t$ denote the FISTA momentum point at iteration t . The gradient of the smooth term at $\mathbf{y}^t$ is

$$\nabla f(\mathbf{y}^t) = D^\top(D\mathbf{y}^t - \mathbf{h}) \quad (3)$$

We then take a gradient step

$$\mathbf{u}^t = \mathbf{y}^t - \eta D^\top(D\mathbf{y}^t - \mathbf{h}) \quad (4)$$

where η is the step size. In our implementation, $L \approx \|D^\top D\|_2$ is estimated using three power iterations without explicitly forming $D^\top D$, and the step size is set to $\eta = 1/L$. The proximal shrinkage step becomes the nonnegative one-sided map

$$\mathbf{a}^{t+1} = \max(0, \mathbf{u}^t - \eta\lambda) \quad (5)$$

Thus, $\mathbf{u}^t$ is the pre-proximal gradient-update vector, η is the FISTA step size, and λ controls the strength of the sparsity-inducing shrinkage. When $\lambda > 0$, Equation (5) performs the usual one-sided shrinkage step. When $\lambda = 0$, it reduces to nonnegative projection, so the resulting update is more precisely an accelerated nonnegative projected-gradient refinement. We therefore use “FISTA-style refinement” to emphasize that the momentum and step-size structure are retained even in this zero-shrinkage special case.

FISTA refinement is restricted to the candidate mask $\mathbf{m}_c = 1[\mathbf{a}^{(c)} > 0]$ rather than directly to the final target support. After N iterations, let $\tilde{\mathbf{a}}$ denote the refined candidate coefficient vector. The final sparse code is $\mathbf{a} = \text{TopK}_k(\tilde{\mathbf{a}})$, with $k \in \{128, 64, 32\}$ in

epochs 1, 2, and 3 onward, respectively. When $N = 0$, the procedure reduces to learned candidate scoring followed by the exact target TopK projection. When $\lambda = 0$, the proximal map reduces to nonnegative projection, so the unrolled iterations act as reconstruction-driven coefficient refinement without additional ℓ_1 shrinkage. This candidate-to-target construction preserves exact target sparsity while allowing gradients to pass through a wider set of near-boundary candidate features during training.

2.3. Training Objective and Optimization

In the SAE experiments, the pre- and post-MLPs are disabled, so the model consists of a learned sparse encoder, a linear dictionary decoder, optional candidate-support FISTA refinement, and an exact target TopK projection. Given $\mathbf{h}$, the model produces the final target code $\mathbf{a}$ and reconstruction $\hat{\mathbf{h}} = D\mathbf{a}$. The primary training objective is a mean squared reconstruction error

$$\mathcal{L}_{\text{recon}} = \|\mathbf{h} - \hat{\mathbf{h}}\|_2^2 \quad (6)$$

averaged over token activations from the target residual-stream hook. The sparse code is nonnegative. In the 32,768-feature ablations, the shared target curriculum is $128 \rightarrow 64 \rightarrow 32$ and the candidate budget remains 128. The principal step-count experiments use $N \in \{0, 1, 3, 5, 8\}$ with $\lambda = 0$; an additional comparison uses $N = 3$, $\lambda = 0.01$. The near-miss ranking experiments use $N = 3$, $\lambda = 0$, $\alpha_{\text{rank}} = 0.03$ unless otherwise stated. All models are optimized with Adam [24] using a learning rate of 3×10^{-5} and bf16 mixed precision. Adam used the PyTorch default coefficients $(\beta_1, \beta_2) = (0.9, 0.999)$ with zero weight decay, and gradients were clipped to a maximum norm of 1.0. Checkpoints were saved at epoch boundaries, and reported models correspond to the predefined training endpoints described in the Results rather than to post hoc selection of the best validation checkpoint.

Randomness in the optimization procedure is limited to specific components. The TopK projection and the FISTA-style coefficient-update equations do not themselves involve random sampling; for a fixed input, model parameters, candidate support, and step-size value, they produce the same coefficient update. Stochasticity during training can instead arise from parameter initialization, epoch-wise ordering of the training activations, and the initialization used in the power-iteration estimate of the FISTA step size. In addition, Random-4 explicitly samples candidates from the positive-gain pool. For the cached 32,768-feature experiments, activation ordering was controlled by a fixed cache seed of 0 with an epoch-dependent deterministic reshuffling rule, while Random-4 used a separate random generator with seed 0. Thus, data exposure is reproducible across the controlled ablations, and the additional stochastic candidate selection in Random-4 is also reproducible.

Representative wall-clock measurements were obtained for the 32,768-feature step-count experiments on the RTX PRO 6000 Blackwell GPU with batch size 160. Processing one 24-million-token training epoch required approximately 36 min 48 s for $n_{\text{steps}} = 0$, 47 min 21 s for $n_{\text{steps}} = 5$, and 54 min 43 s for $n_{\text{steps}} = 8$. Thus, the five- and eight-step configurations increased the observed training time by approximately 29% and 49%, respectively, relative to the zero-step condition.

During training, the fixed number of FISTA refinement steps is unrolled as part of the computation graph. Reconstruction gradients therefore propagate through the final target projection and through the coefficient-refinement path on the candidate support. The dictionary and learned initializer are updated by Adam, after which every dictionary column is renormalized to unit ℓ_2 norm. For ranking experiments, the auxiliary ranking branch is added only to the training objective; it does not alter the exact TopK32 reconstruction returned by the main branch.

2.4. Near-Miss Ranking Loss

The ultra-sparse TopK32 setting exposes a narrow boundary between selected and excluded features. Let $\tilde{\mathbf{a}}$ be the refined candidate coefficient vector before the final TopK32 projection. For each token, S contains the indices at ranks 1–32, W contains the weakest $m = 4$ indices in S , and T contains the near-miss indices at ranks 33–48. Candidate ranks are determined from $\tilde{\mathbf{a}}$, whereas the pairwise loss is applied to the learned initializer scores $\mathbf{z}$ so that the auxiliary gradient directly trains support selection.

For each $j \in T$, define $\mathbf{c}_j = \tilde{\mathbf{a}}_j \mathbf{d}_j$ and the detached main residual $\mathbf{r} = \text{stopgrad}(\mathbf{h} - D\mathbf{a})$. A residual-optimal nonnegative scale and its normalized reconstruction gain are computed as

$$\gamma_j = \text{clip} \left(\frac{\langle \mathbf{r}, \mathbf{c}_j \rangle}{\|\mathbf{c}_j\|_2^2 + \epsilon}, 0, \gamma_{\max} \right) \quad (7)$$

$$g_j = \frac{2\gamma_j \langle \mathbf{r}, \mathbf{c}_j \rangle - \gamma_j^2 \|\mathbf{c}_j\|_2^2}{\|\mathbf{r}\|_2^2 + \epsilon} \quad (8)$$

Equation (8) measures the reduction in squared residual error obtained by adding the candidate contribution $\gamma_j \mathbf{c}_j$. Specifically, expanding the new residual gives

$$\|\mathbf{r} - \gamma_j \mathbf{c}_j\|_2^2 = \|\mathbf{r}\|_2^2 - 2\gamma_j \langle \mathbf{r}, \mathbf{c}_j \rangle + \gamma_j^2 \|\mathbf{c}_j\|_2^2. \quad (9)$$

Therefore, the reduction in residual error is $2\gamma_j \langle \mathbf{r}, \mathbf{c}_j \rangle - \gamma_j^2 \|\mathbf{c}_j\|_2^2$. The γ_j^2 term arises because scaling a vector by γ_j scales its squared norm by γ_j^2 . Division by $\|\mathbf{r}\|_2^2 + \epsilon$ normalizes the gain across tokens with different residual magnitudes. In our experiments, $\gamma_{\max}$ is set to 1.0, and negative gains are clamped to zero. In the proposed Gain-ranked-4 condition, P contains the four candidates in T with the largest positive g_j . In the matched Random-4 ablation, up to four candidates are sampled from the same positive-gain pool using a fixed random generator with seed 0. Thus, Gain-ranked-4 and Random-4 use the same pool of positive-gain candidates and differ only in how candidates are selected from that pool: by descending gain or at random. Candidates with zero or negative residual gain are excluded from both conditions.

