Train One Sparse Autoencoder Across Multiple Sparsity Budgets to Preserve Interpretability and Accuracy

Nikita Balagansky[*♣], **Yaroslav Aksenov**[♣], **Daniil Laptev**[♣], **Vadim Kurochkin**[♣], **Gleb Gerasimov**[♣,♠], **Nikita Koriagin**[♣], **Daniil Gavrilov**[♣]
[♣]T-Tech, [♠]HSE University

Abstract

Sparse Autoencoders (SAEs) have proven to be powerful tools for interpreting neural networks by decomposing hidden representations into disentangled, interpretable features via sparsity constraints. However, conventional SAEs are constrained by the fixed sparsity level chosen during training; meeting different sparsity requirements therefore demands separate models and increases the computational footprint during both training and evaluation. We introduce a novel training objective, *HierarchicalTopK*, which trains a single SAE to optimise reconstructions across multiple sparsity levels simultaneously. Experiments with Gemma-2 2B demonstrate that our approach achieves Pareto-optimal trade-offs between sparsity and explained variance, outperforming traditional SAEs trained at individual sparsity levels. Further analysis shows that HierarchicalTopK preserves high interpretability scores even at higher sparsity. The proposed objective thus closes an important gap between flexibility and interpretability in SAE design.

1 Introduction

Transformers have revolutionised natural language processing (NLP) by achieving state-of-the-art performance across diverse tasks. Yet their internal representations remain notoriously difficult to interpret, often exhibiting *polysemanticity*, in which individual neurons activate for semantically unrelated features. To address this challenge, recent work has focused on Sparse Autoencoders (SAEs), which learn disentangled, human-interpretable directions in Transformer residual streams by enforcing sparsity constraints on the latent representations.

SAEs decompose hidden states into latent embeddings that are theoretically grounded in the independent additivity principle (Ayonrinde et al., 2024). This principle posits that individual features contribute to model behaviour independently, enabling isolated analysis of the latents. In practice, relaxing sparsity constraints (e.g. increasing the number of active latents) often introduces entanglement: latents begin to co-activate for unrelated features, undermining interpretability. Consequently, the effectiveness of existing SAEs is tightly coupled to a single sparsity level fixed during training.

We propose *HierarchicalTopK*, a novel activation mechanism and training objective that enables a single SAE to maintain interpretable features across a range of sparsity levels. Unlike conventional SAEs, which must be retrained to accommodate different sparsity requirements, our method ensures that any subset of latents with $k \leq K$ remains disentangled and faithful to the independent additivity principle. Empirically, HierarchicalTopK SAEs achieve Pareto-optimal trade-offs between sparsity and explained variance, outperforming traditional SAEs trained independently at varying sparsity levels. This work bridges the gap between flexibility and interpretability in SAE design, enabling dynamic adaptation to downstream tasks with varying computational or fidelity requirements.

2 Method

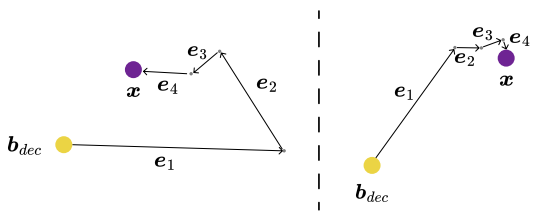


Figure 1: **Left:** SAE trained on a single k. **Right:** SAE trained on all $k \leq K$.