For every selected near-miss candidate p and weak active feature w , we apply a pairwise hinge loss that encourages the initializer score z_p to exceed z_w by a margin of δ [25].

$$\mathcal{L}_{\text{rank}} = \text{mean}_{p \in P, w \in W} \text{ReLU}(\delta + z_w - z_p), \quad \delta = 0.05 \quad (10)$$

The total objective is $\mathcal{L} = \mathcal{L}_{\text{recon}} + \alpha_{\text{rank}} \mathcal{L}_{\text{rank}}$. The ranking order used to define the active, weak-active, and near-miss sets is treated as fixed in the auxiliary branch. Residuals, dictionary atoms, gain values, and the discrete sets P and W are also detached. Gradients from $\mathcal{L}_{\text{rank}}$ flow only through the selected initializer-score entries $\{z_p\}_{p \in P}$ and $\{z_w\}_{w \in W}$, thereby updating the learned initializer parameters but not the dictionary or FISTA refinement path. Consequently, the ranking loss trains the one-shot support selector while leaving the main exact TopK32 reconstruction and its active budget unchanged. The complete candidate-to-target inference and optional near-miss ranking procedure is summarized in Algorithm 1.

Algorithm 1. Candidate-to-target TopK inference with optional FISTA refinement and near-miss ranking

Input:hidden vector $\mathbf{h}$; dictionary D ; encoder W_{enc} , $\mathbf{b}_{\text{enc}}$ candidate budget c ; target budget k ; steps N ; step size η ; coefficient λ rank interval $[r_s, r_e]$; promoted count m ; margin δ ; rank mode $q \in \{\text{gain}, \text{random}\}$

All vectors are written as column vectors.

1: $\mathbf{z} \leftarrow \text{ReLU}(W_{\text{enc}}\mathbf{h} + \mathbf{b}_{\text{enc}})$ 2: $\mathbf{a}_c \leftarrow \text{TopK}(\mathbf{z}, c)$; $\mathbf{m}_c \leftarrow 1[\mathbf{a}_c > 0]$ 3: $\mathbf{a} \leftarrow \mathbf{a}_c$; $\mathbf{y} \leftarrow \mathbf{a}$; $s \leftarrow 1$ 4: **for** $t = 1, \dots, N$ **do**5: $\mathbf{grad} \leftarrow D^\top(D\mathbf{y} - \mathbf{h})$ 6: $\mathbf{u} \leftarrow \mathbf{y} - \eta \mathbf{grad}$ 7: $\mathbf{a}_{\text{next}} \leftarrow \max(0, \mathbf{u} - \eta\lambda) \odot \mathbf{m}_c$ 8: $s_{\text{next}} \leftarrow \frac{1 + \sqrt{1 + 4s^2}}{2}$ 9: $\mathbf{y} \leftarrow \left[\mathbf{a}_{\text{next}} + \frac{s-1}{s_{\text{next}}}(\mathbf{a}_{\text{next}} - \mathbf{a}) \right] \odot \mathbf{m}_c$ 10: $\mathbf{a} \leftarrow \mathbf{a}_{\text{next}}$; $s \leftarrow s_{\text{next}}$ 11: **end for**12: $\tilde{\mathbf{a}} \leftarrow \mathbf{a}$ 13: $\mathbf{a} \leftarrow \text{TopK}(\tilde{\mathbf{a}}, k)$; $\mathbf{recon} \leftarrow D\mathbf{a}$ 14: $\mathcal{L}_{\text{rank}} \leftarrow 0$ 15: **if** $\alpha_{\text{rank}} > 0$ **and** $k = 32$ **then**16: $W \leftarrow$ weakest m indices among ranks $1, \dots, k$ of $\tilde{\mathbf{a}}$ 17: $T \leftarrow$ indices at ranks $r_s, \dots, r_e$ of $\tilde{\mathbf{a}}$ 18: $r \leftarrow \text{stopgrad}(\mathbf{h} - \mathbf{recon})$; compute detached $\{g_j : j \in T\}$ 19: $P \leftarrow$ top- m positive-gain indices (gain mode) or random positive-gain indices (random mode)20: $\mathcal{L}_{\text{rank}} \leftarrow \text{mean}_{p \in P, w \in W} \text{ReLU}(\delta + z_w - z_p)$ 21: **end if**22: **return** $\mathbf{recon}, \mathbf{a}, \mathcal{L}_{\text{rank}}$

2.5. Baseline and Ablation Controls

The matched 12,288-feature TopK SAE baseline uses PyTorch's default Kaiming-uniform-style initialization separately for the encoder and decoder. The auxiliary DL-style SAE control initializes $D \sim \mathcal{N}(0, 1/d)$, sets $W_{\text{enc}} = D^\top$, and uses zero biases. For the 32,768-feature ultra-sparse study, the no-ranking controls vary only N and λ while retaining the candidate TopK128 and the $128 \rightarrow 64 \rightarrow 32$ curriculum. To isolate the effect of the candidate-selection strategy, the Gain-ranked-4 and Random-4 experiments use the same target TopK32 schedule, data, seed, optimization settings, rank interval 33–48, $m = 4$, margin 0.05, and $\alpha_{\text{rank}} = 0.03$. Random-4 differs from Gain-ranked-4 only in selecting candidates randomly from the positive-gain pool rather than ranking them by detached residual gain.

2.6. Evaluation Protocol

Evaluation is performed by patching the reconstructed hidden state back into the transformer continuation at blocks.11.hook_resid_post [1,2]. Each checkpoint is evaluated with the same candidate budget, FISTA step count, λ , and checkpoint-specific target TopK used by its training condition. The auxiliary ranking branch is disabled during evaluation; Gain-ranked-4 and Random-4 therefore use the same candidate-to-target reconstruction

path without random candidate sampling. Here, CE denotes next-token cross-entropy loss. We measure: (i) baseline CE loss, (ii) patched CE loss, (iii) $\Delta\text{CE} = \text{patched_loss} - \text{baseline_loss}$, (iv) logit-difference preservation, and (v) effective L0, density, dead-feature ratio, and feature-usage entropy.

The evaluator uses cached validation tokens and cached residual-stream tensors whenever available, which greatly reduces the cost of causal patching. Specifically, before evaluation, the validation split is tokenized once and the target residual-stream activations at blocks.11.hook_resid_post are precomputed and stored. During each evaluation run, the sparse model is applied directly to these cached activations to produce reconstructed residuals, which are then patched back into the transformer continuation for CE and logit-margin measurement.

The logit-difference preservation ratio measures how much of the model's original next-token preference is retained after patching. For each correct next-token ID y , we define a deterministic reference token with vocabulary ID $\tilde{y} = (y + 1) \bmod V$, where V is the vocabulary size. The logit margin is then computed as the logit of the correct token minus the logit of this reference token. These logits are pre-softmax values and therefore have no absolute reference level, because a common additive shift of all vocabulary logits leaves the softmax distribution unchanged. Subtracting a fixed deterministic reference logit removes such common shifts and expresses the correct-token score relative to a reproducible reference. The reported ratio is the mean patched margin divided by the mean baseline margin. Values close to 1 indicate that patching preserves this reference margin. This is a lightweight causal-preservation metric rather than a replacement for CE, and is best interpreted jointly with ΔCE .