A sparse autoencoder (SAE) is defined as

$$\boldsymbol{l} = \sigma\big(W_{\text{enc}}\boldsymbol{x} + \boldsymbol{b}_{\text{enc}}\big),$$
$$\hat{\boldsymbol{x}} = W_{\text{dec}}\boldsymbol{l} + \boldsymbol{b}_{\text{dec}},$$

[*]Corresponding author: nikitabalagansky@gmail.com

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 10171–10179
November 4-9, 2025

where $W_{\text{enc}} \in \mathbb{R}^{D \times h}$, $W_{\text{dec}} \in \mathbb{R}^{h \times D}$, $\boldsymbol{b}_{\text{enc}} \in \mathbb{R}^{D}$, and $\boldsymbol{b}_{\text{dec}} \in \mathbb{R}^{h}$. Here, D is the dictionary size and h the hidden dimension. The non-linearity $\sigma(\cdot)$ is central. Vanilla SAEs use ReLU (Bricken et al., 2023), requiring an additional sparsity penalty on the latents. Sparsity can instead be induced directly with activations such as TopK (Makhzani and Frey, 2013) or BatchTopK (Bussmann et al., 2024). Our analysis focuses on these activation variants.

The decoder can be viewed as a set of embeddings $W_{\text{dec}} = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D]$, yielding

$$\hat{\boldsymbol{x}} = \sum_{i \in \text{top}_k} \boldsymbol{l}_i(\boldsymbol{x})\, \boldsymbol{e}_i + \boldsymbol{b}_{\text{dec}},$$

where $\boldsymbol{l}_i$ is the i-th component of $\boldsymbol{l}$. Embeddings are thus scaled by $\boldsymbol{l}_i(\boldsymbol{x})$ to reconstruct $\boldsymbol{x}$. The reconstruction error is $\mathcal{L}_{\text{rec}} = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$. Optimising $\mathcal{L}_{\text{rec}}$ for a fixed top_k can be suboptimal when one wishes to interpret individual directions (small k).

Hierarchical loss. We therefore introduce a *hierarchical* loss. Define

$$\hat{\boldsymbol{x}}_j = \sum_{i \in \text{top}_j} \boldsymbol{l}_i(\boldsymbol{x})\, \boldsymbol{e}_i + \boldsymbol{b}_{\text{dec}},$$
$$\mathcal{L}^j_{\text{rec}} = \|\boldsymbol{x} - \hat{\boldsymbol{x}}_j\|^2,$$

for $j \in \mathcal{J} \subset \mathbb{N}$ (e.g. $\mathcal{J} = \{1, \ldots, k\}$). The overall objective is

$$\mathcal{L}_{\text{hierarchical}} = \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathcal{L}^j_{\text{rec}}. \tag{1}$$

Whereas the standard SAE guarantees reconstruction only at k active embeddings, our formulation encourages good reconstructions for every $j \leq k$. The optimal model under $\mathcal{L}_{\text{hierarchical}}$ therefore improves $\hat{\boldsymbol{x}}_j$ monotonically with increasing j, a property absent in the vanilla SAE.

The hierarchical loss is inexpensive: it can be computed in a single forward pass via a cumulative-sum operation and implemented with kernels that avoid materialising intermediate tensors. In our implementation it runs faster than the original TopK loss; see Appendix C for details.

3 Experiments

3.1 Setup

For our experiments, we chose the Gemma-2 2B model (Gemma Team, 2024). We trained SAEs on a 1 B-token subsample of the FineWeb dataset (Penedo et al., 2024). Unless stated otherwise, we use the output of the 12th Transformer layer and set the SAE dictionary size to $D = 65\,536$. Training details are provided in Appendix A.

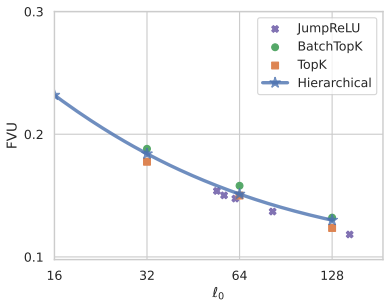


Figure 2: Comparison of an SAE with Hierarchical activation against other activation variants. The proposed method lies on the Pareto-optimal frontier across all sparsity levels, even though it is a single model.