To quantify direct reconstruction quality in activation space, we additionally report normalized mean squared error (NMSE) and fraction of variance explained (FVE) for selected ultra-sparse models. Let h_i denote the original residual-stream activation for token i , $\hat{h}_i$ its sparse reconstruction, and $\bar{h}$ the mean validation activation. We define

$$\text{NMSE} = \frac{\sum_{i=1}^N \|h_i - \hat{h}_i\|^2}{\sum_{i=1}^N \|h_i\|^2} \quad (11)$$

$$\text{FVE} = 1 - \frac{\sum_{i=1}^N \|h_i - \hat{h}_i\|^2}{\sum_{i=1}^N \|h_i - \bar{h}\|^2} \quad (12)$$

Lower NMSE and higher FVE indicate better direct reconstruction. These metrics are computed directly from the cached validation activations and sparse reconstructions and therefore do not require a transformer continuation pass. For the epoch-11 comparisons reported in Section 3.3, 198,656 token activations were included under the existing 200,000-token evaluation cap.

Effective L0 denotes the average number of active sparse coefficients per token. For each evaluation token, coefficients with magnitude greater than 10^{-8} are counted as active, and the count is then averaged over all evaluation tokens. Density is effective L0 divided by the dictionary size K , and therefore represents the average fraction of dictionary features active for one token.

For each feature j , we count the number of evaluation tokens on which its activation exceeds 10^{-8} . A feature is classified as dead when this count is zero, and the dead-feature ratio is the number of dead features divided by K . Thus, a high dead-feature ratio indicates that a substantial portion of the learned dictionary is never used on the validation activation cache. To assess the sensitivity of this statistic, we additionally evaluated selected epoch-11 TopK32 checkpoints using activation thresholds $\epsilon \in \{10^{-8}, 10^{-4}, 10^{-3}, 10^{-2}\}$ and nested validation prefixes containing 49,152, 98,304, and 198,656 token activations. The sensitivity

results are reported in Supplementary Table S9. We additionally examined the magnitude distribution of the retained TopK32 coefficients to determine why the dead-feature counts were invariant across these thresholds.

Feature-usage entropy measures how evenly activation events are distributed across the dictionary. Let n_j denote the number of tokens on which feature j is active, and define $p_j = n_j / \sum_{\ell=1}^K n_{\ell}$. The entropy is

$$H = -\sum_{j=1}^K p_j \log p_j \quad (13)$$

Higher entropy indicates that activation events are distributed more evenly across features, whereas lower entropy indicates that a relatively small subset of features accounts for most activations. The maximum value is $\log K$ when all features are used equally often. This metric reflects dictionary utilization rather than the semantic interpretability of individual features.

3. Results

3.1. 12,288-Feature Benchmark

In the 12,288-feature benchmark, the candidate and final TopK budgets are identical ($c = k$), so FISTA refines coefficient magnitudes within the same TopK support rather than using a wider candidate set. The TopK sweep shows that increasing the active budget predictably improves fidelity, but the best matched-budget operating point is not the largest budget. Instead, TopK256 with $n_steps = 5$ provides the most compelling in-regime trade-off between language-model preservation and feature utilization. The results of the TopK sweep and the matched SAE baseline are summarized in Table 1.

Table 1. TopK sweep and matched sparse autoencoder (SAE) baseline.

Model	TopK	n_Steps	Lambda	ΔCE	Logit_Ratio	Effective_L0	Dead_Ratio_%	Entropy
DL SAE-like	64	0	0.00	0.2803	0.9271	64.0000	7.2917	6.5050
DL SAE-like	64	5	0.01	0.2814	0.9286	64.0000	7.2591	6.5157
DL SAE-like	128	0	0.00	0.1758	0.9502	128.0000	4.5166	7.7727
DL SAE-like	128	5	0.01	0.1618	0.9562	128.0000	3.1250	7.7040
DL SAE-like	256	0	0.00	0.0834	0.9680	256.0000	8.8623	7.6892
DL SAE-like	256	5	0.01	0.0745	0.9702	256.0000	3.0924	8.1136
DL SAE-like	512	0	0.00	0.0143	0.9812	512.0000	35.3109	6.6611
DL SAE-like	512	5	0.01	0.0177	0.9783	511.9999	27.1973	6.6452
TopK SAE baseline	256	-	-	0.0859	0.9673	256.0000	50.7568	6.3688

At TopK256, the proposed method improves the matched SAE baseline from ΔCE 0.0859 to 0.0745, increases the logit ratio from 0.9673 to 0.9702, keeps the active budget fixed at 256, and reduces the dead-feature ratio from 50.76% to 3.09%. This is the paper's strongest headline result because unlike the TopK512 model, it does not change the active-feature budget.

3.1.1. TopK256 n-Step Ablation

Only TopK256 has the full sweep over $n_steps \in \{0,1,3,5,8\}$, making it the natural setting for isolating the effect of iterative refinement. The pattern is clear: small positive step counts help, and five steps are best overall. The detailed results of the TopK256 n-step ablation are summarized in Table 2.

The sweep suggests the following interpretation. Moving from $n_steps = 0$ to $n_steps = 3$ steadily reduces the dead-feature ratio and modestly improves CE. Moving to $n_steps = 5$ further improves both metrics and produces the highest feature-usage entropy.

At $n_steps = 8$, dead features decline slightly further, but fidelity degrades relative to the five-step model, indicating over-refinement in the present configuration.

Table 2. TopK256 ablation.

n_Steps	Lambda	ΔCE	Logit_Ratio	Effective_L0	Dead_Ratio%	Entropy
0	0	0.0834	0.9680	256.000	8.8623	7.6892
1	0.01	0.0829	0.9642	256.000	7.0719	7.7007
3	0.01	0.0788	0.9695	256.000	4.6387	7.8347
5	0.01	0.0745	0.9702	256.000	3.0924	8.1136
8	0.01	0.0861	0.9695	255.698	2.8483	7.7791

3.1.2. Decomposing the Gain: Zero-Step DL Configuration and FISTA Refinement

The benchmark strongly suggests that the dead-feature gain is not a single-factor effect. Table 3 provides a compact decomposition of the gains attributable to the zero-step dictionary-learning configuration and the subsequent FISTA refinement.

Table 3. Decomposition of the TopK256 gain.

System	ΔCE	Logit_Ratio	Effective_L0	Dead_Ratio_ %	Entropy
TopK SAE baseline (k = 256)	0.0859	0.9673	256	50.7568	6.3688
DL SAE-like, learned TopK init, n_steps = 0	0.0834	0.9680	256	8.8623	7.6892
DL SAE-like + FISTA refinement, n_steps = 5, $\lambda = 0.01$	0.0745	0.9702	256	3.0924	8.1136

The most important quantitative observation is that moving from the matched SAE baseline to DL SAE-like $n_steps = 0$ reduces dead features by 41.89 percentage points, whereas the subsequent FISTA refinement from $n_steps = 0$ to $n_steps = 5$ reduces dead features by an additional 5.77 percentage points. By contrast, the CE improvement between SAE and DL $n_steps = 0$ is small but positive, while the CE improvement from $n_steps = 0$ to $n_steps = 5$ is much larger in relative terms. Thus the best-supported interpretation is that DL-style support selection and initialization dominate the dead-feature improvement, while FISTA provides the larger fidelity gain and an additional utilization gain.

3.1.3. Auxiliary Control: SAE Initialization Sweep

The project also contains a separate archived SAE control study in which a TopK SAE was retrained under different initialization schemes. These values are useful for causal interpretation of the dead-feature phenomenon, but they should be treated as an auxiliary control, not as part of the fully DL-vs-converted-SAE main benchmark. In this auxiliary sweep, the “Existing SAE TopK256” row corresponds to the standard TopK SAE trained with the default PyTorch linear-layer initialization, whereas the “DL-style init” row replaces only the initialization scheme with a Gaussian decoder dictionary, transposed encoder initialization, and zero biases.

In PyTorch’s default linear initialization, the encoder and decoder use different fan-in values because their input dimensions differ. The encoder weight has shape $K \times d$ and therefore fan-in $d = 768$, whereas the decoder weight has shape $d \times K$ and therefore fan-in $K = 12,288$. Because the default uniform bound is proportional to $1/\sqrt{\text{fan-in}}$, the decoder is initialized with a bound four times smaller than the encoder. Thus, the two transpose-shaped matrices do not begin at comparable scales.

This scale asymmetry is important for interpreting Table 4. “initT” first initializes the decoder with the default Kaiming-uniform scheme and then sets the encoder weight to the

decoder transpose. It therefore aligns the initial encoder and decoder directions but retains the decoder’s smaller K-based scale; it does not reproduce the residual-dimension scale used by the dictionary-learning initialization. “normD” renormalizes each decoder column to unit ℓ_2 norm before and after optimizer updates. Accordingly, “normD only” controls decoder-atom norms without aligning the encoder, whereas “initT + normD” combines directional alignment with decoder-column normalization. The “DL-style init” condition instead initializes the decoder from a Gaussian distribution with standard deviation $1/\sqrt{d}$, sets the encoder to the decoder transpose, and uses zero biases. The results of the auxiliary SAE initialization sweep are summarized in Table 4.

Table 4. Auxiliary SAE initialization control at TopK256.

Setting	Δ CE	Logit_Ratio	Effective_L0	Dead_Ratio_%	Entropy
Existing SAE TopK256	0.0859	0.9673	256	50.7600	6.3690
initT only	0.1069	0.9581	256	40.8700	6.2510
normD only	0.0853	0.9625	256	63.8600	6.5290
initT + normD	0.0897	0.9614	256	70.4900	6.5380
DL-style init	0.0941	0.9624	256	11.9100	6.7550

The re-evaluated initialization control confirms that the dead-feature conclusion is stable under the corrected evaluator. The default TopK SAE retains a dead-feature ratio of 50.7600%, whereas the DL-style initialization reduces it to 11.9100%. The Δ CE values shift slightly under the corrected evaluation path, but the utilization pattern is unchanged: initT alone only partially reduces dead features, normD does not help in this run, and the full DL-style initialization package produces the largest utilization gain among the SAE initialization controls. Importantly, DL-style initialization mainly explains the utilization improvement rather than the final fidelity improvement; the best corrected DL model with FISTA refinement is the condition that improves both Δ CE and dead-feature ratio.

3.2. 32,768-Feature Matched-Support TopK Sweep

To determine whether the matched-support design remains trainable after expanding the dictionary to $K = 32,768$, we first set the candidate TopK equal to the final TopK and trained each condition for three epochs, corresponding to 72 million tokens. Table 5 shows a sharp degradation at TopK32. With $n_{\text{steps}} = 0$, Δ CE remained 0.7465 and the dead-feature ratio reached 63.12%. Adding five FISTA steps with $\lambda = 0.01$ did not recover this setting; it yielded Δ CE = 0.7575, a logit ratio of 0.8398, and 72.27% dead features. In contrast, TopK64 and TopK128 were substantially more stable, and the refined variants produced small gains at these larger active budgets.

Across the sweep, larger active budgets predictably improved reconstruction fidelity, but dictionary utilization was non-monotonic. TopK128 and TopK256 benefited from refinement in both Δ CE and dead-feature ratio, whereas TopK512 and TopK1024 achieved very low Δ CE while leaving approximately 44–49% of the dictionary dead. The direct TopK32 regime therefore remained poorly trained after 72 million tokens when the candidate and final supports were identical. This result motivated the candidate-to-target design used in the following experiments: the model retained a wider TopK128 candidate support for coefficient refinement and gradient flow, while the final reconstruction budget was reduced through the shared TopK128 $\rightarrow$ TopK64 $\rightarrow$ TopK32 curriculum.

Table 5. 32,768-feature TopK sweep after 72 million training tokens (candidate TopK equals final TopK).

TopK	n_Steps	λ	Δ CE	Logit Ratio	Effective L0	Dead Ratio (%)	Entropy
32	0	0.00	0.7465	0.8465	32.00	63.12	4.6119
32	5	0.01	0.7575	0.8398	32.00	72.27	4.4812
64	0	0.00	0.3013	0.9168	64.00	13.47	6.9096
64	5	0.01	0.2966	0.9210	63.97	13.32	6.9616
128	0	0.00	0.1631	0.9437	127.50	11.24	7.8158
128	5	0.01	0.1430	0.9497	127.40	9.51	8.3241
256	0	0.00	0.0694	0.9652	251.73	22.08	7.9389
256	5	0.01	0.0630	0.9666	251.05	15.16	8.1014
512	0	0.00	0.0150	0.9793	506.59	47.45	6.9606
512	5	0.01	0.0270	0.9698	511.94	44.61	6.9664
1024	0	0.00	−0.0005	0.9995	1022.73	48.94	6.9768
1024	5	0.01	−0.0037	0.9939	1023.51	44.28	7.0016

3.3. Ultra-Sparse TopK32 Training Trajectories

Figure 1 tracks the 32,768-feature runs under the shared TopK128 → TopK64 → TopK32 curriculum, with the target budget changing at the start of epochs 2 and 3. The transition to TopK32 produced a synchronized disturbance across all conditions: the dead-feature ratio and Δ CE rose sharply, while logit preservation and feature-usage entropy fell. The post-transition recovery indicates that optimization can continue to improve the exact TopK32 model when coefficients are refined over a wider TopK128 candidate set before the final TopK32 projection.

With $\lambda = 0$, three FISTA steps primarily accelerated early recovery rather than changing the apparent endpoint. At epoch 3, $n_steps = 3$ improved Δ CE from 0.7323 to 0.6966 and reduced the dead-feature ratio from 65.56% to 60.92% relative to $n_steps = 0$. By epoch 11, however, the two runs were nearly identical: Δ CE was 0.2915 versus 0.2895, and the dead-feature ratio was 24.41% versus 24.51%. This pattern is consistent with short unrolled refinement reducing the amortization gap during training while leaving the exact TopK32 output budget unchanged [20].

Five and eight refinement steps produced a stronger effect. At epoch 11, $n_steps = 5$ achieved Δ CE = 0.2442, a logit ratio of 0.9414, a dead-feature ratio of 17.84%, and entropy of 6.4934. The $n_steps = 8$ run produced slightly better fidelity (Δ CE = 0.2430; logit ratio = 0.9453), but its dead-feature ratio was 24.34% and its entropy was 6.3930. Thus, five steps gave the more balanced fidelity–utilization endpoint. By contrast, $n_steps = 3$ with $\lambda = 0.01$ remained weaker than the corresponding $\lambda = 0$ run through most of the TopK32 phase and ended at Δ CE = 0.3034 with 26.02% dead features. In this exact-budget setting, additional ℓ_1 shrinkage did not improve the trajectory, whereas $\lambda = 0$ allowed FISTA to act as reconstruction-driven coefficient refinement within the candidate support.

Figure 2 isolates the effect of the near-miss ranking objective. The $n_steps = 3$, $\lambda = 0$ reference, Gain-ranked-4, and Random-4 runs share the same TopK128 and TopK64 training history and branch from the same completed TopK64 checkpoint. The two ranking variants differ only in how four positive-gain candidates from ranks 33–48 are selected for auxiliary score updates.