We report the fraction of unexplained variance (FVU) as the main metric:

$$\text{FVU}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{\text{Var}(\boldsymbol{x} - \hat{\boldsymbol{x}})}{\text{Var}(\boldsymbol{x})}.$$

Sparsity is measured by the ℓ_0 norm

$$\ell_0 = \sum_i \mathbf{I}[\boldsymbol{l}_i > 0].$$

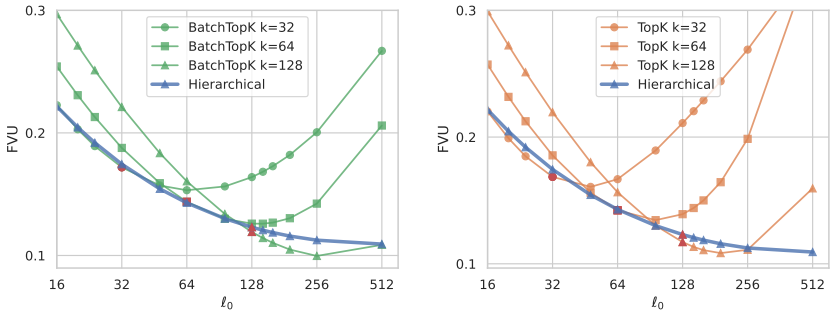
Because we use TopK-based activations, $\ell_0 = k$.

3.2 Hierarchical SAE Pareto Frontier

To evaluate the proposed training technique, we trained baseline SAEs at different sparsity levels. Specifically, we trained JumpReLU (Rajamanoharan et al., 2024) with various sparsity-regularisation coefficients and TopK and BatchTopK SAEs with $k \in \{32, 64, 128\}$. We also trained a single HierarchicalTopK SAE with $K = 128$. Figure 2 shows that our model matches or surpasses the performance of the individually trained baselines across all sparsity levels while requiring only one set of parameters.

3.3 Changing ℓ_0 at the Inference

To assess generalisation across sparsity levels, we trained a single HierarchicalTopK SAE with $K = 128$ and baseline TopK and BatchTopK SAEs with fixed $k \in \{32, 64, 128\}$. At inference we varied ℓ_0 over a dense grid, including both interpolation points within the training range and extrapolation points outside it. As shown in Figure 3, the Hierarchical model performs as well as—or better than—the baselines for $\ell_0 \leq 128$, demonstrating



(a) BatchTopK (b) TopK

Figure 3: Pareto frontier for SAEs with BatchTopK, TopK, and Hierarchical activation functions. Red dots denote the ℓ_0 values on which the BatchTopK and TopK SAEs were trained. HierarchicalTopK matches or surpasses separately trained BatchTopK and TopK SAEs when interpolating ($\ell_0 \leq 128$), allowing a single SAE to select ℓ_0 post-training. See Section 3.3 for details.

that training across multiple k values is crucial for robust performance.

BatchTopK mixes different k values between samples during training, resulting in a primitive form of extrapolation. Consequently, it continues to improve reconstructions for $\ell_0 \in [128, 512]$, a sparsity range rarely used in practice.

3.4 Pointwise Loss

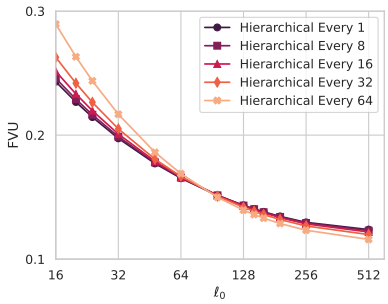


Figure 4: We test simple heuristics to reduce the computation required to train HierarchicalTopK. Computing the loss on every 8th term does not affect performance; see Section 3.4 for details.

To reduce computational overhead we evaluated computing the hierarchical loss on a subsampled index set (Equation 1):

$$J_x = \{1\} \cup \{i \in \mathbb{N} : i \bmod x = 0 \wedge 1 < i \leq k\},$$

with $x \in \{1, 8, 16, 32, 64\}$. As Figure 4 shows, computing the loss on every 8th term ($x = 8$) yields performance indistinguishable from the full loss, providing an eight-fold theoretical reduction in FLOPs.

An SAE whose loss is calculated on every 64th term suffers a significant performance decrease for $\ell_0 < 128$, but extrapolates better for $\ell_0 > 128$. Remarkably, using the hierarchical loss on every 8th term ($J_8 = \{1, 8, 16, 24, 32, \ldots, 128\}$) reduces theoretical overhead by a factor of eight without sacrificing reconstruction quality. In practice, however, there is almost no difference in per-step training time between vanilla TopK and HierarchicalTopK; see Appendix C for details.

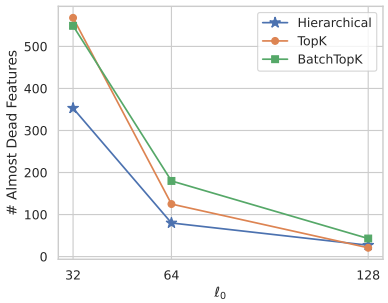
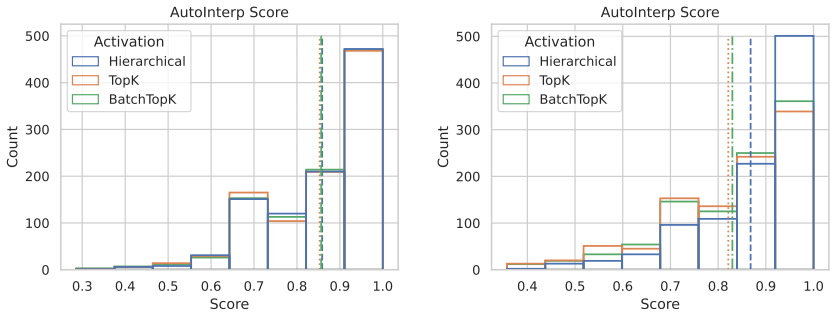


Figure 5: Number of features with activation frequency below 10^{-5} ("almost dead") for SAEs trained with $k = 128$. "Optimal scaling" denotes the number of almost-dead features in a BatchTopK SAE trained with $k = \ell_0$. The BatchTopK model accumulates almost-dead features more rapidly than the Hierarchical model when k is reduced at inference time; see Section 3.5 for details.

3.5 Why SAE Struggle to Reduce ℓ_0?

To investigate why simple SAE variants struggle to interpolate to lower ℓ_0 values than those used during training, we measured the number of features whose activation frequency falls below 10^{-5} (i.e. they activate once in 10^5 tokens). We call these features *almost dead*.

We trained TopK and BatchTopK models with



(a) $\ell_0 = 32$ (b) $\ell_0 = 128$

Figure 6: AutoInterp Score (Paulo et al., 2024). TopK and BatchTopK scores are obtained from two separate SAEs trained with $k = 32$ and $k = 128$; the Hierarchical model uses a single SAE trained on all $k \leq 128$. Hierarchical activation preserves the interpretability level of SAEs trained with smaller ℓ_0.

$k = 128$ and then, following Section 3.3, evaluated them with $k \in \{32, 64, 128\}$. The Hierarchical variant was trained once with $k = 128$. Results are shown in Figure 5. Although all SAEs exhibit similar numbers of dead features at $\ell_0 = 128$, the Hierarchical model keeps significantly more features alive than the TopK and BatchTopK variants as k decreases.

3.6 Interpretability

To validate interpretability we use the detection score of Paulo et al. (2024), implemented in SAE Bench (Karvonen et al., 2025). For TopK and BatchTopK we evaluate two SAEs trained with $k = 32$ and $k = 128$; for Hierarchical we evaluate a single SAE trained on all $k \leq 128$ (see Figure 6).

For the Hierarchical SAE the interpretability score at $\ell_0 = 128$ is almost identical to that at $\ell_0 = 32$, while its explained variance remains on the Pareto frontier (Figure 2). By contrast, in both TopK and BatchTopK variants, less-sparse models tend to be less interpretable. This observation underscores the superiority of the hierarchical loss.