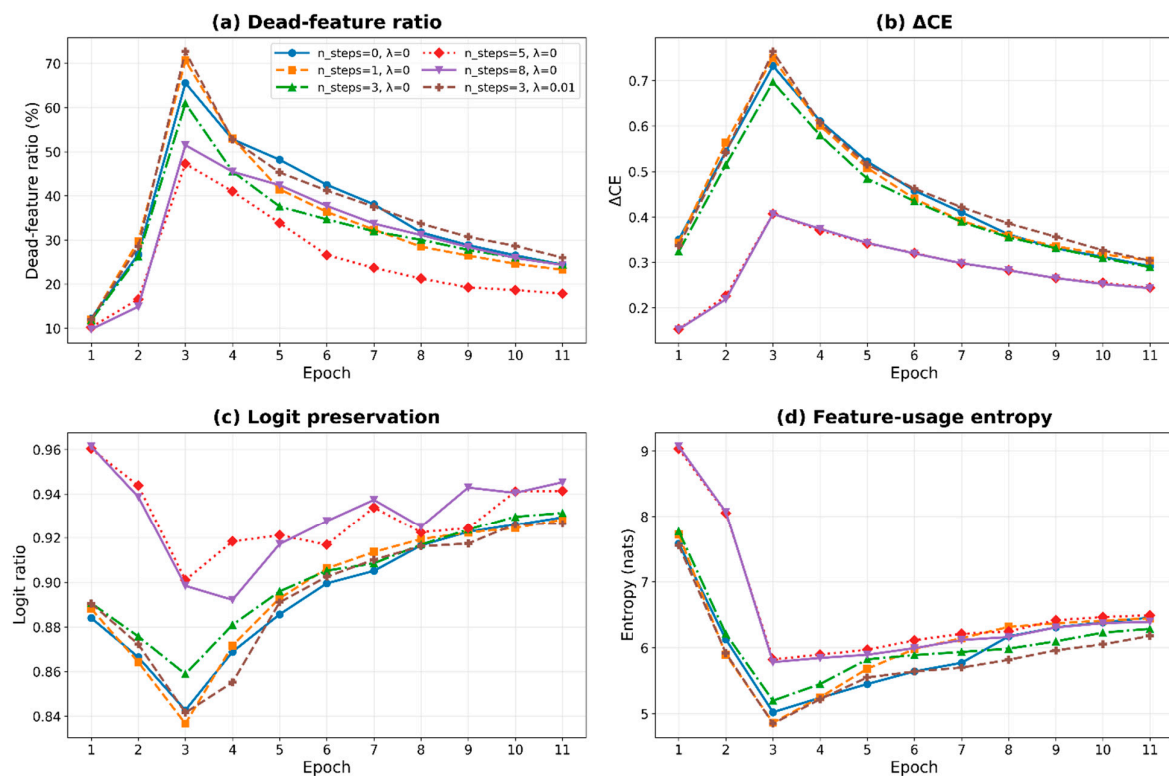


Figure 1. Ultra-sparse TopK32 training trajectories under the shared TopK128 $\rightarrow$ TopK64 $\rightarrow$ TopK32 curriculum. Panels show (a) dead-feature ratio, (b) ΔCE , (c) logit-difference preservation ratio, and (d) feature-usage entropy for $n_steps \in \{0, 1, 3, 5, 8\}$ with $\lambda = 0$ and for $n_steps = 3$ with $\lambda = 0.01$. All models use $K = 32,768$ and a candidate TopK128 support; the target budget is TopK128 at epoch 1, TopK64 at epoch 2, and TopK32 from epoch 3 onward. Each epoch corresponds to 24 million training-token presentations, and the plotted trajectories cover 11 epochs (264 million token presentations). Each condition represents one training run ($n = 1$).

Both ranking variants accelerated feature-utilization recovery immediately after the TopK32 transition. At epoch 3, the dead-feature ratio was 50.12% for Gain-ranked-4 and 55.26% for Random-4, compared with 60.92% for the unranked reference. Gain-based selection was consistently stronger than random promotion. At epoch 11, Gain-ranked-4 reached $\Delta CE = 0.3028$, a logit ratio of 0.9324, 18.46% dead features, and entropy of 6.3109; Random-4 reached 0.3216, 0.9257, 20.11%, and 6.2898, respectively. Gain-ranked-4 did not surpass the reference in ΔCE at epoch 11 (0.3028 versus 0.2895), but it substantially reduced the dead-feature ratio while preserving a similar logit ratio. Because the auxiliary objective changes only four candidates in the narrow rank-33–48 band, the result indicates that ultra-sparse convergence is sensitive to a small set of near-boundary features and that ranking them by detached reconstruction gain is more effective than random selection. Positive-gain candidates in ranks 33–48 were available for essentially all evaluated token positions during training (>99.99% in both ranking conditions). The required ranking margin was not yet satisfied in 97.85% of valid comparisons for Gain-ranked-4 and 98.60% for Random-4, indicating that the auxiliary ranking loss provided a nonzero training signal for most comparisons. The mean promoted residual gain was higher for Gain-ranked-4 (0.0328) than for Random-4 (0.0161).

To complement the causal patching metrics with direct activation-space reconstruction measures, we evaluated selected epoch-11 TopK32 checkpoints using the NMSE and FVE definitions in Section 2.6. Table 6 reports these measures alongside ΔCE and logit preservation. The five-step model achieved the lowest NMSE (0.0293) and the highest FVE (0.8421), while the eight-step model was close behind (0.0295 and 0.8411, respectively). The

ordering is not identical to the downstream metrics, indicating that direct activation-space reconstruction and causal preservation capture related but distinct aspects of fidelity.

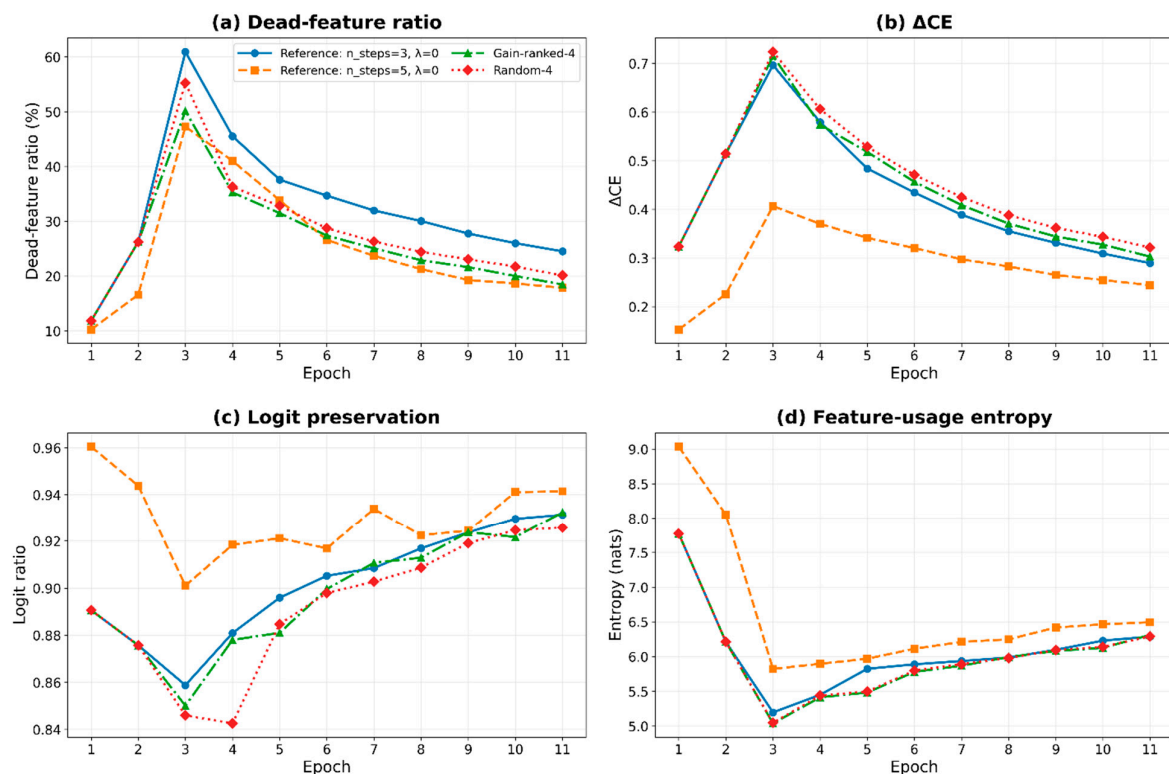


Figure 2. Effect of near-miss ranking during ultra-sparse TopK32 training. The $n_steps = 3$, $\lambda = 0$ reference is compared with Gain-ranked-4 and Random-4, and the $n_steps = 5$, $\lambda = 0$ trajectory is shown as an additional no-ranking reference. All models use $K = 32,768$ and a candidate TopK128 support under the shared TopK128 $\rightarrow$ TopK64 $\rightarrow$ TopK32 target curriculum. Gain-ranked-4 and Random-4 use $n_steps = 3$, $\lambda = 0$, and ranking-loss weight 0.03; both promote four positive-gain candidates from ranks 33–48, with Gain-ranked-4 selecting the largest detached residual gains and Random-4 sampling from the same positive-gain pool. Panels show (a) dead-feature ratio, (b) ΔCE , (c) logit-difference preservation ratio, and (d) feature-usage entropy. Each epoch corresponds to 24 million training-token presentations, and the plotted trajectories cover 11 epochs (264 million token presentations). Each condition represents one training run ($n = 1$); therefore, uncertainty bands are not shown.