4 Related Work

Research on sparse autoencoders increasingly focuses on feature interpretability in Transformer representations. The seminal work of Gao et al. (2025) introduced TopK-sparse autoencoders; Bussmann et al. (2024) extended this idea with batch-level sparsity control. However, these approaches lack a mechanism for establishing feature importance or relationships.

Structural constraints have also been explored. Bussmann et al. (2025) and Ayonrinde et al. (2024) investigate hierarchical dictionaries, demonstrating the benefits of progressive refinement. Building on these insights, our training method naturally encodes feature importance through progressive reconstruction, mirroring gradient-descent dynamics and feature hierarchies while maintaining interpretability and improving generalisation across sparsity levels.

5 Conclusion

We introduced *HierarchicalTopK*, a single sparse-autoencoder objective that enforces high-quality reconstructions at every sparsity level up to a chosen budget K. Experiments on Gemma-2 2B representations show that our approach:

- Achieves Pareto-optimal trade-offs between explained variance and ℓ_0 compared with independently trained TopK and BatchTopK baselines, despite using a single model.
- Maintains high interpretability across sparsity levels and prevents the proliferation of "dead" features when ℓ_0 is varied at inference time.

These contributions provide a flexible, efficient, and interpretable framework for analysing Transformer latent spaces under varying computational constraints.

6 Limitations

Our work has two principal limitations. (*i*) **Evaluation scope**: experiments are limited to the Gemma-2 2B model and a FineWeb subset; transfer to other architectures and datasets remains to be tested. (*ii*) **Interpretability measures**: we rely on automated metrics as proxies for human judgement; user studies are needed to validate semantic alignment.

References

Kola Ayonrinde, Michael T. Pearce, and Lee Sharkey. 2024. Interpretability as compression: Reconsidering sae explanations of neural activations with mdl-saes. *Preprint*, arXiv:2410.11179.

Vincent-Pierre Berges, Barlas Oguz, Daniel Haziza, Wen tau Yih, Luke Zettlemoyer, and Gargi Gosh. 2024. Memory layers at scale.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, and 6 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Bart Bussmann, Patrick Leask, and Neel Nanda. 2024. Batchtopk sparse autoencoders. *arXiv preprint arXiv: 2412.06410*.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. 2025. Learning multi-level features with matryoshka sparse autoencoders. *Preprint*, arXiv:2503.17547.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2025. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*.

Google DeepMind Gemma Team. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv: 2408.00118*.

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum McDougall, Kola Ayonrinde, Matthew Wearden, Arthur Conmy, Samuel Marks, and Neel Nanda. 2025. Saebench: A comprehensive benchmark for sparse autoencoders in language model interpretability. *arXiv preprint arXiv: 2503.09532*.

Alireza Makhzani and Brendan J. Frey. 2013. k-sparse autoencoders. *International Conference on Learning Representations*.

Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. 2024. Automatically interpreting millions of features in large language models. *arXiv preprint arXiv: 2410.13928*.

Guilherme Penedo, Hynek Kydlicek, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. 2024. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *Preprint*, arXiv:2407.14435.

A SAE Training Details

All SAEs were trained with a modified version of the code from Bussmann et al. (2024). Hyperparameters are listed in Table 1. We use NVIDIA H100 80GB GPU and spent about 20 GPU-days of compute, including preliminary experiments.

Parameter	Value
Optimizer	Adam
Learning Rate	0.0008
# tokens	10^9
Dataset	FineWeb
Batch Size	8096
Decoder Normalization	True

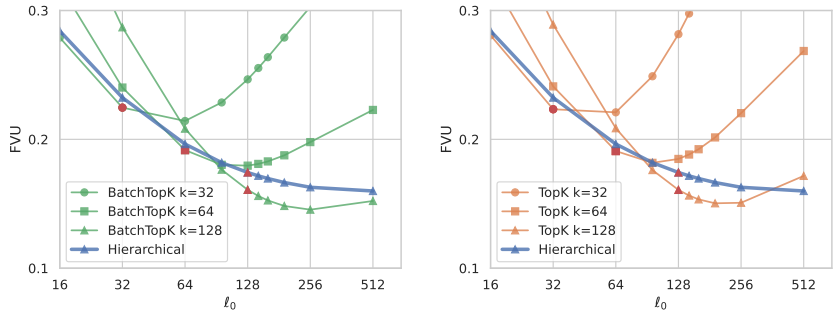
Table 1: Hyperparameters used to train the SAEs. See Section A for more details.

We also used modified kernels from Gao et al. (2025); see Appendix C for details.

B Additional Results

B.1 Qwen-2.5-7B results

Similar to the experiments in Section 3.3, we trained BatchTopK, TopK, and HierarchicalTopK SAEs on layer 14 of the Qwen-2.5-7B model. The results are shown in Figure 7.



(a) BatchTopK (b) TopK

Figure 7: Pareto frontier for SAEs with BatchTopK, TopK, and Hierarchical activation functions. Red dots denote the ℓ_0 values on which the BatchTopK and TopK SAEs were trained. HierarchicalTopK matches or surpasses separately trained BatchTopK and TopK SAEs when interpolating ($\ell_0 \leq 64$), however we found gap in performance for HierarchicalTopK in case of $\ell_0 = 128$.

Although the HierarchicalTopK SAE performs worse at larger ℓ_0 values, it achieves comparable or better performance for all $\ell_0 \leq 64$. Moreover, performance under larger sparsity budgets can be controlled via the pointwise loss (see Section 3.4).

B.2 Latent Structure

To support Figure 1 we measured the cosine similarity between feature embeddings in the reconstruction sum

$$\hat{\boldsymbol{x}} = \sum_{i \in \text{top}_k} \boldsymbol{l}_i(\boldsymbol{x})\, \boldsymbol{e}_i + \boldsymbol{b}_{\text{dec}},$$

In Figure 8, e_1 denotes the top-1 activation, and so on. Ideally, similarity should decrease monotonically as activation values diminish.

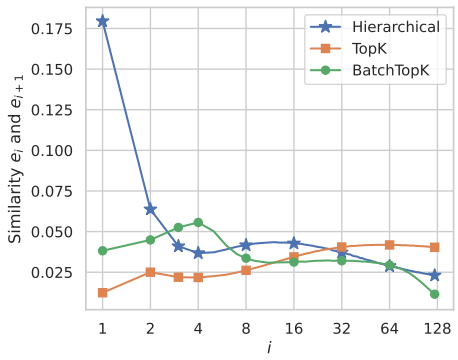


Figure 8: Cosine similarity of feature embeddings in the reconstruction sum.

Vanilla TopK SAEs show the undesired trend that similarity increases with the index i, whereas the Hierarchical model preserves the expected monotonic decrease.

B.3 Top-1 Features

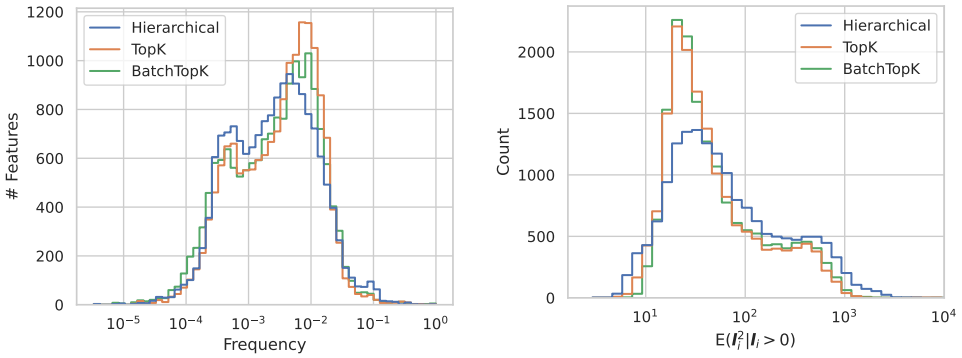
While vanilla Sparse Autoencoders are trained on top-k tokens, interpretation typically relies on the top-1 activated feature. Therefore, the performance of the top features is crucial for interpretability. We evaluated different levels of Explained Variance (EV) under small ℓ_0 budgets.