Table 6. Direct reconstruction metrics for selected epoch-11 ultra-sparse TopK32 models. $\uparrow$ indicates that higher values are better, whereas $\downarrow$ indicates that lower values are better.

Condition	ΔCE	Logit Ratio	NMSE $\downarrow$	FVE $\uparrow$
$n_steps = 0$, $\lambda = 0$	0.2915	0.9294	0.0310	0.8331
$n_steps = 3$, $\lambda = 0$	0.2895	0.9314	0.0315	0.8305
$n_steps = 5$, $\lambda = 0$	0.2442	0.9414	0.0293	0.8421
$n_steps = 8$, $\lambda = 0$	0.2430	0.9453	0.0295	0.8411
Gain-ranked-4 ($n_steps = 3$)	0.3028	0.9324	0.0317	0.8293

All rows use $K = 32,768$, candidate TopK128, final TopK32, and $\lambda = 0$ at epoch 11 (264M token presentations); Gain-ranked-4 uses $n_steps = 3$. NMSE and FVE were computed on the same 198,656 cached validation-token activations.

The dead-feature ratios were identical across activation thresholds from 10^{-8} to 10^{-2} . This invariance reflects the exact TopK32 projection rather than insensitivity of the metric to

arbitrary thresholds: across the five evaluated models, the smallest retained nonzero coefficient ranged from 2.19 to 3.78, well above the tested threshold range. Consistently, extending the threshold analysis showed no change in effective L0 or dead-feature ratio through a threshold of 1.0. Increasing the evaluation-set size reduced the absolute dead-feature ratios, as expected, while the main utilization advantage of the five-step and Gain-ranked-4 models over the zero-step model was preserved (Supplementary Tables S9 and S10).

Complete epoch-wise values are reported in Supplementary Tables S1–S8. The supplementary trajectories also show that no single condition dominates every metric: $n_steps = 8$ slightly favors endpoint fidelity, $n_steps = 5$ provides the best balanced utilization, and Gain-ranked-4 most clearly accelerates near-boundary feature recruitment at a modest ΔCE cost.

3.4. Final Comparison of Selected Ultra-Sparse Models

Table 7 compares five representative endpoints: the matched-duration $n_steps = 3$ and $n_steps = 5$, $\lambda = 0$ runs after 11 epochs; the Gain-ranked-4 run after 11 epochs; the extended $n_steps = 3$, $\lambda = 0$ reference after 16 epochs; and the OpenAI 32k SAE as an auxiliary external reference. The in-house rows share the exact TopK32 output budget, while the epoch-11 rows permit a direct training-duration-matched comparison.

Table 7. Final comparison of selected ultra-sparse TopK32 models.

Condition	Training Endpoint	ΔCE	Logit Ratio	Effective L0	Dead Ratio (%)	Entropy
FISTA-refined, $n_steps = 5$, $\lambda = 0$	Epoch 11 (264M)	0.2442	0.9414	32.00	17.84	6.4934
FISTA-refined, $n_steps = 3$, $\lambda = 0$	Epoch 11 (264M)	0.2895	0.9314	32.00	24.51	6.2867
Gain-ranked-4, $n_steps = 3$, $\lambda = 0$	Epoch 11 (264M)	0.3028	0.9324	32.00	18.46	6.3109
FISTA-refined, $n_steps = 3$, $\lambda = 0$	Epoch 16 (384M)	0.2359	0.9454	32.00	18.56	6.6049
OpenAI SAE 32k [8,21]	Billions+ tokens	0.3300	0.9500	32.00	9.95	8.4140

Among the in-house models, the epoch-11 comparison shows that increasing refinement from three to five steps substantially accelerated convergence at the same 264-million-token endpoint: ΔCE improved from 0.2895 to 0.2442, the logit ratio increased from 0.9314 to 0.9414, the dead-feature ratio fell from 24.51% to 17.84%, and entropy rose from 6.2867 to 6.4934. The extended $n_steps = 3$ run achieved the best endpoint fidelity after 384 million tokens ($\Delta CE = 0.2359$; logit ratio = 0.9454), but the $n_steps = 5$ model nearly matched it after only 264 million tokens and retained a slightly lower dead-feature ratio, 17.84% versus 18.56%. This matched-duration and extended-training comparison suggests that five-step refinement primarily accelerates convergence toward a useful fidelity–utilization region rather than producing a fundamentally different endpoint.

Gain-ranked-4 reached a dead-feature ratio of 18.46% after 264 million tokens, but its ΔCE of 0.3028 was weaker than both matched-duration unranked endpoints: 0.2895 for $n_steps = 3$ and 0.2442 for $n_steps = 5$. The ranking loss therefore appears to act primarily on support utilization and convergence speed rather than directly optimizing final reconstruction fidelity. The OpenAI 32k SAE achieved the lowest dead-feature ratio (9.95%) and the highest entropy (8.4140), but it was trained on billions or more tokens under a different pipeline [8,21]. It is therefore included as a scale reference rather than a matched benchmark. Its reported metrics were recomputed using the same evaluation pipeline and cached WikiText-103 validation activations used for the in-house models; therefore, the evaluation protocol is matched, whereas the training conditions are not. Its ΔCE and logit

ratio also show that the two metrics do not always change in the same direction across different models.

4. Discussion

The experiments support a specific claim: learned TopK support selection and lightweight coefficient refinement can improve the fidelity–utilization trade-off of SAE-style decompositions, but the two components contribute in different ways. In the matched 12,288-feature TopK256 benchmark, the proposed five-step model improves Δ CE from 0.0859 to 0.0745 and reduces the dead-feature ratio from 50.76% to 3.09% without changing effective L0. In the 32,768-feature experiments, the main role of few-step FISTA is most visible as faster recovery after the transition to exact TopK32 rather than as evidence for a fundamentally different eventual fidelity–utilization endpoint.

We frame this contribution as an optimization study of fixed-budget sparse decomposition rather than as a claim about semantic feature quality. The metrics reported here—patched cross-entropy, logit preservation, normalized MSE, fraction of variance explained, dead-feature ratio, and usage entropy—characterize reconstruction fidelity and dictionary utilization, and the algorithmic mechanisms we study (learned-TopK-plus-FISTA refinement, the candidate-to-target curriculum, and the residual-gain-ranked near-miss loss) are defined over a dictionary and a sparse-coding objective independently of the specific residual-stream layer that supplies the activations. The GPT-2 Small layer-11 residual stream therefore serves as a controlled testbed on which each mechanism is isolated and measured, not as a claim that the specific numerical values transfer unchanged to other layers or models.

Table 3 separates the improvements obtained before FISTA refinement from the additional improvements obtained after coefficient refinement. Moving from the matched TopK256 SAE baseline to the zero-step dictionary-learning model reduces the dead-feature ratio from 50.76% to 8.86%, a decrease of 41.89 percentage points. Adding five FISTA steps with $\lambda = 0.01$ further reduces the dead-feature ratio to 3.09%, an additional decrease of 5.77 percentage points, while providing the larger improvement in reconstruction fidelity. Thus, most of the feature-utilization gain in this TopK256 experiment arises before iterative refinement, through DL-style initialization and the joint optimization of the learned TopK initializer and normalized dictionary decoder. FISTA then refines the coefficient magnitudes within the selected support and provides an additional utilization and fidelity benefit. This interpretation is consistent with the support-versus-magnitude distinction emphasized by gated SAEs [9].