SAE	ℓ_0	EV (↑)
TopK (trained with $\ell_0 = 128$)	1	0.350 ± 0.091
HierarchicalTopK (trained with max $\ell_0 = 128$)	1	$\mathbf{0.457 \pm 0.091}$
TopK (trained with $\ell_0 = 128$)	2	0.442 ± 0.096
HierarchicalTopK (trained with max $\ell_0 = 128$)	2	$\mathbf{0.542 \pm 0.090}$

Table 2: Performance on top-1 and top-2 features.

Hierarchical variant significantly outperforms vanilla variant, which indicates better description obtained via AutoInerp Paulo et al. (2024).

B.4 Distribution of the Latents Activations



(a) Feature frequency

(b) Mean squared activation

Figure 9: Latent-feature distributions for SAEs trained with $k = 128$ ($J = \{1, \ldots, k\}$ in Hierarchical training).

To compare the distributions learned by standard SAEs and the Hierarchical variant we analyse both feature frequency and mean-squared activation (Figure 9). Hierarchical training yields more latents

with higher activation values (panel 9b), which may explain its superior interpretability. Its frequency distribution is skewed towards lower values, indicating that the hierarchical loss encourages activations to appear as the top-1 feature, enabling accurate reconstruction even at $k = 1$.

B.5 JumpReLU and TopK evaluations

A BatchTopK SAE is trained with batch-wise sparsity but, if evaluated directly with per-token TopK, the training and inference settings mismatch. We therefore apply a constant-threshold JumpReLU at inference time, choosing the threshold so that the expected number of active features equals k. To study the effect of switching activations we trained SAEs with $\ell_0 = 64$ (the Hierarchical model was trained on $k \leq 128$) and evaluated every model at $\ell_0 = 64$. Results are shown in Figure 10.

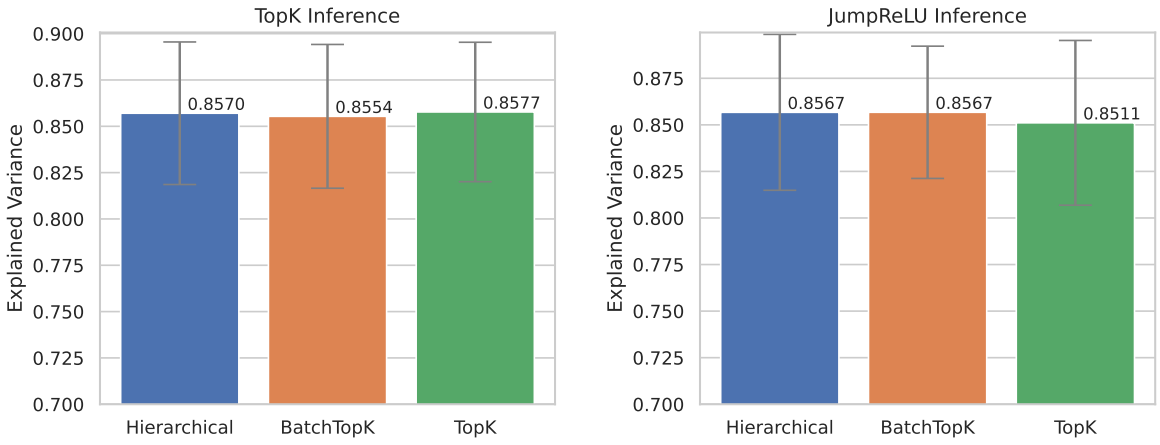


Figure 10: TopK versus JumpReLU inference. We did not find a significant difference between JumpReLU and fixed TopK evaluation.

The largest change in explained variance occurs for the TopK SAE, which drops from 0.8577 to 0.8511 under JumpReLU. BatchTopK improves marginally ($+0.0013$), and the Hierarchical variant is virtually unchanged for either activation.

Implementation	$K = 32$	$K = 64$	$K = 128$
Pure Torch-compiled			
TopK	8.79 ms / 2.92 GiB	11.75 ms / 2.92 GiB	18.88 ms / 2.93 GiB
Flex	12.82 ms / 6.29 GiB	23.38 ms / 10.79 GiB	43.85 ms / 19.80 GiB
TopK kernels			
xFormers (Berges et al., 2024)	5.58 ms / 2.92 GiB	6.34 ms / 2.92 GiB	7.96 ms / 2.93 GiB
OpenAI (Gao et al., 2025)	7.05 ms / 2.92 GiB	8.90 ms / 2.92 GiB	11.79 ms / 2.93 GiB
HierarchicalTopK kernels			
Triton HierarchicalTopK	**6.70 ms / 2.92 GiB**	**7.99 ms / 2.92 GiB**	**10.61 ms / 2.93 GiB**

Table 3: Training latency (time per step, ms) and peak memory (GiB) for different sparse-autoencoder implementations. Evaluations were run with dictionary size $F = 65536$ and embedding dimension $D = 2304$, at sparsity levels $K \in \{32, 64, 128\}$.

C Implementation

HierarchicalTopK is memory-intensive if implemented naively (even when using `torch.compile`). To address this, we implemented Triton kernels[1] that fuse the prefix-sum computation with the loss calculation; this fusion makes the additional overhead in both time and memory negligible.

We adapted embedding bag Triton kernels[2] to produce a TopK sparse-decoder implementation. As an additional baseline we use OpenAI's sparse-autoencoder[3] implementation.

As reported in Table 3, our HierarchicalTopK Triton kernel is substantially faster than a naïve `torch.compile` implementation while exhibiting comparable peak memory. Compared to previously published TopK Triton kernels, the HierarchicalTopK kernel achieves competitive latency at the evaluated settings. Crucially, these results explain why HierarchicalTopK requires specialized kernels: the naïve implementation suffers from much higher latency and considerably larger peak memory, which makes it impractical at scale. By contrast, the performance gap between a pure torch-compiled TopK and Triton TopK implementations is small.

For clarity, we also provide a minimal (naïve) PyTorch implementation of the HierarchicalTopK hierarchical loss. This reference implementation illustrates the core idea: gather decoder embeddings at the active indices, scale them by the sparse activations, compute cumulative reconstructions across the top-K entries, and measure the mean squared error across all sparsity levels.

Listing 1: Naïve PyTorch implementation of the Hierarchical loss.

```
def hierarchical_loss(sparse_idx, sparse_val, decoder, b_dec, target):
    """
    sparse_idx: LongTensor of shape (B, K) with indices of active embeddings
    sparse_val: FloatTensor of shape (B, K) with corresponding activation values
    decoder:     FloatTensor of shape (D, h) containing the dictionary embeddings
    b_dec:       FloatTensor of shape (h) containing decoder bias
    target:      FloatTensor of shape (B, h) with the original inputs
    """
    B, K = sparse_idx.shape
    flatten_idx = sparse_idx.view(-1)
    emb = decoder[flatten_idx].view(B, K, -1)
    emb = emb * sparse_val.unsqueeze(-1)
    recon_cum = emb.cumsum(dim=1) + b_dec.unsqueeze(1)
    diff = recon_cum - target.unsqueeze(1)
    total_err = diff.pow(2).mean()
    return total_err
```

[1] HierarchicalTopK kernel source code: https://github.com/corl-team/flexsae

[2] https://github.com/facebookresearch/memory/

[3] https://github.com/openai/sparse_autoencoder