We emphasize that these SAE variants—gated, JumpReLU, BatchTopK, and AuxK-enhanced encoders—modify the encoder or activation rule that produces the one-shot support. The refinement and candidate-curriculum mechanisms studied here are orthogonal to that choice: they act on the coefficients after support proposal and could in principle be combined with any of these encoders.

The initialization control further shows that high dead-feature ratios are not an unavoidable consequence of TopK sparsity. Under PyTorch’s default linear initialization, the encoder has fan-in $d = 768$ while the decoder has fan-in $K = 12,288$, so their transpose-shaped weights are initialized at different scales. Replacing this arrangement with a decoder initialized with standard deviation $1/\sqrt{d}$, where $d = 768$, a transposed encoder, zero biases, and dictionary-column normalization reduces dead features from 50.76% to 11.91% even within an SAE control. However, initialization alone does not reproduce the best Δ CE or the 3.09% dead-feature ratio of the refined model, indicating that training dynamics and coefficient correction remain consequential.

The direct 32,768-feature TopK32 experiments clarify why ultra-sparse learning is difficult. When the candidate and final supports are identical, the model repeatedly reconstructs each token from only 32 of 32,768 atoms, and features that fall just below the cutoff have little opportunity to displace the current support. After three epochs of direct TopK32 training, corresponding to 72 million token presentations, both the zero-step and five-step models still showed high ΔCE and dead-feature ratios above 60%. Retaining a wider TopK128 candidate set and gradually reducing the final budget to 32 changes this optimization geometry: near-boundary coefficients can be refined before the final projection, and the support selector is trained while exposed to a broader set of plausible atoms. The resulting recovery after epoch 3 shows that exact TopK32 is trainable, but only after the support bottleneck is relaxed during candidate generation.

The $\lambda = 0$ trajectories are especially informative. With no additional shrinkage, FISTA performs reconstruction-driven nonnegative coefficient updates within the candidate support. Three steps improve the early TopK32 trajectory but converge toward the zero-step run by epoch 11, whereas five steps reach a substantially better fidelity–utilization region at the same 264-million-token endpoint. The extended three-step run later approaches the five-step result, supporting an acceleration interpretation. By contrast, $n_steps = 3$ with $\lambda = 0.01$ is consistently weaker in the ultra-sparse phase. Because the final TopK projection already fixes $L_0 = 32$, extra ℓ_1 shrinkage appears unnecessary in this setting and may suppress useful candidate coefficients before the final selection.

The ranking experiments identify the rank-33–48 band as an important optimization boundary. Promoting only four positive-gain near-miss features reduces the epoch-11 dead-feature ratio from 24.51% in the unranked three-step reference to 18.46% with Gain-ranked-4. Because Random-4 uses the same auxiliary loss but selects candidates at random, the lower dead-feature ratio of Gain-ranked-4 suggests that residual-gain-based candidate selection is more effective than random selection. Nevertheless, Gain-ranked-4 has higher ΔCE than the unranked references. The ranking loss should therefore be interpreted as a support-utilization and convergence mechanism, not as a direct fidelity improvement. To the best of our knowledge, this form of residual-gain-ranked near-boundary training has not previously been applied to SAEs.

This ranking objective differs materially from the AuxK mechanism used in large-scale SAE training [8,21]. AuxK adds an auxiliary reconstruction using selected dead or underused latents, thereby giving those latents a direct role in reconstructing the residual and providing auxiliary gradients to both their encoder parameters and decoder atoms. Our loss instead compares initializer scores for weak active and near-miss candidates, uses detached residual gain only to select the ranking targets, and sends auxiliary gradients only through the selected initializer-score entries. The auxiliary branch therefore does not update the dictionary or backpropagate through the FISTA refinement path, and it does not directly modify the exact TopK32 reconstruction produced by the main branch. Accordingly, AuxK is primarily a dead-latent reconstruction mechanism, whereas the proposed loss is a boundary-ordering mechanism for the one-shot support selector.

The findings are consistent with an amortization-gap perspective, but they do not directly measure the optimal sparse code or prove that the gap is closed. In the present model, a few unrolled FISTA steps insert reconstruction-aware computation between the encoder scores and the final sparse code. The improved trajectories show that this correction can compensate for part of the one-shot encoder error, especially when $\lambda = 0$ and the candidate support is wider than the final budget.

The OpenAI 32k SAE is useful only as a scale reference. It reports a lower dead-feature ratio and higher usage entropy than the in-house TopK32 models, but it was trained on billions or more tokens under a different architecture and optimization pipeline [8,21].

The comparison therefore does not isolate a method effect. More broadly, ΔCE , logit preservation, dead-feature ratio, and entropy measure different aspects of decomposition quality and need not move together. In particular, broader feature usage does not by itself establish semantic coherence, causal usefulness, or canonical feature recovery [13–17].

Limitations and Future Work

Several limitations define the scope of the present study and motivate future work. First, the experiments are restricted to GPT-2 Small layer-11 residual-stream activations and therefore do not establish generality across models, layers, or activation sites. Future evaluation should include additional transformer layers, larger models, and alternative activation sites such as MLP and attention outputs. The ultra-sparse experiments also repeatedly optimize over a fixed cached 24-million-token activation corpus; although this provides a controlled multi-epoch setting for comparing optimization mechanisms, direct comparison with fresh-token or online activation training would test whether the observed gains transfer to continually refreshed data. In addition, the present metrics characterize reconstruction fidelity and dictionary utilization rather than semantic feature quality. Semantic purity, human or LLM-based interpretability, sparse probing, and feature-level causal or circuit analyses therefore remain important next-stage evaluations [26]. Direct comparisons with modern SAE variants, including gated, JumpReLU, BatchTopK, and AuxK-enhanced approaches, under a common evaluation pipeline would further clarify the relative contribution of the proposed refinement strategy.

Several experimental factors also remain to be isolated more systematically. The principal comparisons are based on single training runs. The fixed data-ordering used in the cached experiments, and the separate fixed candidate-sampling seed used for Random-4, make those sources of randomness reproducible; however, multi-seed replication is still needed to quantify run-to-run variability arising from parameter initialization and optimization. The direct-TopK32 experiment uses a shorter training budget than the curriculum-based models and should be repeated under duration- and compute-matched conditions. Likewise, the wider TopK128 candidate support and the TopK128→TopK64→TopK32 curriculum are used jointly, and their individual contributions should be separated in a controlled ablation. The near-miss ranking experiments use a single configuration of ranks 33–48, four promoted candidates, and a ranking-loss weight of 0.03; systematic sensitivity analyses over these hyperparameters are also needed. A more direct analysis of the amortization gap should compare the learned code with a well-converged sparse-coding solution under the same dictionary and active-budget constraint [20]. Finally, although representative wall-clock measurements are reported for the refinement-depth experiments, systematic condition-by-condition peak-memory profiling and hardware-normalized computational analysis remain for future work.

Within these limits, the results support a conservative conclusion: a wider candidate support, a learned selector, and a few reconstruction-aware refinement steps form a practical hybrid for training fixed-budget sparse decompositions.

5. Conclusions

This work evaluates a hybrid SAE-like dictionary-learning system in which a learned TopK encoder selects candidate features and a small number of FISTA iterations refines their coefficients before exact TopK reconstruction. In the 12,288-feature TopK256 benchmark, the five-step model outperforms the TopK SAE baseline under the same dictionary size and active-feature budget. The ablations show that initialization and learned support selection account for most of the reduction in dead features, while iterative refinement provides the larger incremental fidelity gain.

The 32,768-feature experiments show that exact TopK32 learning is limited by support selection as well as coefficient estimation. Direct TopK32 training remains poorly optimized, whereas a TopK128 candidate support, a TopK128→TopK64→TopK32 curriculum, and a small number of FISTA refinement steps with $\lambda = 0$ substantially accelerate convergence. The near-miss experiments further show that a small group of features at ranks 33–48 can strongly influence feature recruitment. Gain-ranked promotion reduces dead features more effectively than random promotion, although the present loss trades some reconstruction fidelity for faster and broader support utilization.

The main design implication is that support selection and coefficient refinement should be treated as separate but interacting components of sparse representation learning. A one-shot encoder provides efficient candidate selection, while short optimization-based correction can reduce encoder error without replacing SAE-style inference with a full solver.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/a19080697/s1>, Table S1: Training trajectory for $n_steps = 0$ and $\lambda = 0$; Table S2: Training trajectory for $n_steps = 1$ and $\lambda = 0$; Table S3: Training trajectory for $n_steps = 3$ and $\lambda = 0$ (epochs 1–16); Table S4: Training trajectory for $n_steps = 5$ and $\lambda = 0$; Table S5: Training trajectory for $n_steps = 8$ and $\lambda = 0$; Table S6: Training trajectory for $n_steps = 3$ and $\lambda = 0.01$; Table S7: Training trajectory for Gain-ranked-4 (rank weight = 0.03, $n_steps = 3$, $\lambda = 0$); Table S8: Training trajectory for Random-4 (rank weight = 0.03, $n_steps = 3$, $\lambda = 0$); Table S9: Sensitivity of dead-feature ratio (%) to activation threshold and evaluation-set size for selected epoch-11 ultra-sparse TopK32 models; Table S10: Magnitude distribution of retained nonzero coefficients in the epoch-11 TopK32 sensitivity models.

Author Contributions: Conceptualization, J.P.B.; software, J.P.B.; methodology, K.-I.J. and J.P.B.; formal analysis, J.P.B.; investigation, K.-I.J. and J.P.B.; data curation, J.P.B.; writing—original draft preparation, J.P.B.; writing—review and editing, K.-I.J. and J.P.B.; visualization, J.P.B.; supervision, K.-I.J. and J.P.B.; project administration, K.-I.J. and J.P.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets used in this study are publicly available. The source code is not publicly available; however, additional implementation details may be obtained from the corresponding author upon reasonable request.

Acknowledgments: During the preparation of this manuscript, the authors used ChatGPT (OpenAI, GPT-5.5) for software development assistance, and Claude (Anthropic, Claude Sonnet 4.6) for writing assistance and manuscript preparation. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nanda, N.; Bloom, J. TransformerLens. 2022. Available online: <https://github.com/TransformerLensOrg/TransformerLens> (accessed on 6 May 2026).
2. Heimersheim, S.; Nanda, N. How to Use and Interpret Activation Patching. *arXiv* **2024**, arXiv:2404.15255.
3. Olshausen, B.A.; Field, D.J. Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images. *Nature* **1996**, *381*, 607–609. [[CrossRef](#)] [[PubMed](#)]
4. Yun, Z.; Chen, Y.; Olshausen, B.; LeCun, Y. Transformer Visualization via Dictionary Learning: Contextualized Embedding as a Linear Superposition of Transformer Factors. In Proceedings of the Deep Learning Inside Out (DeeLIO) Workshop, Online, 10 June 2021; pp. 1–10.
5. Bricken, T.; Templeton, A.; Batson, J.; Chen, B.; Jermyn, A.; Conerly, T.; Turner, N.L.; Anil, C.; Denison, C.; Askell, A.; et al. Towards Monosemanticity: Decomposing Language Models with Dictionary Learning. Transformer Circuits Thread, 2023. Available online: <https://transformer-circuits.pub/2023/monosemantic-features> (accessed on 6 May 2026).

6. Cunningham, H.; Ewart, A.; Riggs, L.; Huben, R.; Sharkey, L. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
7. Makhzani, A.; Frey, B.J. k-Sparse Autoencoders. In Proceedings of the 2nd International Conference on Learning Representations (ICLR), Banff, AB, Canada, 14–16 April 2014.
8. Gao, L.; Dupré la Tour, T.; Tillman, H.; Goh, G.; Troll, R.; Radford, A.; Sutskever, I.; Leike, J.; Wu, J. Scaling and Evaluating Sparse Autoencoders. In Proceedings of the 13th International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
9. Rajamanoharan, S.; Conmy, A.; Smith, L.; Lieberum, T.; Varma, V.; Kramár, J.; Shah, R.; Nanda, N. Improving Dictionary Learning with Gated Sparse Autoencoders. *arXiv* **2024**, arXiv:2404.16014.
10. Rajamanoharan, S.; Lieberum, T.; Sonnerat, N.; Conmy, A.; Varma, V.; Kramár, J.; Nanda, N. Jumping Ahead: Improving Reconstruction Fidelity with JumpReLU Sparse Autoencoders. *arXiv* **2024**, arXiv:2407.14435.
11. Bussmann, B.; Leask, P.; Nanda, N. BatchTopK Sparse Autoencoders. *arXiv* **2024**, arXiv:2412.06410.
12. Braun, D.; Taylor, J.; Goldowsky-Dill, N.; Sharkey, L. Identifying Functionally Important Features with End-to-End Sparse Dictionary Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates: Red Hook, NY, USA, 2024; Volume 37.
13. Karvonen, A.; Rager, C.; Lin, J.; Tigges, C.; Bloom, J.; Chanin, D.; Lau, Y.-T.; Farrell, E.; McDougall, C.; Ayonrinde, K.; et al. SAEbench: A Comprehensive Benchmark for Sparse Autoencoders in Language Model Interpretability. *arXiv* **2025**, arXiv:2503.09532.
14. Leask, P.; Bussmann, B.; Pearce, M.T.; Bloom, J.; Tigges, C.; Al Moubayed, N.; Sharkey, L.; Nanda, N. Sparse Autoencoders Do Not Find Canonical Units of Analysis. In Proceedings of the International Conference on Learning Representations (ICLR), Singapore, 24–28 April 2025.
15. Paulo, G.; Shabalin, S.; Belrose, N. Transcoders Beat Sparse Autoencoders for Interpretability. *arXiv* **2025**, arXiv:2501.18823.
16. Kantamneni, S.; Engels, J.; Rajamanoharan, S.; Tegmark, M.; Nanda, N. Are Sparse Autoencoders Useful? A Case Study in Sparse Probing. *arXiv* **2025**, arXiv:2502.16681.
17. Korznikov, A.; Galichin, A.; Dontsov, A.; Rogov, O.; Oseledets, I.; Tutubalina, E. Sanity Checks for Sparse Autoencoders: Do SAEs Beat Random Baselines? *arXiv* **2026**, arXiv:2602.14111.
18. Beck, A.; Teboulle, M. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Imaging Sci.* **2009**, *2*, 183–202. [CrossRef]
19. Mairal, J.; Bach, F.; Ponce, J. Task-Driven Dictionary Learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *34*, 791–804. [CrossRef] [PubMed]
20. O’Neill, C.; Gumran, A.; Klindt, D. Compute Optimal Inference and Provable Amortisation Gap in Sparse Autoencoders. In *Proceedings of the 42nd International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2025; Volume 267, pp. 46877–46896.
21. OpenAI. Openai/Sparse_Autoencoder: Sparse Autoencoders Trained on GPT-2 Small Activations and Associated Visualizer. *Software Repository*. 2024. Available online: https://github.com/openai/sparse_autoencoder (accessed on 6 May 2026).
22. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language Models Are Unsupervised Multitask Learners. OpenAI Technical Report. 2019. Available online: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (accessed on 13 March 2026).
23. Merity, S.; Keskar, N.S.; Socher, R. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv* **2018**, arXiv:1803.08240.
24. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
25. Joachims, T. Optimizing Search Engines Using Clickthrough Data. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, AB, Canada, 23–26 July 2002; pp. 133–142. [CrossRef]
26. Marks, S.; Rager, C.; Michaud, E.J.; Belinkov, Y.; Bau, D.; Mueller, A. Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models. *arXiv* **2024**, arXiv:2403.19647.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.